Agilni razvoj web sustava za praćenje stanja oboljelih od dijabetesa u ovisnosti o izazvanom stresu

Džoić, Mario

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

Sveučilišni studij

Agilni razvoj web sustava za praćenje stanja oboljelih od dijabetesa u ovisnosti o izazvanom stresu

Diplomski rad

Mario Džoić

Osijek, 2021.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Obrazac D1: Obrazac za imenovanje Povjerenstva za diplomski ispit

Osijek, 30.08.2021.

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za diplomski ispit

Ime i prezime studenta:	Mario Džoić
Studij, smjer:	Diplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina	D-1050R, 06.10.2019.
OIB studenta:	16177109834
Mentor:	Prof.dr.sc. Goran Martinović
Sumentor:	
Sumentor iz tvrtke:	izv.prof.dr.sc. Tatjana Bačun
Predsjednik Povjerenstva:	Izv. prof. dr. sc. Krešimir Nenadić
Član Povjerenstva 1:	Prof.dr.sc. Goran Martinović
Član Povjerenstva 2:	Izv. prof. dr. sc. Ivica Lukić
Naslov diplomskog rada:	Agilni razvoj web sustava za praćenje stanja oboljelih od dijabetesa u ovisnosti o izazvanom stresu
Znanstvena grana rada:	**Programsko inženjerstvo (zn. polje računarstvo)**
Zadatak diplomskog rada:	U teorijskom dijelu diplomskog rada potrebno je opisati i analizirati utjecaj stresa i kretanja na dijabetes, postupke umjetnog izazivanja stresa, te načine i parametre praćenja dijabetesa i kretanja oboljelih od dijabetesa. Također, treba predložiti postupke izazivanja stresa prilagodljivog intenziteta i trajanja zasnovane na aritmetičkim i sličnim načelima, načine praćenja razine glukoze u krvi uz pomoć
Prijedlog ocjene pismenog dijela ispita (diplomskog rada):	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 3 bod/boda Jasnoća pismenog izražavanja: 3 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene mentora:	30.08.2021.

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

IZJAVA O ORIGINALNOSTI RADA	
Osijek, 08.09.2021.	
Ime i prezime studenta:	Mario Džoić
Studij:	Diplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	D-1050R, 06.10.2019.
Turnitin podudaranje [%]:	10

Ovom izjavom izjavljujem da je rad pod nazivom**: Agilni razvoj web sustava za praćenje stanja oboljelih od dijabetesa u ovisnosti o izazvanom stresu**

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

i sumentora

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

Potpis studenta:

Sadržaj

1 UVOD

Dijabetes se smatra bolešću modernog doba te zahvaća sve veći broj stanovništva zbog modernog načina života. To je stanje koje nastaje zbog nedostatka inzulina ili rezistencije na njega što dovodi do visoke razine glukoze u krvi. Povišena razina glukoze u krvi dovodi do raznih oštećenja i oboljenja koji, ako nisu kontrolirani, mogu biti opasni po život. Postoje dva tipa dijabetesa, od kojih je jedan uzrokovan genetikom, dok drugi nastaje zbog načina života. Dijabetes tipa dva može se odgoditi ili spriječiti promjenom životnih navika, u vidu zdrave prehrane i povećanja tjelesne aktivnosti. Na razinu glukoze također utječe stres, pa je stoga bitno naučiti podnositi stres na što bolji način kako bi se smanjio učinak istoga. Moderna tehnologija i računalni uređaji mogu značajno pomoći i pružiti potporu u borbi protiv bolesti.

Tema ovog diplomskog rada je razvoj web sustava za praćenje oboljelih od dijabetesa u ovisnosti o izazvanom stresu. Web aplikacija će nuditi korisniku umjetno izazivanje stresa pomoću aritmetičkih metoda, te će korisnik moći pratiti na koji način je izazvani stres utjecao na razinu glukoze u krvi. Sustav omogućuje unos podataka o kretanju, kao i podatke o razini glukoze u krvi, te će se na taj način moći vidjeti međusobni utjecaj. Web aplikacija će biti izrađena koristeći principe agilnog razvoja i pratit će se razvoj programske podrške pokretanim testiranjem.

U drugom poglavlju opisan je dijabetes i problemi s kojima se susreću oboljeli od dijabetesa. Također, pojašnjeni su utjecaji stresa i kretanja na dijabetes. Nadalje, objašnjene su metode umjetnog izazivanja stresa zasnovane na aritmetičkim načelima i objašnjen je način praćenja razine glukoze u krvi pomoću sustava Libre i praćenja fizičke aktivnosti uz pomoć pametne narukvice. U trećem poglavlju objašnjeni su principi agilnog razvoja te su navedene neke od mnogih metodologija agilnog razvoja. Način razvoja programske podrške pokretan testiranjem te utjecaj agilnog razvoja i razvoja pokretanim testiranjem bit će opisani u četvrtom poglavlju. U petom poglavlju opisane su programske tehnologije i okviri korišteni prilikom izrade web aplikacije, te način razvoja i korištenja same aplikacije.

2 UTJECAJ STRESA NA DIJABETES I PREGLED STANJA U PODRUČJU

Tijekom posljednja dva desetljeća broj oboljelih od dijabetesa uvelike je porastao. Glavni razlozi tome su visokokalorična prehrana, koja je povezana s prekomjernom tjelesnom težinom. Procjenjuje se da je 2019. godine 463 milijuna osoba u dobi od 20 do 79 godina živjelo s dijabetesom. Čak polovica istih nije imala postavljenu dijagnozu, stoga se nisu ni liječili. Nešto više od 4 milijuna osoba godišnje umre od ove bolesti, a svaka druga ima manje od 60 godina. Troškovi liječenja dijabetesa na svjetskoj razini iznose više od 700 milijardi dolara.

Dijabetes je bolest koja izaziva povišenu razinu glukoze u krvi. Inzulin je jedan od osnovnih hormona koji se proizvodi u gušterači. Dozvoljava glukozi, koja se nalazi u krvi, protok u stanice tijela gdje se pretvara u energiju. Nedostatak inzulina dovodi do visokih razina glukoze, što se još naziva hiperglikemija, koja je klinički pokazatelj dijabetesa. Ako se nedostatak inzulina ne tretira, dugoročno može oštetiti mnoge organe u ljudskom tijelu i dovesti do komplikacija kao što su kardiovaskularne bolesti, oštećenje živaca, oštećenje bubrega, gubitak vida itd. Međutim, ako se dijabetes kontrolira, može se odgoditi ili spriječiti pojavljivanje komplikacija.

Prema tablici 2.1 preuzeto iz [1], ako dva mjerenja glukoze natašte iznose 7,0 i više mmol/L (126mg/dL), smatra se da osoba ima dijabetes. Osobe s razinom glukoze natašte od 6.1 do 6.9 mmol/l (110 do 125 mg/dL) imaju poremećenu razinu glukozu. Osobe s glukozom sa ili iznad 7.8 mmol/l (140 mg/dL), ali ne preko 11,1 mmol/l (200mg/dL), dva sata nakon jela, imaju problem u toleranciji glukoze. Zadnja dva stanja nazivaju se preddijabetesom, od kojih je posljednji glavni rizik za napredovanje u potpuni dijabetes. Oba stanja potrebno je rano dijagnosticirati i tretirati, jer mogu dovesti do razvoja dijabetesa tipa dva.

Tablica 2.1. Klasifikacija razine glukoze [1].

Stanje	Dva sta nakon jela		Natašte	
Jedinica	mmol/L	mg/dL	mmol/L	mg/dL
Normalno	< 7.8	< 140	< 6.1	< 110
Poremećena glukoza natašte	< 7.8	< 140	6.1–7.0	110–125
Intolerancija glukoze	≥ 7.8	≥ 140	< 7.0	< 126
Dijabetes	≥ 11.1	≥ 200	≥ 7.0	≥ 126

Prema [1], postoje dva glavna tipa dijabetesa, a to su dijabetes tipa jedan i dijabetes tipa dva. Tip jedan dijabetesa je izazvan autoimunom reakcijom u kojoj obrambeni sustav tijela napada stanice gušterače koje proizvode inzulin. Kao rezultat toga, tijelo proizvodi nedovoljno ili nikako inzulina. Može se pojaviti u bilo kojoj dobi, ali najčešće se javlja kod djece i mladih osoba. Osobe s dijabetesom tipa jedan trebaju dnevno unositi inzulin kako bi održavale razinu glukoze u normalnom rasponu, a bez njega ne bi preživjele. Pomoću inzulina, praćenjem glukoze i zdravim načinom života mogu se odgoditi ili preventirati mnoge komplikacije.

Nastanak je obično iznenadan i uključuje sljedeće simptome:

- Često mokrenje
- Prekomjernu žeđ i suhoću usta
- Izraziti umor ili nedostatak energije
- Stalnu glad
- Nagli gubitak težine
- Probleme s vidom

Dijabetes tipa dva nastaje zbog nesposobnosti gušterače da proizvodi dovoljnu količinu inzulina ili zbog neučinkovitog korištenja inzulina. Ujedno, ovo je i najčešči oblik dijabetesa te ga ima oko 90% ljudi dijagnosticiranih s dijabetesom. Obično se javlja u starijoj dobi, te se u većini slučajeva otkrije kada dođe do komplikacija. Ako se osoba pravilno hrani, fizički je aktivna te koristi lijekove, dijabetes se može držati pod kontrolom. Simptomi su mnogo blaži i teže ih je dijagnosticirati nego kod tipa jedan. Kod trudnica se može javiti gestacijski dijabetes, iako ga do tada nisu imale, a prestaje nakon trudnoće [1].

Dijabetes izaziva komplikacije, koje se mogu podijeliti na akutne i kronične.

Akutne komplikacije dijabetesa zbog ekstremno visokih razina glukoze su:

- Dijabetička ketoacidoza – obično će se javiti ako tijelo ima premalo inzulina i može utjecati na ljude s dijabetesom tipa jedan, ljude koji su uklonili gušteraču ili ljude s dijabetesom tipa dva koji proizvode vrlo malo vlastitog inzulina.
- Hipoglikemija – stanje preniske razine glukoze. Nastupa kada je razina glukoze u krvi ispod 4.0 mmol/l. Simptomi su umor, slabost, zbunjenost i povećani otkucaji srca.
- Hiperglikemija – stanje u kojem je razina glukoze previsoka.
- Neketotični hiperosmolarni sindrom – događa se samo kod osoba s dijabetesom tipa dva. Nastaje zbog jake dehidracije i vrlo visoke razine glukoze.

Kronične komplikacije dijabetesa dijele se na vaskularne i nevaskularne komplikacije:

- Očne bolesti – neki ljudi s dijabetesom razviju dijabetičku retinopatiju. Ako se retinopatija rano prepozna, može se liječiti i tako spriječiti gubitak vida.
- Problemi sa stopalima –osobe s dijabetesom imaju veći rizik od ozbiljnih problema sa stopalima. Povišena razina glukoze u krvi može oštetiti cirkulaciju, što usporava zacjeljivanje čireva i posjekotina.
- Srčani problemi – dugotrajna povišena glukoza u krvi može oštetiti krvne žile, zbog čega može doći do srčanog i moždanog udara.
- Problemi s bubrezima – nekontrolirano stanje dijabetesa tijekom dužeg vremenskog razdoblja može dovesti do dijabetičke nefropatije, odnosno oštećenja bubrega.
- Oštećenje živaca – kod nekih ljudi s dijabetesom s vremenom dolazi do oštećenja živaca. Kao posljedica toga, javljaju se problemi s kretanjem i vidom jer se otežava prijenos poruka između mozga i svakog dijela tijela.
- Bolesti desni i drugi problemi s ustima – visoka razina šećera u krvi može dovesti do pojave bakterija u slini, čija kiselina napada zubnu caklinu i oštećuje zubno meso [2].

A1C test je uobičajeni test za dijagnozu dijabetesa tipa jedan i tipa dva. Ako je dijabetes već dijagnosticiran, može se koristiti za praćenje razina glukoze u krvi. A1C test rezultati pokazuju prosječnu razinu glukoze u krvi zadnja dva ili tri mjeseca. Što je veća razina A1C, to je lošija kontrola glukoze te je veći rizik od komplikacija dijabetesa.

A1C test pomaže liječniku:

- Dijagnosticirati preddijabetes
- Dijagnosticirati dijabetes tipa jedan i tipa dva
- Pratiti liječenje dijabetesa

Koliko često se provodi A1C test, ovisi o tipu dijabetesa i načinu liječenja dijabetesa. Npr., test se može provoditi jedanput godišnje ako je dijagnosticiran preddijabetes, dva puta godišnje ako se ne koristi inzulin i ako su razine glukoze stalno u potrebnim razinama, te četiri puta godišnje ako se koristi inzulin i ako postoje problemi s održavanjem razine glukoze u potrebnim razinama. Veći postotak A1C predstavlja veću prosječnu razinu glukoze u krvi. Razina ispod 5.7% smatra se normalnom, između 5.7% i 6.4% predstavlja preddijabetes, a 6.5% i više, u dva različita testa, predstavlja dijabetes. Tablica 2.2 preuzeta iz [3], prikazuje razine A1C testa [3].

Tablica 2.2. Razine A1C testa [3].

A1C razina	Prosječna razina glukoze
6%	126 mg/dL (7 mmol/L)
7%	154 mg/dL (8.6 mmol/L)
8%	183 mg/dL (10.2 mmol/L)
9%	212 mg/dL (11.8 mmol/L)
10%	240 mg/dL (13.4 mmol/L)
11%	269 mg/dL (14.9 mmol/L)
12%	298 mg/dL (16.5 mmol/L)

2.1 Utjecaj stresa na dijabetes

Prema [4], postoje mnoga istraživanja o utjecaju stresa na razinu glukoze kod osoba oboljelih od dijabetesa. Proveden je niz laboratorijskih studija o učincima određenih stresnih situacija na razinu glukoze u krvi, kao što su rješavanje aritmetičkih problema ili neugodnih intervjua. Mnoge od ovih studija pokazale su da ove vrste izazivanja stresa mogu destabilizirati razinu glukoze u krvi. Glavni nedostatak ovih pristupa je taj što ne odražavaju stvarni svijet u kojem osobe žive. Druge studije odnosile su se na stvarni svijet i pokušale su izmjeriti stres koji se javlja u prirodi. Rezultati su pokazali da su oni, kojima se glikemijska kontrola pogoršavala s vremenom, prijavili negativni stres, dok su oni, kojima se razina glukoze poboljšavala, prijavili pozitivan stres. [4] navode da negativan stres predstavlja međuljudske sukobe, smrt bliske osobe i sl., dok pozitivan predstavlja zaruke, rođenje djeteta, željene promjene u zaposlenju i sl. [4].

Stres je način na koji tijelo i um utječu na novu ili tešku situaciju. Može biti kratkoročan, kao što je briga o prezentaciji i slično, ali i dugoročan stres kao što su briga o novcu, vezi ili nekome. Za vrijeme stresa tijelo ispušta hormone kao što su kortizol i adrenalin. Ti hormoni tijelu daju energiju za obranu, ali i otežavaju rad inzulina. Zbog toga energija ne može pristupiti stanicama tijela, a kao posljedica toga raste razina šećera. Dužem izlaganju stresu, visoka razina šećera u krvi ostaje dugoročno, što povećava rizik od pojave dijabetesa. Sama dijagnoza dijabetesa izaziva stres, pogotovo u ranim danima dijagnoze. Stalna potreba za praćenjem razine šećera u krvi, kao i načina prehrane, može biti teška i zahtjevna. Visoka razina hormona stresa može zaustaviti pravilno funkcioniranje stanica gušterače, koje proizvode inzulin te smanjiti količinu inzulina koji se stvara, što doprinosi razvoju dijabetesa tipa dva. Prejedanje tijekom stresa jedan je od načina razvijanja dijabetesa tipa dva, jer neki ljudi na stres reagiraju jedenjem. Svi se nose sa stresnim situacijama na različite načine, stoga je potrebno naučiti se nositi sa stresom.

Dovoljna količina sna, tjelovježba, odmor, relaksacija i razgovor o tome što nas čini stresnim mogu pomoći u boljem podnošenju stresa [5].

Osobe s dijabetesom tipa jedan mogu imati različite odgovore na stres, što znači da može doći do povećanja ili snižavanja razine glukoze. Kada je osoba pod fizičkim stresom, također može doći do povećanja razine šećera u krvi, što utječe na oba tipa dijabetesa. Utjecaj stresa na razinu glukoze može se provjeriti praćenjem informacija kao što su datum i vrijeme, te što je rađeno u to vrijeme. Npr., ako je osoba pod stresom ponedjeljkom ujutro, može promijeniti rutinu kako bi smanjila stres. Razinu stresa može ocijeniti od 1 do 10, gdje razina deset predstavlja najveću razinu stresa. Zatim treba provjeriti razinu glukoze u krvi. Ovakav način praćenja treba primjenjivati narednih nekoliko tjedana. Ako se primijeti da je razina glukoze visoka u dane kada je osoba pod stresom, to znači da stres negativno utječe na razinu šećera u krvi [6].

Simptomi stresa su:

- Glavobolja
- Bol mišića i napetost
- Dugo ili kratko spavanje
- Generalno loš osjećaj
- Umor

Za vrijeme stresa osoba se može osjećati:

- Demotivirano
- Depresivno
- Anksiozno

2.2 Utjecaj kretanja na dijabetes

Uvođenje i održavanje fizičke aktivnosti od presudne je važnosti za održavanje razine šećera u krvi, kao i općenitog zdravlja svakog pojedinca s dijabetesom ili preddijabetesom. Prema [7], fizička aktivnost uključuje sve vrste pokreta koji povećavaju korištenje energije, dok vježbanje podrazumijeva planiranu i strukturiranu fizičku aktivnost. Vježbanje poboljšava zdravlje kod osoba s dijabetesom tipa dva na način da reducira kardiovaskularne bolesti, smanjuje težinu te poboljšava sveukupno zdravlje. Redovno vježbanje može preventirati ili odgoditi razvoj dijabetesa tipa dva. Također, ima zdravstvene benefite kod ljudi s dijabetesom tipa jedan kao što su poboljšani kardiovaskularni sustav, mišićna snaga, osjetljivost na inzulin itd. Tjelesna

aktivnost i vježbanje trebali bi se prilagoditi specifičnim potrebama svakog pojedinca. Trening snage uključuje vježbe tijelom, vježbe sa spravama ili s elastičnim trakama.

2.2.1 Koristi aerobnih vježbi

Prema [7], aerobne vježbe uključuju ponavljajuće i kontinuirane pokrete velikih skupina mišića. Aktivnosti kao što su hodanje, bicikliranje, jogiranje i plivanje fokusiraju se na aerobnu energiju. Aerobni treninzi povećavaju mitohondrijsku gustoću, osjetljivost na inzulin, oksidativne enzime, usklađenost i reaktivnost krvnih žila, plućne funkcije, imunološke funkcije i srčani učinak. Umjerene do velike količine aerobnih aktivnosti povezane su sa znatno nižim rizicima od kardiovaskularnog i ukupnog mortaliteta kod dijabetesa tipa jedan i tipa dva. Kod dijabetesa tipa jedan, aerobni trening povećava kardiorespiratornu sposobnost, smanjuje rezistenciju na inzulin i poboljšava razinu lipida i funkciju endotela. Kod osoba s dijabetesom tipa dva, redoviti trening smanjuje A1C razine, trigliceride, krvni tlak i rezistenciju na inzulin. Intervalni trening visokog intenziteta potiče brzo povećanje oksidacijskog kapaciteta skeletnih mišića, osjetljivosti na inzulin i kontrolu razine glukoze u krvi kod odraslih osoba s dijabetesom tipa dva te se može izvoditi bez pogoršanja razine glukoze kod dijabetesa tipa jedan.

2.2.2 Prednosti vježbe otpora

Dijabetes je neovisni čimbenik rizika za nisku mišićnu snagu i ubrzani pad mišićne snage i funkcionalnog statusa. Zdravstvene prednosti treninga otpora za sve odrasle uključuju poboljšanje mišićne mase, tjelesne građe, snage, tjelesne funkcije, mentalnog zdravlja, mineralne gustoće kostiju, osjetljivosti na inzulin, krvnog tlaka, lipidnih profila i kardiovaskularnog zdravlja. Učinak vježbanja otpora na kontrolu razine glukoze kod dijabetesa tipa jedan nije jasan. Kada se otpor snage i aerobne vježbe izvode u jednom vježbanju, izvođenje vježbi otpora prvo rezultira manjom hipoglikemijom nego kad se prvo izvode aerobne vježbe. Pogodnosti treninga otpora za osobe s dijabetesom tipa dva uključuju poboljšanje kontrole razine glukoze, inzulinske rezistencije, udjela masnoće, krvnog tlaka, snage i poboljšanje tjelesne mase.

2.2.3 Prednosti drugih vrsta tjelesne aktivnosti

Vježbe fleksibilnosti i ravnoteže važnije su za starije odrasle osobe s dijabetesom jer poboljšavaju mobilnost i koordinaciju. Često je prisutna ograničena pokretljivost zglobova, što rezultira stvaranjem glikacije, koja se nakuplja tijekom normalnog starenja, a ubrzava se stvaranje hiperglikemijom. Istezanje povećava opseg pokreta oko zglobova i fleksibilnost, ali ne

utječe na kontrolu glukoze. Trening ravnoteže može smanjiti rizik od padova, čak i kada je prisutna periferna neuropatija.

2.2.4 Tjelesna aktivnost i dijabetes tipa jedan

Osobe oboljele od dijabetesa tipa jedan mogu imati koristi od tjelesne aktivnosti, iako se aktivnost svima preporučuje. Tjelesna aktivnost kod osoba s dijabetesom tipa jedan različito utječe na razinu glukoze u krvi, ovisno o vrsti aktivnosti i vremenu izvođenja. Dodatni unos ugljikohidrata i smanjenje inzulina obično su potrebni za održavanje uravnotežene razine glukoze u krvi tijekom i nakon tjelesne aktivnosti. Za točan unos ugljikohidrata i doza inzulina, osobe trebaju raditi česte provjere razine glukoze u krvi. Kontinuirano praćenje glukoze tijekom tjelesne aktivnosti može pomoći u otkrivanju hipoglikemije.

2.2.5 Tjelesna aktivnost i dijabetes tipa dva

Prema [7], preporuke za osobe oboljele od dijabetesa tipa dva su:

- svakodnevno vježbanje, kako bi se pojačalo djelovanje inzulina
- izvođenje aerobnih treninga i treninga otpora za optimalne glikemijske i zdravstvene ishode
- strukturirane intervencije u načinu života, koje uključuju najmanje 150 minuta tjedne tjelesne aktivnosti te zdrave prehrambene navike, kako bi se spriječila ili odgodila pojava dijabetesa tipa dva kod osoba s visokim rizikom i preddijabetesom

Redoviti trening povećava gustoću kapilara mišića, oksidacijski kapacitet, metabolizam lipida i proteine koji signaliziraju inzulin, a koji su reverzibilni s prestankom treniranja. I aerobni trening, kao i trening otpora, promiču prilagodbe u koštanim mišićima, masnom tkivu i jetri povezane s pojačanim djelovanjem inzulina, čak i bez gubitka težine. Redoviti aerobni trening povećava osjetljivost na inzulin u mišićima, kod osoba s preddijabetesom i dijabetesom tipa dva, proporcionalno volumenu vježbanja. Stoga bi odrasli s dijabetesom tipa dva trebali izvoditi i aerobne treninge i vježbe otpornosti radi optimalnih glikemijskih i zdravstvenih rezultata.

2.2.6 Sjedilački način života

- Sve bi odrasle osobe, a posebno one s dijabetesom tipa dva, trebale smanjiti količinu vremena provedenog u svakodnevnom sjedilačkom načinu života

- Dugotrajno sjedenje treba prekinuti promjenom intenziteta svjetlosti svakih 30 minuta zbog poboljšanja razine glukoze u krvi

Gornje dvije preporuke su dodatak, a ne zamjena za pojačano strukturirano vježbanje i kretanje. Sjedilački način života budno je ponašanje s niskom potrošnjom energije (gledanje televizije, rad za stolom itd.) te ima značajan utjecaj na kardiometaboličko zdravlje u cijeloj populaciji. Veća prisutnost sjedilačkog načina života povezana je s povećanom smrtnošću i morbiditetom. Kod osoba s dijabetesom ili s rizikom za razvoj dijabetesa tipa dva, produljeno vrijeme sjedenja također je povezano s lošijom kontrolom glikemije i skupnim metaboličkim rizikom. Dugotrajno sjedenje prekinuto kratkim (≤5 min) stajanjem ili promjenom intenziteta svjetlosti, svakih 20–30 min, poboljšava kontrolu glikemije. Također, prekid duljeg sjedenja kratkim hodanjem i jednostavnim aktivnostima poboljšava kontrolu glikemije.

2.2.7 Djelovanje inzulina i tjelesna aktivnost

Prema [7], djelovanje inzulina u mišićima i jetri može regulirati vježbanjem i redovitim tjelesnim aktivnostima. Aerobne vježbe do pet puta povećavaju unos glukoze u mišićima putem mehanizama neovisnih o inzulinu. Ako je intenzitet povremeno povišen na gotovo maksimalan napor, poboljšano djelovanje inzulina može trajati 24 sata nakon kraćih aktivnosti (od 20 minuta). Čak i aerobna tjelovježba niskog intenziteta, u trajanju od ≥60 min, pojačava djelovanje inzulina u pretilih osoba otpornih na inzulin tijekom najmanje 24 sata. Ako je poboljšanje djelovanja inzulina primarni cilj, tada je optimalno svakodnevno vježbanje umjerenog ili visokog intenziteta [7].

2.3 Umjetno izazivanje stresa aritmetičkim metodama

Proučavanje stresa uključuje dizajn i provedbu metoda za indukciju stresa koji jamče njegovu manipulaciju kao neovisnu varijablu te omogućuje uspostavu odnosa utjecaja i uzročnosti. Razvoj eksperimenta s ljudima zahtijeva dizajn nezavisnih postupaka koji osiguravaju ograničene učinke. Ti učinci generiraju mjerenu razinu stresa, dovoljnu za promatranje njihovih učinaka, bez stvaranja štete sudionicima. Treba uključivati elemente novosti, nepredvidljivosti te nedostatka kontrole. Postoji više varijacija umjetnog izazivanja stresa aritmetičkim metodama. Neke od njih će biti kratko objašnjeni u ovom odlomku.

Aritmetički test izazivanja stresa nije standardiziran, stoga postoji više varijacija i načina provedbe ovog testa. U nastavku će biti opisani neki od već korištenih varijacija aritmetičkih metoda. Često se aritmetički test provodi u kombinaciji s nekim drugim testom.

Prema [8], provođen je aritmetički test na način da osobe dva sata prije testiranja nisu konzumirale hranu, alkohol te duhanske i kofeinske proizvode. Kako bi se odredio izazvani stres, pratili su se otkucaji srca i krvni tlak. Sudionici su serijski oduzimali broj 17 od broja 1000 u vremenskom razdoblju od tri minute. Razlika između prosječne vrijednosti početnog krvnog tlaka i maksimalnog krvnog tlaka tijekom razdoblja stresa, definirana je kao odgovor testa. Rezultati testa pokazuju da su krvni tlak i otkucaji srca bili značajno veći nego prije samog testa [8].

Nadalje, prema [9], sudionici su prestali s konzumacijom alkohola i kofeina dvanaest sati prije testa. Imali su usmeni 5-minutni test aritmetike, koji se sastoji od uzastopnog oduzimanja višeznamenkastih brojeva u što kraćem vremenskom roku. Kako bi se nametnuo osjećaj hitnosti, korišten je uređaj metronom koji pokazuje brzinu izvođenja i postavljen je na 30 otkucaja u minuti. Ispitanici su dobivali zvučni signal u slučaju pogreške. Također, ispitanici su dobili zvučni signal i novi broj ako su dali više od pet točnih uzastopnih odgovara [9].

Prema [10], za izazivanje stresa provodile su se metode mentalne aritmetike i stresnog intervjua. Sudionici nisu smjeli konzumirati alkohol, kofein i duhan 30 minuta prije testa. Trebali su završiti oba zadatka u istom danu, ali sa 7-minutnom pauzom između i nakon zadataka. Oba zadatka provodila su se nasumičnim redoslijedom. Tijekom aritmetičkog izazivanja stresa, sudionici su određivali različite standardne aritmetičke kalkulacije. Tijekom dvije minute trebali su riješiti aritmetike kao što su zbrajanje, oduzimanje i množenje, a nakon toga su tijekom 3 minute ponavljali niz brojeva obrnutim redoslijedom te su na kraju od ponuđenog broja oduzimali 7 ili 13. Tijekom aritmetičkog testa, sudionici su trebali odgovarati brzo i točno, a uređajima su mjerena brzina otkucaja srca i krvni tlak [10].

Prema [11], aritmetički test trajao je dvije minute. Sudionici su imali vremenski rok od tri sekunde. Ako ispitanik nije dao odgovor, to se bilježilo kao netočan odgovor. Nakon svakog odgovora, ispitanik je dobio povratnu informaciju o točnosti odgovora s različitim zvučnim signalima. U dvije minute sudionici su dobili 40 različitih aritmetičkih zadataka, a nakon završetka dobili su povratnu informaciju o uspješnosti. Sa svim točnim odgovorima dobili bi 400 bodova. Da bi prošli test, trebali su dati 80% točnih odgovora, odnosno dobiti 320 bodova.

Nakon aritmetičkog testa, sudionici su bili podvrgnuti upitniku agresivnosti i skali anksioznosti [11].

2.4 Sustav Libre

Nadzor glukoze u krvi prati se uređajem za ispitivanje razine glukoze u krvi. Test glukoze obično se provodi vađenjem krvi, zatim se krv nanosi na kemijski aktivnu trakicu za jednokratnu upotrebu. Različiti proizvođači koriste različite tehnologije, ali većina sustava mjeri električne karakteristike te pomoću njih određuje razinu glukoze u krvi. Test se obično naziva kapilarna glukoza u krvi. Nadzor glukoze u krvi pomaže u planiranju obroka i aktivnosti te određivanju doba dana u kojem treba uzimati lijekove. Ispitivanje omogućuje brzi odgovor na povišenu ili sniženu razinu glukoze u krvi, te se na taj način može pravovremeno reagirati. Kontinuirani sustavi praćenja glukoze sastoje se od:

- Senzora glukoze postavljenog neposredno ispod kože koji se nosi nekoliko dana do zamjene
- Veze od senzora do uređaja
- Uređaja koji prikazuje razinu glukoze uz kontinuirano ažuriranje

Kontinuirani sustavi mjere razinu glukoze u uzorku međustanične tekućine. Nedostatci ovakvih sustava su:

- Moraju se kalibrirati tradicionalnim mjerenjem glukoze u krvi
- Razina glukoze u međustaničnoj tekućini odstupaju za vrijednostima glukoze u krvi [12]

Libre sustav je došao kao potpuno novi koncept praćenja glukoze u krvi, pružajući mnogo više podataka od kontinuiranih sustava. Omogućuje brzo praćenje glukoze pomoću očitavanja glukoze skeniranjem senzora, a ne bockanjem prsta.

Prednosti sustava Libre su:

- Smanjena potreba za testiranjem preko krvi
- Pruža grafički prikaz promjene razine glukoze
- Skeniranje senzora pokazuje u kojem se smjeru rezultati mijenjaju
- Povoljniji su od kontinuiranih sustava praćenja glukoze

Senzor je vodootporan do razine 1 metra tijekom 30 minuta. Libre sustav radi na način da se na ruku nalijepi mali okrugli senzor, visok 5 mm i promjera 35mm. Senzor se nanosi na kožu ručno i traje 14 dana. U roku od 14 dana korištenja, senzor omogućava skeniranje pomoću kojeg se

šalju podatci o razini glukoze u krvi u Libre sustav. Kad se skeniraju podatci, može se vidjeti hoće li razina šećera rasti, padati ili će biti stabilna. Libre sustav određuje razinu glukoze iz međustanične tekućine koja djeluje kao spremište hranjivih sastojaka uključujući glukozu. Ovo je ista metoda mjerenja razine šećera koju koriste kontinuirani sustavi praćenja glukoze. Iako se razine šećera dobivene očitavanjem iz međustanične tekućine uglavnom podudaraju s očitavanjem glukoze u krvi, ponekad može doći do razlike, koje najčešće nisu velike. Iz tog razloga, poželjno je tijekom dana provesti nekoliko testova glukoze u krvi kako bi se provjerila točnost [13].

2.5 Pametna narukvica

Pametna narukvica malo je računalo u obliku sata. Sadrži sučelje zaslona osjetljivog na dodir za svakodnevnu upotrebu, dok pridružena aplikacija za pametne mobitele omogućuje upravljanje i telemetriju. Pametne narukvice sadrže razne funkcije kao što su praćenje fizičke aktivnosti, otkucaja srca, kvalitete sna, obavljanja poziva, slušanja audio zapisa, razinu stresa itd. [14].

S obzirom da su fizička aktivnost tijekom dana i strukturirana vježba jako bitni za očuvanje zdravlja kod oboljelih od dijabetesa, pametna narukvica može pomoći u tome na način da, u slučaju dugog sjedenja, obavijeste korisnika na potrebu kretanja. U stvarnom vremenu prate količinu vremena provedenu vježbajući i broj koraka napravljenih tijekom dana. Također se mogu postaviti zadani ciljevi pa osoba može pratiti je li ih uspjela zadovoljiti. Prikazuje se broj potrošenih kalorija, što može pomoći u smanjivanju tjelesne težine. Pametna narukvica, također, u stvarnom vremenu prati i otkucaje srca i razinu stresa, te u slučaju povećanja gore navedenih stavki, upozorava korisnika na odmor. Povezivanje uređaja za kontinuirano praćenje glukoze i pametne narukvice omogućuje korisniku, u svakom trenutku, provjeru glukoze na pametnom satu [15].

3 AGILNI RAZVOJ I RAZVOJ POKRETAN TESTIRANJEM

U ovom poglavlju obrađeni su teorijski principi razvoja programske podrške koji su korišteni prilikom izrade web sustava.

3.1 Agilni razvoj

Uspješan i kvalitetan razvoj programske podrške, zahtijeva dobru metodologiju razvoja programske podrške. Metodologije se mogu podijeliti na dvije osnovne skupine, a to su standardne i agilne. Standardne metodologije mogu se još nazvati i vodopadne metode, jer se odvijaju po fazama, gdje jedna faza slijedi za drugom i nema iteracija. Kod vodopadnog modela prelazi se na sljedeću fazu tek kada su sve faze prije gotove [16].

Prema [17], postoji šest faza vodopadnih metoda, a to su:

1) Zahtjevi
2) Dizajn
3) Razvoj
4) Integracija
5) Testiranje
6) Isporuka

Nedostatci vodopadnog modela su:

- Tim unaprijed mora znati sve zahtjeve vezane uz procjenu vremena, proračuna, članova tima i resursa
- Procjena je složena i zahtijeva visok stupanj kompetentnosti i iskustva
- Kupac neće biti dostupan za odgovaranje na pitanja tijekom razvojnog razdoblja
- Tim mora izbjegavati dodavanje novih zahtjeva
- Tim mora stvoriti dokumentaciju vezanu za upravljanje i kontrolu projekta
- Završno testiranje radi se na kraju projekta
- Potpune i cjelovite povratne informacije kupaca dostupne su tek na kraju projekta

Kod vodopadnih modela, oko 29% projekata je neuspjelo, a uvođenjem agilnih metoda taj se postotak sveo na 9%. Smanjivanje broja neuspjelih projekata rezultat je agilnih metoda koji zastupaju neposredne prilagodbe na temelju čestih provjera napretka i zadovoljstva kupaca [17].

Agilni razvoj je pojam koji se koristi za opisivanje pristupa razvoja programske podrške, koji koristi kontinuirano praćenje, učenje, poboljšavanje, timsku suradnju i ranu dostupnost razvijene programske podrške, te podržava fleksibilne odgovore na promjenu.

Naglašava četiri osnovne vrijednosti:

1) Individualne i timske interakcije prije procesa i alata
2) Programska podrška koja radi prije sveobuhvatne dokumentacije
3) Suradnja kupaca prije pregovora ugovora
4) Odgovor na promjenu prije praćenja plana

Karakteristike agilnog modela su:

- Agilne metode imaju inkrementalan i iterativan pristup razvoju programskog dizajna
- Agilni proces rastavljen je na individualne modele na kojima se radi
- Kupac može rano vidjeti proizvod i donijeti odluke o promjenama na projektu
- Agilni model se smatra nestrukturiranim u usporedbi s modelom vodopada
- Mali projekti se mogu implementirati vrlo brzo, dok je za velike projekte jako teško napraviti estimaciju trajanja razvoja
- Greške se mogu ispraviti usred razvoja projekta
- Razvojni proces je iterativan te se projekt razvija u kratkim iteracijama 2 – 4 tjedna, dok planiranje traje još kraće
- Dokumentacija ima manje značenje naspram razvoja programske podrške
- Svaka iteracija ima vlastitu fazu testiranja i omogućuje implementiranje regresijskog testiranja svaki put kada se nova funkcionalnost doda
- Testira se kada se iteracija završi te se nove funkcionalnosti sustava šalju korisniku na uvid. Te nove funkcionalnosti su odmah dostupne za korištenje, što je jako korisno kada postoji dobar kontakt s kupcima
- Testeri i programeri rade zajedno
- Na kraju svakog sprinta, provodi se korisnički test prihvatljivosti
- Zahtjeva blisku komunikaciju programera, zajedničko analiziranje zahtijeva i planiranje

3.1.1 Vrste agilnih metodologija

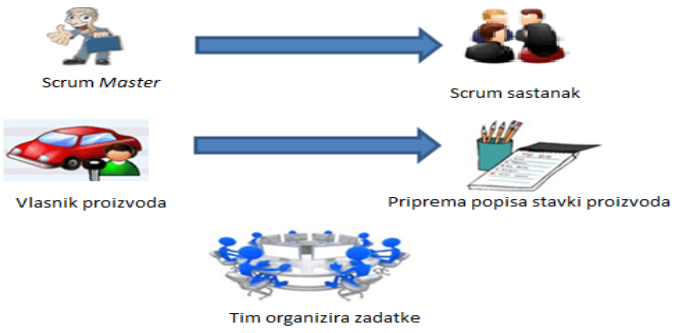
Postoje razne vrste agilnih metoda, ali tri najčešća tipa su scrum, ekstremno programiranje i *lean* razvoj. Ove metode savršeno rade zajedno i dijele mnoge osnovne elemente. *Lean* i Scrum se više baziraju na strukturu, dok se ekstremno programiranje više bazira na prakse razvoja,

testiranje, kodiranje i integraciju. Svaka metoda ima svoje karakteristike, ali se sve drže temeljnih principa agilnih metoda, koji je sastavio tim od sedamnaest autora, i nalaze se u manifestu za agilni razvoj programske podrške. Dvanaest osnovnih principa su:

- Najveći prioritet je zadovoljstvo kupca, što se ostvaruje brzom i kontinuiranom isporukom programskog rješenja
- Promjene zahtjeva sustava mogu se uvoditi u bilo kojem stadiju razvoja
- Učestala isporuka programskog rješenja, od nekoliko tjedana do nekoliko mjeseci, ali s preferencama na kraće vrijeme
- Poslovni korisnici i programeri surađuju na dnevnoj bazi tijekom razvoja projekta
- Projekt se gradi oko motiviranih pojedinaca, treba im dati potrebnu okolinu i podršku
- Najučinkovitija metoda razmjena informacije je komunikacija licem u lice
- Glavnu mjeru napretka predstavlja upotrebljivo programsko rješenje
- Agilne metode potiču i podržavaju održivi razvoj
- Stalni naglasak na tehničku izvrsnost i dobar dizajn uspješuju agilnost
- Naglasak na jednostavnost
- Iz samoorganizirajućih timova proizlazi dobra arhitektura, zahtjevi i dizajn
- Timovi u redovitim intervalima raspravljaju kako biti što više učinkovit i po tome usklađuju svoje ponašanje [18]

3.1.2 Scrum

Scrum je prema [16] jedna od najzastupljenijih agilnih metoda razvoja, koja se bazira na upravljanju zadatcima među timovima koji sudjeluju u razvoju. Zasniva se na snazi razvojnog tima i zagovara rad u malim timovima od 5 – 9 članova. Sadrži tri uloge, a njihove odgovornosti prikazane su slikom 3.1 napravljenom po uzoru na [16]:

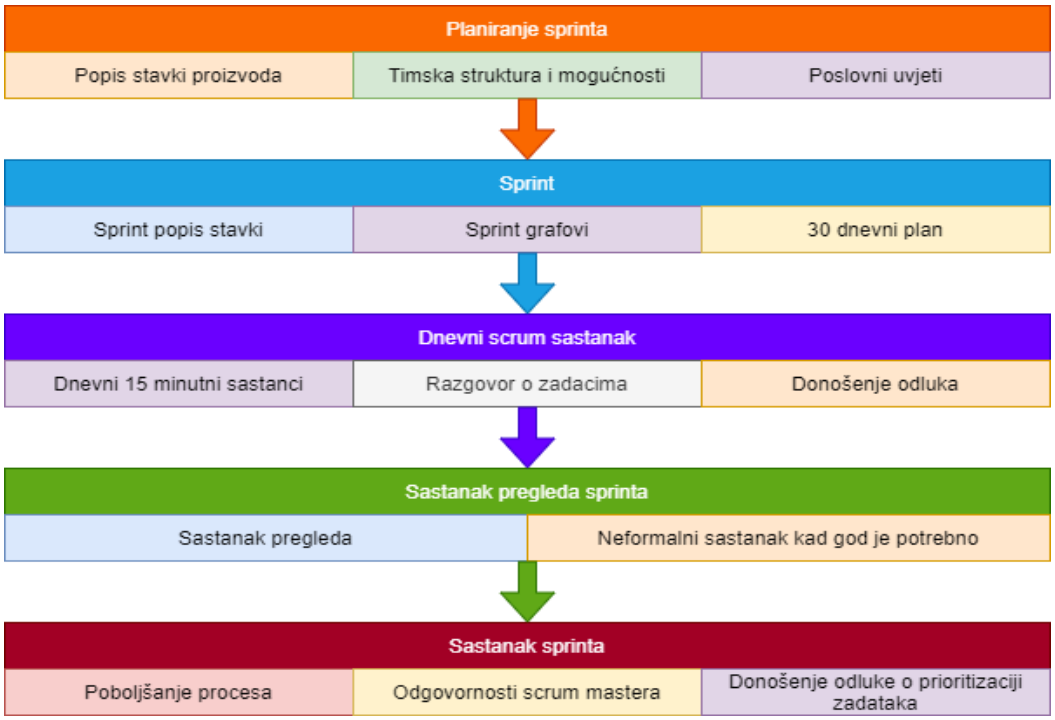


Slika 3.1. Uloge scrum-a [16].

- Scrum master odgovoran je za postavljanje tema, sastanak sprinteva i otklanjanje prepreka napretku razvoja. Osigurava napredak tima tako što potiče kreativnost i povećava produktivnost na sve moguće načine. Također, unaprjeđuje inženjerske prakse i alate. Bilježi informacije o napretku tima koji su vidljivi svim sudionicima.
- Vlasnik proizvoda (eng. *product owner*) - kreira popis stavki proizvoda (eng. *product backlog*), prioritizira popis stavki i odgovoran je za dostavljanje funkcionalnosti svake iteracije. Također, vlasnik proizvoda zastupa kupca, uvijek je upoznat sa željama kupca, a to je uvijek jedna osoba.
- Scrum tim odgovoran je za vlastiti posao i organizira posao među članovima tima kako bi se uspješno završio sprint.

Popis stavki proizvoda mjesto je gdje su poredane funkcionalnosti proizvoda od najvećeg do najmanjeg prioriteta. Vlasnik proizvoda održava i određuje prioritete te ga distribuira do tima koji radi na razvoju. Tim može napraviti zahtjev za novim dodatnim funkcionalnostima, modifikacijom ili brisanjem. Prije svake iteracije, dogovara se koje funkcionalnosti će se napraviti u sljedećoj iteraciji. Na kraju svakog sprinta dolazi do razvijenih funkcionalnosti koje mogu, a i ne moraju ući u produkciju. Postoje tri vrste sastanka kod scrum-a. Dnevni sprint sastanak je sastanak koji se održava svaki dan u isto vrijeme i traje maksimalno 15 minuta. Članovi tima odgovaraju na pitanja: „Što smo radili jučer?“, „Što ćemo raditi danas?“ i „Imamo li kakvih problema?“. Nakon sprinta radi se poseban sastanak pregled sprinta (eng. *sprint review)* u kojem se pojašnjava vlasniku proizvoda što je napravljeno u sprintu. Na sastanku se, na kraju sprinta, odgovara na pitanja „Što je bilo dobro u zadnjem sprintu?“, „Što nije bilo dobro u zadnjem sprintu?“ i „Što učiniti da bi radili bolje?“.

Prakse scrum su detaljno prikazane slikom 3.2. slično kao u [16]:



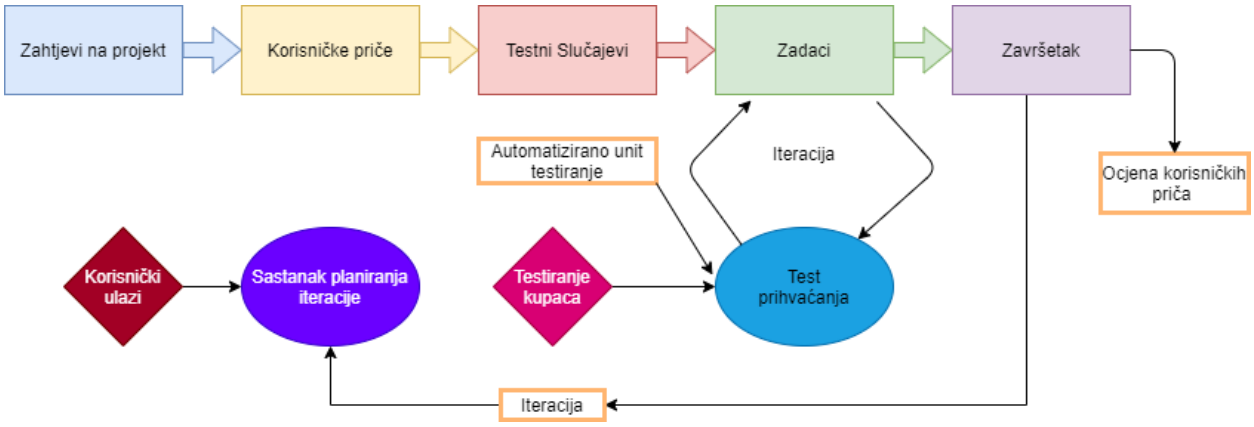
Slika 3.2. Prakse scrum-a [16].

Tijek procesa scrum metodologije:

- Svaka iteracija poznata je kao sprint, a traje između 1 – 4 tjedna
- Popis stavki proizvoda je lista gdje su uneseni svi detalji kako bi se dobio krajnji proizvod
- Tijekom svakog sprinta, funkcionalnosti s najvećim prioritetom uzimaju se i pretvaraju u popis stavki sprinta
- Tim radi na definiranju popisa stavki sprinta
- Tim provjerava što se trenutno treba raditi
- Na kraju svakog sprinta, tim dostavlja funkcionalnost proizvoda

3.1.3 Ekstremno programiranje

Ekstremno programiranje vrlo je korisna tehnika kada postoji konstantna potražnja za zahtjevima od strane kupca ili kada nisu sigurni za neku funkcionalnost sistema. Zagovara česte isporuke proizvoda u kratkim intervalima razvoja, što povećava produktivnost sustava i uvodi točke gdje se novi zahtjevi kupca lagano mogu biti implementirani. Ekstremno programiranje drži kupca u središtu pozornosti. Slika 3.3 preuzeta iz [16] prikazuje tijek razvoja programske podrške s metodologijom ekstremnog programiranja.



Slika 3.3. Tijek ekstremnog programiranja [16].

U ovom načinu metodologije, isporuke su bazirane u kratkim ciklusima nazvanim iteracijama, od kojih svaki postoji 14 dana. Svaka iteracija uključuje faze kao što su programiranje, testiranje jedinica (eng. *unit testing*) i testiranje sustava, gdje se u svakoj fazi napravi minorna ili glavna funkcionalnost aplikacije [16].

Jednostavni koraci u ekstremnom programiranju baziraju se na agilnim principima. Ti koraci su:

- Kodiranje je osnovna aktivnost – programski kod donosi rješenja i može se koristiti za otkrivanje problema. Npr., programer može objasniti problem koristeći programski kod.
- Timovi koriste mnogo testova – programeri ne počinju s programiranjem sve dok nisu osigurali kriterije za razvoj i dizajnirali unit testove. Problem ili defekt nije greška koda, nego greška definiranja pravog testa.
- Komunikacija između programera i kupca je direktna – programer mora razumjeti poslovne zahtjeve kako bi dizajnirao tehničko rješenje.
- Za složene sustave potrebna je neka razina ukupnog dizajna izvan bilo koje određene funkcije. U ekstremnom programiranju (eng. *extreme programming* - *xp*) projektima cjelokupni dizajn uzima se u obzir tijekom redovitog refaktoriranja.
- Metoda ekstremnog programiranja često se može pronaći kombinirano s lean i scrum modelom, jer su procesni elementi slični i međusobno si dobro odgovaraju.

Karakteristike xp-a:

- Planiranje – svi članovi tima trebaju sudjelovati u planiranju, odnosno ne postoji razlika između poslovnog i tehničkog osoblja.

- Sudjelovanje cijelog tima – kupac treba biti dostupan razvojnom timu. Ova dostupnost kupca omogućuje postavljanje pitanja uvezi projekta i dobivanje brzih odgovara, te na kraju izrađeni proizvod koji je više u skladu s potrebama kupca.
- Standardi kodiranja – kako bi se potaklo programere na donošenje odluka, koriste se standardi kodiranja, što pridonosi odražavanju dosljednosti u cijelom proizvodu.
- Kolektivno vlasništvo programskog koda – čitav tim je odgovaran za kvalitetu koda.
- Održivi tempo – prezaposleni ljudi nisu učinkoviti. Previše posla dovodi do pogrešaka, što uzrokuje još više posla. Stoga treba izbjegavati rad dulje od 40 sati tjedno.
- Poboljšanje dizajna – stalno poboljšavanje dizajna refaktoriranjem koda.
- Jednostavan dizajn – što je dizajn jednostavniji, to je niža cijena promjene programskog koda.
- Razvoj pokretan testiranjem.
- Kontinuirana integracija [17].

3.1.4 Lean Software Development

Fokus lean razvoja programskog rješenja bazira se na minimiziranju akcija izvan razvoja proizvoda. Cilj mu je povećanje brzine razvoja programske podrške i smanjivanje troškova. Lean development može se sumirati u sedam točaka:

- Eliminacija „otpada“ – „otpad“ predstavlja stvaranje pogrešne stvari te prebacivanje između zadataka, što kreira previše funkcionalnosti koje nisu skroz dovršene.
- Pojačavanje učenja – učenje pokreće predvidljivost, omogućava poboljšanje i nastoji potaknuti organizacijsku kulturu koja dopušta neuspjeh te učenje iz istog.
- Odgoditi obvezu – omogućiti što kasniju adaptaciju funkcionalnosti, kako bi se imalo dovoljno vremena za odlučivanje, ali istovremeno pratiti vremenska ograničenja.
- Rana dostava – brzina, trošak i kvaliteta su kompatibilni. Što se prije dostavi, to će se ranije dobiti povratna informacija. Raditi na manje stvari odjednom te unaprijed planirati, kako bi se skratilo vrijeme razvoja ciklusa i isporuke.
- Osnaživanje tima – raditi samostalno, svladavati vještine i vjerovati kako posao može motivirati tim. Menadžer ne govori timu kako treba raditi svoj posao, nego ih potiče na samu organizaciju posla. Timu treba osigurati okolinu i potrebne alate za obavljanje posla.
- Kvaliteta izrade – uspostaviti mehanizme za uočavanje i ispravljanje nedostataka prije konačne provjere. Kvaliteta je prisutna otpočetka.

- Optimiziranje cjeline – čitav sustav je jak onoliko koliko je jaka njegova najslabija karika. Treba riješiti probleme, a ne samo simptome, također, treba misliti dugoročno prilikom kreiranja rješenja.

3.1.5 Metrike agilnog razvoja

Metrike koje se mogu pratiti kod efektivnosti agilnog razvoja su:

- Čimbenik povlačenja (eng. *drag factor*)
- Napor u satima koji ne doprinosi cilju sprinta
- Faktor povlačenja može se poboljšati smanjivanjem broja dijeljenih resursa i reduciranjem rada koji ne doprinosi cilju
- Brzina – količina korisničkih priča pretvorenih u funkcionalnost tijekom sprinta
- Broj dodanih jediničnih testova
- Broj detektiranih grešaka u iteraciji
- Broj grešaka koji je završio u produkciji [16]

3.2 Testiranje programske podrške

Postupak testiranja ima dva različita cilja:

1) Demonstrirati programeru i kupcu kako programsko rješenje ispunjava njegove zahtjeve. Za prilagođeno programsko rješenje, znači da bi trebao postojati barem jedan test za svaki zahtjev u dokumentu sa zahtjevima. Dok za generičko programsko rješenje, znači da bi trebalo proizvesti testove za sve značajke sustava i kombinacije tih značajki.
2) Otkriti situaciju u kojima je ponašanje programskog rješenja netočno, nepoželjno ili nije u skladu s njegovim specifikacijama. Testiranje nedostataka bavi se iskorjenjivanjem nepoželjnog ponašanja sustava, poput pada sustava, neželjenih interakcija s drugim sustavima, netočnih izračuna i nepravilnim rukovanjem podatcima.

Prvi cilj dovodi do provjere valjanosti, gdje se očekuje da će sustav pravilno funkcionirati koristeći zadani skup testnih slučajeva koji odražavaju očekivanu upotrebu sustava. Drugi cilj vodi ispitivanju nedostataka, gdje su testni slučajevi osmišljeni tako da otkriju nedostatke. Testiranje je dio šireg postupka verifikacije i validacije. Validacija odgovara na pitanje „Gradimo li pravi proizvod?“, a verifikacije „Gradimo li proizvod na ispravan način?“. Verifikacija i validacija su procesi koji provjeravaju ispunjava li programsko rješenje zahtjeve i dostavlja li očekivane funkcionalnosti, a započinju već prilikom dostupnosti zahtjeva. Cilj

verifikacije je provjeriti je li sustav u skladu s funkcionalnim i nefunkcionalnim zahtjevima. Dok je validacija proces koji osigurava da sustav zadovoljava korisničke zahtjeve.

Testiranje programske podrške važno je zbog ranog pronalaska grešaka u sustava prije nego se sustav pusti u produkciju. Ispravno testirani proizvod osigurava pouzdanost, sigurnost i visoke performanse što rezultira dodatnom uštedom vremena, isplativosti i zadovoljstvom kupaca [19].

Postoji dva tipa testiranja:

- Ručno testiranje – je postupak ručnog testiranja programskog rješenja kako bi se provjerilo što radi ispravno, a što ne. Obično uključuje provjeru svih značajki navedenih u dokumentima sa zahtjevima, ali često uključuje i testere koji provjeravaju programsko rješenje iz perspektive krajnjeg korisnika.
- Automatizirano testiranje – testiranje programske podrške pomoću alata za automatizaciju. Testeri pokreću testne skripte te se rezultati ispitivanja generiraju pomoću alata za automatizaciju [20].

Postoje tri pristupa testiranju:

- White box testiranje – bazira se na poznavanju strukture koda sustava i obično se provodi kod jediničnog testiranja
- Black box testiranje – je testiranje kod kojeg se provjeravaju funkcionalnosti programskog rješenja bez poznavanja unutarnjeg koda.
- Gray box testiranje – kombinacija gore dva navedena testiranja. Testeri imaju pristup dokumentaciji dizajna.

Postoje četiri razine testiranja:

- Jedinično testiranje – predstavlja testove koji provjeravaju određeni dio koda. Ove testove programeri pišu dok rade na pisanju programskog koda kako bi se osigurao očekivani rad funkcije. Jedna funkcija ili dio koda može imati više testova kako bi se obuhvatili svi slučajevi. Osigurava ispravnost rada blokova programskog koda neovisno jedni o drugima.
- Integracijsko testiranje – testiranje je faza ispitivanja programskog rješenja u kojoj se pojedinačni moduli kombiniraju i testiraju kao grupa. Provodi se nakon jediničnog testiranja, za ispitivanje sukladnosti sustava ili komponente s navedenim funkcionalnim zahtjevima.

- Testiranje sustava– testira se potpuno integrirani sustav kako bi se potvrdilo ispunjava li sustav svoje zahtjeve. Cilj je sistemskog testiranja otkriti nedostatke u integriranim jedinicama i u cijelom sustavu.
- Testiranje prihvatljivosti – provodi se kako bi se utvrdilo ispunjava li programsko rješenje zahtjeve ili ne. Postoji više oblika ispitivanja prihvatljivosti kao što su ispitivanje prihvaćanja korisnika, testiranje prihvaćanja poslovanja, alfa testiranje i beta testiranje [21].

Tipovi black-box testiranja su:

- Funkcionalno testiranje - provjerava se radi li svaka funkcija aplikacije onako kako piše u dokumentu zathjeva. Testiranje svih funkcionalnosti, pružanjem odgovarajućeg ulaza, provjerava se stvarni izlaz i uspoređuje s očekivanim rezultatom.
- Ne-funkcionalno testiranje – odnosi se na različite aspekte kao što su performanse, opterećenje, sigurnost, kompatibilnost itd. Glavni je cilj poboljšanje korisničkog iskustva ovisno o tome koliko sustav brzo odgovara na zahtjev [20].

Testiranje programske podrške proizvodi nekoliko artefakata:

- Testni plan – dokument koji detaljno opisuje pristup koji će se poduzeti za predviđanje ispitne aktivnosti. Plan može uključivati aspekte kao što su ciljevi, opseg, procesi i postupci, zahtjevi za osobljem i planovi za slučaj nepredviđenih događaja.
- Matrica sljedivosti – koristi se za usklađivanje zahtjeva ili projektnih dokumenata s ispitnim dokumentima. Koristi se za promjenu testova kada se promijene povezani izvorni dokumenti.
- Testni slučaj –sastoji se od jedinstvenog identifikatora, referenci iz specifikacija dizajna, preduvjeta, događaja i koraka koje treba slijediti, ulaza i izlaza, očekivanog i stvarnog rezultata.
- Testna skripta – je postupak ili programski kod koji ponavlja radnje korisnika.
- Paket testova – sadrži detaljnije upute ili ciljeve za svaku zbirku testnih slučajeva.
- Test podatci – obično se koristi više skupova podataka za testiranje iste funkcionalnosti određene značajke.
- Ispitni pojas – tu pripadaju programi za testiranje, alati, primjeri ulaznih i izlaznih podataka.
- Izvještaj o rezultatima pokretanja testnog slučaja ili skupa testova [22].

3.3 Razvoj pokretan testiranjem

Prema [23] i [24], razvoj pokretan testiranjem uvodi se kao dio agilnih metoda kao što su ekstremno programiranje, ali se može koristiti prilikom svih metoda razvoja. Razvoj pokretan testiranjem (TDD) pristup je razvoja programskog rješenja, gdje se pišu prvo jedinični testovi, a zatim produkcijski kod [23, 24].

Postoje dva levela TDD-a:

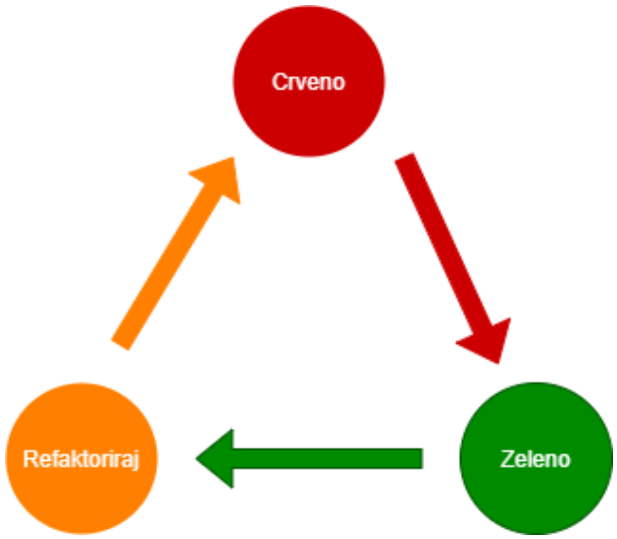
1. *Acceptance* TDD – u ovom tipu prvo se pišu testovi prihvatljivosti. Oni provjeravaju zahtjeve specifikacija i ponašanje sustava. Nakon toga piše se dovoljno koda kako bi se zadovoljio test prihvatljivosti.
2. *Developer* TDD – podrazumijeva pisanje jediničnih testova i produkcijskog koda kako bi se zadovoljio taj test. Jedinično testiranje fokusira se na svaku malu funkcionalnost sustava [25].

Osnovni koraci TDD procesa su:

- Počinje se s identificiranjem potrebne funkcionalnosti, koja obično treba biti mala i implementirajuća u par linija koda.
- Drugi korak je pisanje jediničnih testova za ovu funkcionalnost i omogućavanje pokretanja kao automatiziranog testa.
- Zatim se pokreću svi testovi koji su implementirani, ali kako funkcionalnosti nisu implementirane, testovi će pasti.
- Zatim se implementiraju funkcionalnosti i ponovno se pokreću testovi. Ovdje se može pojaviti refaktoriranje već postojećeg koda kako bi se kod poboljšao. Kod se refaktorira kako bi bio lakše čitljiv i održavan. Refaktoriranje je micanje dupliciranog koda, stvaranje smislenijih imena, razdvajanje funkcija u manje dijelove itd.
- Nakon što su svi testovi uspješno prošli, prelazi se na implementaciju sljedeće funkcionalnosti.

Prema [23], okolina za automatizirano testiranje jako je bitna, jer se programski kod razvija u malim iteracijama i svi se testovi trebaju pokrenuti svaki put nakon dodavanja nove funkcionalnosti ili refaktoriranja programa. Dobar argument za TDD je taj da pomaže programeru u određivanju što neki dio koda radi. Kako bi se napisao test, treba se dobro poznavati što treba napraviti i na taj način je lakše napisati pripadajući kod [23].

Slika 4.1. preuzeta iz [24] prikazuje tri osnovna principa po kojima TTD funkcionira.



Slika 4.1. Graf crveno, zeleno, refaktoriraj [24].

Prema 24, tri osnovna principa su:

- Crvena faza – napisati test i pratiti hoće li pasti
- Zelena faza – napisati dovoljno koda samo kako bi test prošao
- Refaktoriraj kod ako postoje defekti [24]

TDD osigurava testove za svaku značajku i dovodi do dubljeg i ranijeg razumijevanja zahtjeva proizvoda, osigurava učinkovitost testnog koda i održava stalan fokus na kvalitetu programskog rješenja.

Jedinica u jediničnom testiranju definirana je kao klasa ili grupa povezanih funkcija nazvanih modulom. Očuvanje ovih jedinica malim od presudne je važnosti za smanjivanje vremena otklanjanja pogreške i bolje dokumentiranog koda [26].

TDD pristup, osim dobrog razumijevanja problema, ima prednosti kao što su:

- Pokrivenost koda – svaki segment koda treba biti pokriven testom. Na taj način se osigurava da je svaki dio koda provjeren i defekti se rano otkrivaju.
- Regresijsko testiranje – uvijek se mogu pokrenuti regresijski testovi, kako bi se provjerilo postoje li nove greške prilikom promjene u programu.
- Pojednostavljivanje otklanjanja pogrešaka – kada test padne, trebalo bi biti očito gdje je nastao problem.
- Sistemska dokumentacija – programski kod djeluje kao dokumentacija koja prikazuje što radi. Čitanjem testova olakšava se razumijevanje programskog koda.

Ograničenja razvoja pokretanog testiranjem:

- Razvoj pokretan testiranje ne provodi dovoljno ispitivanja u situacijama u kojima su za ispitivanje uspjeha potrebni cjeloviti funkcionalni testovi.
- Podrška menadžmenta je bitna, jer, ako cijela organizacija ne vjeruje da će razvoj pokretan testiranjem poboljšati proizvod, uprava može osjećati da je vrijeme provedeno u pisanju testova nepotrebno.
- Jedinični testovi su napisani od strane programera koji sudjeluju u razvoju stoga testovi mogu dijeliti slijepe točke s kodom. Npr., programer može izostaviti neke ulazne parametre koji bi se trebali provjeriti prilikom testiranja.
- Ispitivanja postaju dio troškova održavanja projekta, loše napisani testovi su skloni neuspjehu te su skupi za održavanje [23].

3.4 Postupak provedbe predloženog aritmetičkog stres testa

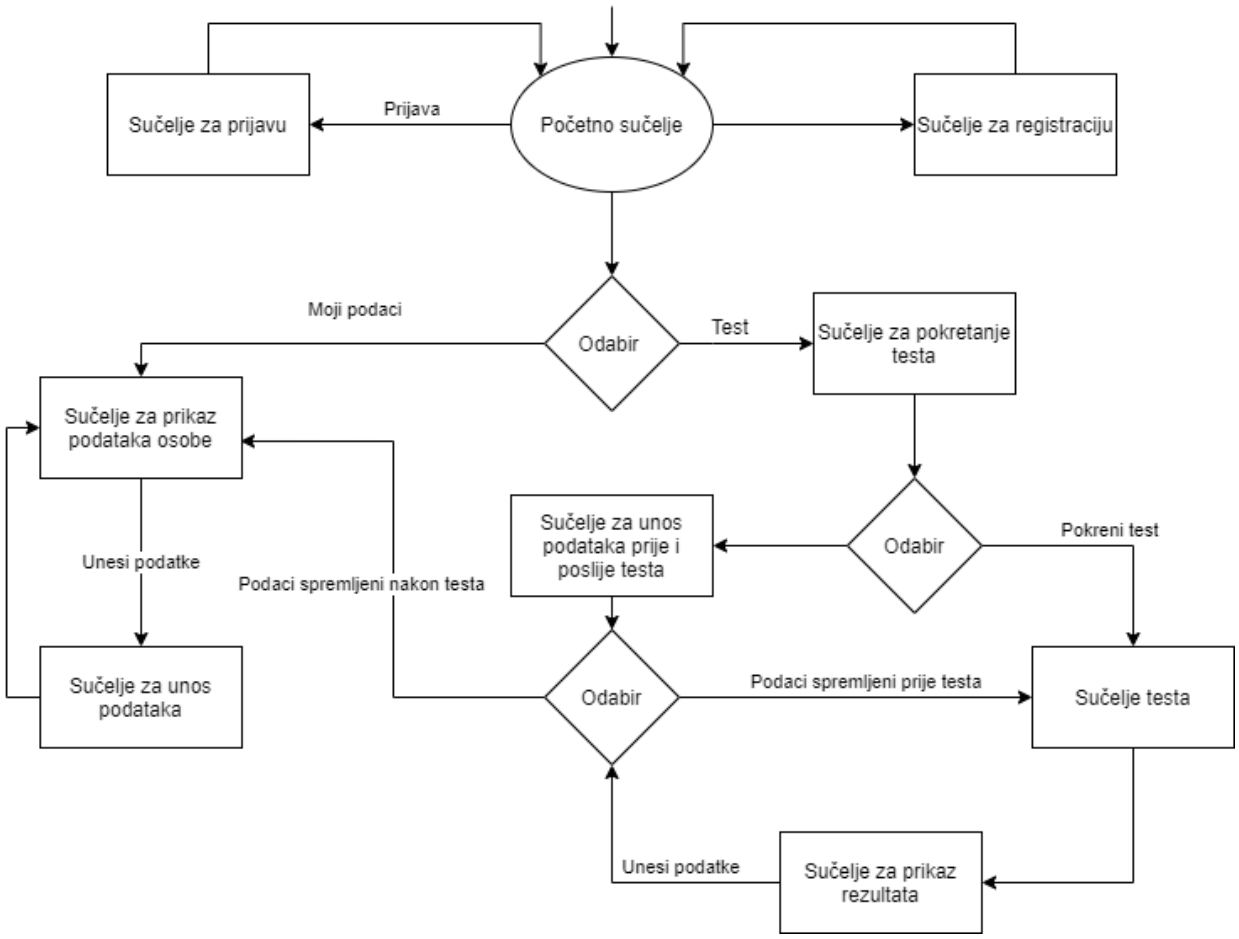
Za potrebe ovog rada napravljena je programska podrška, koja putem web preglednika omogućuje izazivanje stresa aritmetičkom metodom. Aritmetički stres test sastoji se od serijskog oduzimanja manjeg troznamenkastog broja od većeg. U slučaju pogrešnog ili nikakvog odgovora, kako bi se izazvala veća razina stresa, korisnik je obaviješten zvučnim signalom. Postoje tri postavke za promjenu brzine generiranja novih brojeva, odnosno „*Easy*“, „*Medium*“ i „*Hard*“. U slučaju odabira „*Easy*“ postavke, vrijeme do generiranja novog broja veće je od „*Medium*“ i „*Hard*“ postavke. Na taj će način korisnik imati više vremena za razmišljanje o odgovoru, što bi trebalo rezultirati manjim izazvanim stresom i obrnuto. Korisnik ima mogućnost odabira vremena trajanja testa od tri, pet i sedam minuta. Što je korisnik duže izložen aritmetičkom stres testu, to je razina izazvanog stresa veća.

4 MODEL I IDEJNO RJEŠENJE SUSTAVA

U ovom poglavlju bit će definirani funkcionalni i nefunkcionalni zahtjevi na aplikaciju te idejno rješenje web sustava.

4.1 Idejno rješenje web sustava

Glavni je cilj sustava umjetno izazivanje stresa aritmetičkom metodom i praćenje parametara dijabetesa u ovisnosti o izazvanom stresu.



Slika 5.1. Dijagram toka rada aplikacije.

Korisniku se prvo prikazuje početna stranica na kojoj može, bez korisničkog računa, pristupiti pokretanju aritmetičkog testa, ali u tom slučaju neće moći unositi razinu glukoze prije i poslije testa. Ako korisnik napravi račun ili se prijavi s već postojećim korisničkim računom, pruža mu se mogućnost unosa razine glukoze prije testa, kao i nakon testa. Također, korisnik će tada moći pristupiti sučelju za grafički prikaz unesenih podataka te ih može filtrirati po datumu.

4.2 Funkcionalni zahtjevi

Prije izrade same aplikacije potrebno je definirati korisničke scenarije i zahtjeve koje je potrebno ispuniti. Za bazu podataka i autentifikaciju korisnika korišten je Firestore, a za klijentski dio razvijena je aplikacija sa sljedećim korisničkim zahtjevima:

- Korisnik se može registrirati
- Korisnik se može prijaviti u aplikaciju
- Korisnik se može odjaviti iz aplikacije
- Korisnik može pristupiti testu
- Korisnik može unijeti razinu glukoze
- Korisnik može unijeti osobne podatke
- Korisnik može unijeti podatke s pametnog uređaja
- Korisnik ima pregled rezultata testova
- Korisnik ima prikaz glukoze u ovisnosti o izazvanom stresu
- Korisnik ima prikaz glukoze u vremenu
- Korisnik ima prikaz podataka s pametnog uređaja

Za pristup i korištenje web aplikacije potrebno je imati pristup internetu te se aplikaciji može pristupiti putem bilo kojeg web preglednika.

4.3 Nefunkcionalni zahtjevi

Pouzdanost sustava je bitna kako bi se korisnici sa sigurnošću mogli oslanjati na podatke prikazane od strane sustava. Sljedeći nefunkcionalni bitan zahtjev je brzina izvođenja aplikacije koja utječe na korisničko iskustvo. Spremanje rezultata, prikaz i filtriranje trebaju biti implementirani na način da ne zahtijevaju puno vremena za izvođenje, te je bitno da korisniku bude odmah jasno kako sustav radi. To se postiže dobrim korisničkim dizajnom i učinkovitim programskim kodom kao što je korištenje asinkronih funkcija. Također je bitna sigurnost sustava, odnosno, da korisničke lozinke budu adekvatno zaštićene.

5 PROGRAMSKO RJEŠENJE WEB SUSTAVA

Za potrebe diplomskog rada izrađen je web sustav za praćenje parametara oboljelih od dijabetesa u ovisnosti o izazvanom stresu. Za izradu aplikacije korišten je program Visual Studio Code, a u ovom poglavlju opisane su tehnologije i programski jezici korišteni za razvoj te su na kraju dodani dijelovi koda koji su pritom objašnjeni.

5.1 Korištene programske tehnologije, jezici i razvojne okoline

U ovom poglavlju opisane su tehnologije i okviri korišteni za izradu navedenog web sustava.

5.1.1 JavaScript, HTML i CSS

JavaScript programski je jezik koji omogućuje implementaciju složenih značajki web stranice. Svaki put kad stranica prikazuje nešto, osim statičkih podataka, koristi se JavaScript, npr. interaktivne mape, animirane 2D/3D grafike, video itd.

HTML je jezik za označavanje koji se koristi za strukturiranje web sadržaja, npr. definiranje odlomaka, naslova i tablice podataka ili ugrađivanja slika i videozapisa u stranicu. Uobičajeni način korištenja JavaScript-a je dinamičko modificiranje HTML elemenata i CSS kako bi se promijenilo korisničko sučelje pomoću *Document Object Model* API-a.

CSS (eng. *cascading style sheets)* je jezik koji se koristi za oblikovanje stranice. Pomoću CSS-a se opisuju HTML elementi kako će biti prikazani.

5.1.2 React

React je JavaScript okvir otvorenog koda koji se koristi za izgradnju korisničkog sučelja. Održava ga Facebook i zajednica individualnih programera i kompanija. React se koristi za razvoj *single page* aplikacija. Bavi se jedino *state managementom* i prikazom *state*-a u DOM-u. Jedna od glavnih značajki React-a je virtualni DOM. React kreira strukturu podataka u memoriji, računa razliku i zatim promijeni DOM preglednika.

5.1.3 Firebase

Firestore u oblaku računala je fleksibilna, skalabilna baza podataka za mobilne i web aplikacije.

Ključne mogućnosti su:

- Fleksibilnost – podržava fleksibilne hijerarhijske strukture podataka. Podatci se spremaju u dokumente organizirane u zbirke. Dokumenti mogu sadržavati složene ugniježđene objekte uz podzbirke.
- Postavljanje upita – mogu se koristiti upiti za dohvaćanje pojedinačnih, određenih dokumenata ili za preuzimanje svih dokumenata u zbirci, koji odgovaraju parametrima upita. Upiti mogu sadržavati sortiranje i više lančano povezanih filtera.
- Ažuriranje u stvarnom vremenu – koristi se sinkronizacija podataka za ažuriranje podataka na bilo kojem povezanom uređaju.
- Izvanmrežna podrška – Firestore *caches* podatke koji se aktivno koriste, tako da aplikacija može unositi i dohvaćati podatke iako je uređaj izvan mreže. Kada se uređaj vrati na mrežu, Firestore sinkronizira sve lokalne promjene u oblak Firestore.
- Skalabilnost – pruža automatsko kopiranje podataka u više regija, jako jamstvo dosljednosti atomske *batch* operacije i stvarna podrška za transakcije.

Firestore je NoSQL baza podataka u oblaku računala. Podatci se spremaju u dokumente koji sadrže polja u koja se mapiraju vrijednosti. Dokumenti se spremaju u kolekcije, koji predstavljaju kontejner za dokumente koji se koriste kako bi se organizirali podatci i stvarali upiti za dohvaćanjem podataka. Dokumenti podržavaju mnoge tipove podataka, od *string*-a do brojeva, te kompleksnih i ugniježđenih objekata. Mogu se napraviti sub kolekcije u dokumentima i na taj način moguće je izgraditi hijerarhijsku strukturu podataka. Upiti u Firestore su učinkoviti i fleksibilni. Moguće je napraviti upite na razini dokumenta bez potrebe za dohvaćanjem cijele zbirke ili bilo koje ugniježđene podzbirke [27].

5.1.4 React testing library i Jest

Za testiranje programske podrške korišten je React testing library koji predstavlja učinkovito rješenje za testiranje React komponenata. Pruža jednostavne funkcije koje potiču bolju praksu testiranja. Testovi će raditi sa stvarnim DOM čvorovima. Ova biblioteka pruža funkcije koje olakšavaju postavljanje upita prema DOM elementima na isti način kao što bi to korisnik učinio. Biblioteka potiče pristupačnost da aplikacije i omogućuje da testovi budu približeni korištenju komponenata onako kako to želi korisnik, što daje više sigurnosti da će aplikacija raditi kad ga stvarni korisnik koristi [28].

Jest je JavaScript okvir za testiranje s naglaskom na jednostavnost. Jest omogućava pouzdano paralelno pokretanje testova. Da bi stvari bile brže, Jest prvo pokreće prethodno neuspjele testove i ponovno organizira pokretanje na temelju vremena trajanja testnih datoteka. Generira pokrivenost koda dodavanjem zastavice – *coverage*. Nije potrebno dodatno postavljanje i omogućava pokrivenost koda iz cijelog projekta [29].

5.2 Programska podrška na strani korisnika

Klijentska aplikacija razvijena je koristeći razvojni okvir React. Razvoj aplikacije započeo je korištenjem *npx create-react-app* naredbe. Ova naredba postavlja razvojno okruženje tako da se mogu koristiti najnovije JavaScript značajke i optimizira aplikaciju za produkciju. Na ovaj način automatski je postavljena konfiguracija za Jest okvir za testiranje. Za bazu podataka korišten je Firestore koji ujedno služi i kao poslužitelj za autentifikacija korisnika. Korištenje Firestore baze podataka i Firestore funkcija za autentifikaciju iznimno je jednostavno i učinkovito što pokazuje prednosti korištenja usluga u oblaku računala. Programski kod 5.1 prikazuje inicijalizaciju Firestore baze podataka.

```
import firebase from "firebase/app";
import "firebase/auth";
import "firebase/firestore";

const app = firebase.initializeApp({
  apiKey: process.env.REACT_APP_FIREBASE_API_KEY,
  authDomain: process.env.REACT_APP_FIREBASE_AUTH_DOMAIN,
  projectId: process.env.REACT_APP_FIREBASE_PROJECT_ID,
  storageBucket: process.env.REACT_APP_FIREBASE_STORAGE_BUCKET,
  messagingSenderId: process.env.REACT_APP_FIREBASE_MESSAGING_SENDER_ID,
  appId: process.env.REACT_APP_FIREBASE_APP_ID,
  measurementId: process.env.REACT_APP_FIREBASE_MEASUREMENT_ID,
});

export const db = firebase.firestore();
export const auth = app.auth();
export default app;
```

Programski kod 5.1. Inicijalizacija Firestore baze podataka.

S obzirom da je aplikacija razvijana pokretanim testiranjem, prije pisanja koda neke funkcionalnosti, prvo su napisani testovi za tu funkcionalnost. Na primjeru jedne komponente bit će prikazan način razvoja aplikacije i ova metoda je primijenjena na sve ostale komponente. Za primjer je uzeta komponenta *„Login*“ koja prikazuje sučelje za prijavu korisnika. Programski kod 5.2 provjerava hoće li se komponenta prikazati bez grešaka, te sadrži li komponenta sve potrebne elemente. Biblioteka *react-testing-library* omogućava upite za dohvaćanje elemenata kao što su *getByLabelText, getByText, getAllByText*, itd. Pomoću biblioteke Jest napravljen je *mock login* funkcije.

```
it("renders without crashing", () => {
  const login = jest.fn();

  const { getByLabelText, getByText, getAllByText, getByTestId } = render(
    <BrowserRouter>
      <AuthContext.Provider value={{ login }}>
        <Login></Login>
      </AuthContext.Provider>
    </BrowserRouter>
  );

  expect(getByTestId("email-element")).toBeTruthy();
  expect(getByLabelText("Password")).toBeTruthy();
  expect(getByText("Log In")).toBeTruthy();
  expect(getByText("Forgot Password?")).toBeTruthy();
  expect(getAllByText("Need an account?")).toBeTruthy();
  expect(getAllByText("Sing Up")).toBeTruthy();
  expect(getByTestId("email-input")).toBeTruthy();
  expect(getByTestId("password-input")).toBeTruthy();
});
```

Programski kod 5.2. Test za provjeru prikaza komponente.

Programski kod 5.3 provjerava polja za unos e-mail-a i lozinke, odnosno, hoće li polje za e-mail nakon unosa korisnika sadržavati vrijednost koju je korisnik unio, isto tako i polje za lozinku. Nakon dohvaćanja polja, pomoću *getByTestId* funkcije, simulira se unos korisnika pomoću *fireEvent.change()* funkcije. Nakon toga se provjerava sadrže li polja unesene vrijednosti pomoću *expect().toBe()* funkcije.

```
it("email and paswword input updates on change", () => {
  const login = jest.fn();

  const { getByTestId } = render(
    <BrowserRouter>
      <AuthContext.Provider value={{ login }}>
        <Login></Login>
      </AuthContext.Provider>
    </BrowserRouter>
  );

  const emailInput = getByTestId("email-input");
  const passwordInput = getByTestId("password-input");

  fireEvent.change(emailInput, { target: { value: "mariodz95@gmail.com" } });
  fireEvent.change(passwordInput, { target: { value: "test1234" } });

  expect(emailInput.value).toBe("mariodz95@gmail.com");
  expect(passwordInput.value).toBe("test1234");
});
```

Programski kod 5.3. Test za provjeru ponašanja polja za unos.

Programski kod 5.4 provjerava klik na gumb *submit*. Prvo se dohvaća gumb s nazivom *submit*, nakon toga se simulira klik korisnika, odnosno test provjerava hoće li se aktivirati *handleSubmit* funkcija nakon klika na navedeni gumb. Za simulaciju korisničkog klika koristi se *fireEvent.click()* funkcija kojoj se predaje element gumba koji se testira.

```
it("log in should pass without error", async () => {
  const login = jest.fn();

  const { getByText } = render(
    <BrowserRouter>
      <AuthContext.Provider value={{ login }}>
        <Login></Login>
      </AuthContext.Provider>
    </BrowserRouter>
  );

  const submitButton = getByText("Submit");
  fireEvent.click(submitButton);

  await act(async () => {
    expect(login).toHaveBeenCalledTimes(1);
  });
});
```

Programski kod 5.4. Test za provjeru login funkcije nakon klika na gumb submit.

Programski kod 5.5 provjerava hoće li se uhvatiti pogreška nakon što korisnik stisne na gumb *submit* i pozove funkciju *login*. Pomoću Jest okvira simulira se greška.

```
it("log in should catch error", () => {
  const login = () => {
    throw new TypeError("Failed to sign in");
  };

  const { getByText } = render(
    <BrowserRouter>
      <AuthContext.Provider value={{ login }}>
        <Login></Login>
      </AuthContext.Provider>
    </BrowserRouter>
  );

  const submitButton = getByText("Submit");
  fireEvent.click(submitButton);

  expect(() => {
    login();
  }).toThrow("Failed to sign in");
});
```

Programski kod 5.5. Test za provjeru hvatanja greške nakon pozivanja login funkcije.

Kao što je ranije spomenuto, razvoj pokretan testiranjem podrazumijeva prvo pisanje testova, nakon toga se testovi pokreću te će svi biti neuspješni, tj. padaju. Nakon toga se piše programski kod koji će zadovoljiti sve testove. Nakon što je napisan programski kod, pokreću se testovi te bi sada svi testovi trebali biti prolazni. Ovo predstavlja ciklus razvoja programske podrške pokretanim testiranjem. Programski kod 5.6 prikazuje funkcionalnu komponentu „*Login*“. Komponenta prikazuje sučelje za prijavu korisnika, te sadrži *handleSubmit()* funkciju koja poziva *login* funkciju iz *authContext*-a za prijavu korisnika. Funkciji za prijavu predaju se e-mail i lozinka, koji su spremljeni u referencu nakon unosa korisnika. Kako bi sučelje bilo responzivno i bolje izgledalo, korišteni su elementi *bootstrap*-a. Programski kod 5.6 prikazuje komponentu login koja prikazuje sučelje za prijavu korisnika.

```
export default function Login() {
  const emailRef = useRef();
  const passwordRef = useRef();
  const history = useHistory();
  const { login } = useAuth();
  const [error, setError] = useState("");
  const [loading, setLoading] = useState(false);

  async function handleSubmit(e) {
    e.preventDefault();

    try {
      setError("");
      setLoading(true);
      await login(emailRef.current.value, passwordRef.current.value);
      history.push("/");
    } catch {
      setError("Failed to sign in");
    }

    setLoading(false);
  }

  return (
    <>
      <Container
        className="d-flex align-items-center justify-content-center"
        style={{ minHeight: "100vh" }}
      >
        <div className="w-100" style={{ maxWidth: "400px" }}>
          <Card>
            <Card.Body>
              <h2 className="text-center mb-4 title">Log In</h2>
              {error && <Alert variant="danger">{error}</Alert>}
              <Form onSubmit={handleSubmit}>
                <Form.Group id="email" aria-labelledby="email-label">
                  <Form.Label
                    data-testid="email-element"
                    className="email-label"
                    id="email-label"
                    htmlFor="email-input"
                  >
                    Email
                  </Form.Label>
                  <Form.Control
                    data-testid="email-input"
                    id="email-input"
                    type="email"
                    ref={emailRef}
                    required
                  ></Form.Control>
                </Form.Group>
                <Form.Group id="password" aria-labelledby="password-label">
                  <Form.Label id="password-label">Password</Form.Label>
                  <Form.Control
                    data-testid="password-input"
                    type="password"
                    ref={passwordRef}
                    required
                  ></Form.Control>
                </Form.Group>
                <Button disabled={loading} className="w-100" type="submit">
                  Submit
                </Button>
              </Form>
              <div className="w-100 text-center mt-3">
                <Link to="/forgot-password">Forgot Password?</Link>
              </div>
            </Card.Body>
          </Card>
          <div className="w-100 text-center mt-2">
            Need an account? <Link to="/signup">Sing Up</Link>
          </div>
        </div>
      </Container>
    </>
  );
}
```

Programski kod 5.6. Prikaz komponente login.

5.2.1 Autentifikacija

S obzirom da aplikacija zahtjeva provjeru korisnika te korisničke podatke u više komponenti, korišten je *React context* koji omogućava navedeno. Programski kod 5.7 prikazuje funkcije potrebne za registriranje, prijavu, odjavu, promjenu lozinke, promjenu e-mail-a i resetiranje lozinke. Sve navedene funkcije koriste Firestore autentifikaciju.

```
export const AuthContext = React.createContext();

export function useAuth() {
  return useContext(AuthContext);
}

export function AuthProvider({ children }) {
  const [currentUser, setCurrentUser] = useState();
  const [loading, setLoading] = useState(true);

  function signup(email, password) {
    auth.createUserWithEmailAndPassword(email, password);
  }

  function login(email, password) {
    return auth.signInWithEmailAndPassword(email, password);
  }

  function logout() {
    return auth.signOut();
  }

  function resetPassword(email) {
    return auth.sendPasswordResetEmail(email);
  }

  function updateEmail(email) {
    return currentUser.updateEmail(email);
  }

  function updatePassword(password) {
    return currentUser.updatePassword(password);
  }

  useEffect(() => {
    const unsubscriber = auth.onAuthStateChanged((user) => {
      setCurrentUser(user);
      setLoading(false);
    });

    return unsubscriber;
  }, []);

  const value = {
    currentUser,
    signup,
    login,
    logout,
    resetPassword,
    updateEmail,
    updatePassword,
  };

  return (
    <AuthContext.Provider value={value}>
      {!loading && children}
    </AuthContext.Provider>
  );
}

export default AuthContext;
```

Programski kod 5.7. Prikaz komponente login.

5.2.2 Spremanje rezultata aritmetičkog testa

Nakon završenog aritmetičkog testa, sustav automatski sprema broj točnih te broj ukupnih pitanja u bazu podataka i trajanje testa. Podatci se spremaju u zbirku (eng. *collection*) *stressTestResults*, zatim se u tu zbirku dodaje dokument koji sadrži podatke. Za komunikaciju s Firestore bazom podataka korištena je *db.collection().add()* funkcija, a *db* predstavlja inicijaliziranu Firestore bazu podataka. Funkciji se predaje objekt koji sadrži datum spremanja testa, broj pitanja, broj točnih odgovora, te e-mail korisnika koji je završio aritmetički test. Kako se ne bi blokirala aplikacija u slučaju dužeg spremanja podataka, funkcija je asinkrona, što znači da neće blokirati korisnika u daljnjem radu. Programski kod 5.8 prikazuje funkciju za spremanje rezultata aritmetičkog testa. Programski kod 5.9 prikazuje klasu, čiji objekt se predaje funkciji za spremanje rezultata aritmetičkog testa.

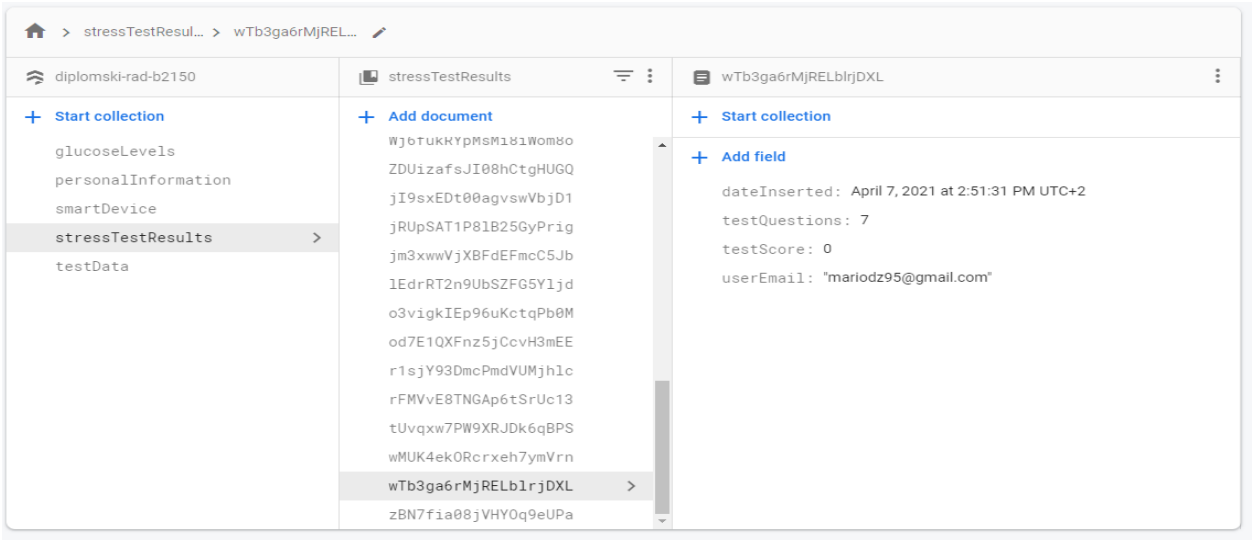
```
async function saveStressTestResult(stressTestResult) {
  db.collection("stressTestResults").add({
    dateInserted: stressTestResult.date,
    testQuestions: stressTestResult.testQuestions,
    testScore: stressTestResult.score,
    userEmail: stressTestResult.userEmail,
  });
}
```

Programski kod 5.8. Funkcija za spremanje rezultata.

```
export class StressTestObject {
  constructor(score, testQuestions, userEmail, date) {
    this.score = score;
    this.testQuestions = testQuestions;
    this.userEmail = userEmail;
    this.date = date;
  }
}
```

Programski kod 5.9. Objekt za spremanje rezultata.

Slika 5.1 prikazuje spremljene podatke aritmetičkog testa na firestore bazi podataka



Slika 5.1 Prikaz spremljenih podataka rezultata aritmetičkog testa.

5.2.3 Spremanje razine glukoze prije i poslije testa

Za spremanje razine glukoze prije i poslije testa korištena je asinkrona funkcija (programski kod 5.10), koja, kao parametre, prima id korisnika i dva objekta od kojih prvi predstavlja podatke prije testa, a drugi podatke poslije testa. U zbirku *testData* dodaje se dokument koji ima naziv jednak kao id korisnika u koji se dodaje zbirka *information* i ta zbirka sadrži dokumente s podatcima. Objekti sadrže razinu glukoze, broj otkucaja srca i razinu stresa. Programski kod 5.11 prikazuje klasu, čiji objekti se predaju funkciji za spremanje razine glukoze

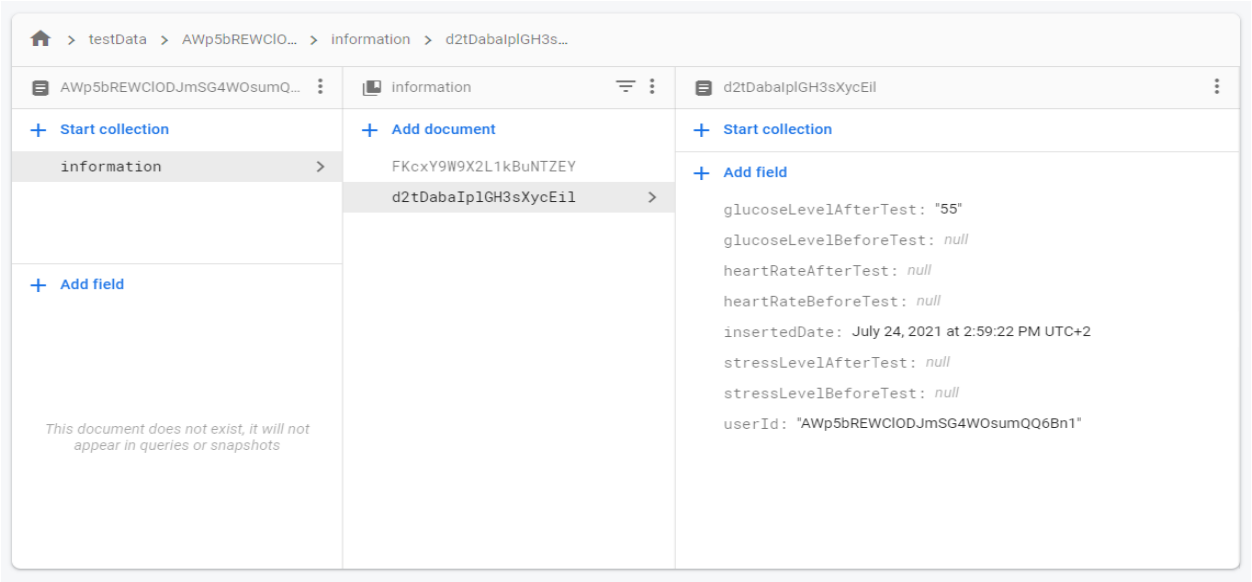
```
async function insertTestDataGlucose(userId, beforeTestData, afterTestData) {
  await db
    .collection("testData")
    .doc(userId)
    .collection("information")
    .add({
      userId: userId,
      glucoseLevelBeforeTest:
        beforeTestData === null ? null : beforeTestData.glucoseLevel,
      heartRateBeforeTest:
        beforeTestData === null ? null : beforeTestData.heartRate,
      stressLevelBeforeTest:
        beforeTestData === null ? null : beforeTestData.stressLevel,
      insertedDate: firebase.firestore.Timestamp.fromDate(new Date()),
      glucoseLevelAfterTest:
        afterTestData === null ? null : afterTestData.glucoseLevel,
      heartRateAfterTest:
        afterTestData === null ? null : afterTestData.heartRate,
      stressLevelAfterTest:
        afterTestData === null ? null : afterTestData.stressLevel,
    })
    .then(() => {
      localStorage.removeItem("testData");
    });
}
```

Programski kod 5.10. Funkcija za spremanje razine glukoze prije i poslije testa.

```
export class TestData {
  constructor(glucoseLevel, heartRate, stressLevel) {
    this.glucoseLevel = glucoseLevel;
    this.heartRate = heartRate;
    this.stressLevel = stressLevel;
  }
}
```

Programski kod 5.11. Objekt za podatke.

Slika 5.2 prikazuje spremljene podatke prije i poslije testa na firestore bazi podataka.



Slika 5.2 Prikaz spremljenih podataka prije i poslije testa.

5.2.4 Pregled rezultata

Svi prijavljeni korisnici mogu, nakon završenog aritmetičkog testa, vidjeti listu korisnika koji su završili test te broj točnih odgovora i ukupan broj pitanja. Programski kod 5.12 prikazuje asinkronu funkciju za dohvaćanje rezultata testa. Ova funkcija vraća sve podatke iz zbirke *stressTestResults* te ih sortira od većeg prema manjem po datumu kreiranja. Za komunikaciju s Firestore bazom podataka korištena je funkcija *db.collection().orderBy().get()*. Podatci su vraćeni u JSON formatu te su na korisničkom sučelju prikazani u obliku tablice.

```
async function getAllTestResults() {
  var testData = [];
  var result = await db
    .collection("stressTestResults")
    .orderBy("dateInserted", "desc")
    .get()
    .then((querySnapshot) => {
      querySnapshot.forEach((doc) => {
        testData.push(doc.data());
      });
      return testData;
    });
  return result;
}
```

Programski kod 5.12. Funkcija za dohvaćanje rezultata testa.

5.2.5 Funkcije za prikaz podataka

Za dohvaćanje podataka o razini glukoze korištena je asinkrona funkcija koja prima id korisnika i trenutni datum. Iz datuma se dohvaćaju prvi i zadnji dan u mjesecu. Zatim se iz zbirke *glucoseLevels* dohvaća dokument koji ima naziv isti kao id korisnika, a iz tog dokumenta pristupa se zbirci *levels* iz koje se dohvaćaju dokumenti za traženi mjesec. Podatci se sortiraju po

datumu spremanja od manjeg prema većem. Programski kod 5.13 prikazuje funkciju za dohvaćanje razine glukoze iz Firestore baze podataka.

```
async function getGlucoseData(userId, date) {
  var firstDay = new Date(date.getFullYear(), date.getMonth(), 1);
  let lastDay = new Date(
    date.getFullYear(),
    date.getMonth() + 1,
    0,
    23,
    59,
    59
  );

  var glucoseData = [];
  var result = await db
    .collection("glucoseLevels")
    .doc(userId)
    .collection("levels")
    .where("insertedDate", ">", firstDay)
    .where("insertedDate", "<", lastDay)
    .orderBy("insertedDate", "asc")
    .get()
    .then((querySnapshot) => {
      querySnapshot.forEach((doc) => {
        glucoseData.push(doc.data());
      });
      return glucoseData;
    });
  return result;
}
```

Programski kod 5.13. Funkcija za dohvaćanje razine glukoze.

Za dohvaćanje podataka unesenih s pametne narukvice korištena je asinkrona funkcija koja kao parametre prima id korisnika i datum. Uzimaju se prvi i zadnji dan u mjesecu od predanog datuma. Iz zbirke *smartDevice* dohvaća se dokument koji ima id isti kao id korisnika te dokument sadrži zbirku *data* iz koje se dohvaćaju podatci za navedeni mjesec. Podatci su sortirani po datumu od manjeg prema većem. Programski kod 5.14 prikazuje funkciju za dohvaćanje podataka unesenih s pametne narukvice.

```
async function getSmartDeviceData(userId, date) {
  var firstDay = new Date(date.getFullYear(), date.getMonth(), 1);
  let lastDay = new Date(
    date.getFullYear(),
    date.getMonth() + 1,
    0,
    23,
    59,
    59
  );
  var smartDeviceData = [];
  var result = await db
    .collection("smartDevice")
    .doc(userId)
    .collection("data")
    .where("dateTime", ">", firstDay)
    .where("dateTime", "<", lastDay)
    .orderBy("dateTime", "asc")
    .get()
    .then((querySnapshot) => {
      querySnapshot.forEach((doc) => {
        smartDeviceData.push(doc.data());
      });
      return smartDeviceData;
    });
  return result;
}
```

Programski kod 5.14. Funkcija za dohvaćanje podataka unesenih s pametne narukvice.

Za dohvaćanje podataka razine glukoze prije i poslije testa korištena je asinkrona funkcija koja za parametre prima id korisnika i datum. Uzimaju se prvi i zadnji dan u mjesecu od predanog datuma. Iz zbirke „*testData*“ dohvaća se dokument koji ima naziv isti kao i predani id te se nakon toga dohvaćaju svi dokumenti koji se nalaze u zbirci *information* i zadovoljavaju uvjet filtriranja po datumu spremanja. Podatci se sortiraju po datumu spremanja od manjeg prema većem. Programski kod 5.15 prikazuje funkciju za dohvaćanje podataka razine glukoze prije i poslije testa.

```
async function getStressTestData(userId, date) {
  var firstDay = new Date(date.getFullYear(), date.getMonth(), 1);
  let lastDay = new Date(
    date.getFullYear(),
    date.getMonth() + 1,
    0,
    23,
    59,
    59
  );

  var stressTestData = [];
  var result = await db
    .collection("testData")
    .doc(userId)
    .collection("information")
    .where("insertedDate", ">", firstDay)
    .where("insertedDate", "<", lastDay)
    .orderBy("insertedDate", "asc")
    .get()
    .then((querySnapshot) => {
      querySnapshot.forEach((doc) => {
        stressTestData.push(doc.data());
      });
      return stressTestData;
    });
  return result;
}
```

Programski kod 5.15. Funkcija za dohvaćanje podataka razine glukoze prije i poslije testa.

5.3 Korisnički podatci

Za dohvaćanje korisničkih podataka koristi se asinkrona funkcija koja za parametar prima id korisnika. Iz zbirke *personalInformation* dohvaća se dokument koji ima naziv isti kao id korisnika. Programski kod 5.16 prikazuje funkciju za dohvaćanje korisničkih podataka.

```
async function getPersonalData(userId) {
  var result = await db
    .collection("personalInformation")
    .doc(userId)
    .get()
    .then((doc) => {
      if (doc.exists) {
        return doc.data();
      }
    });

  return result;
}
```

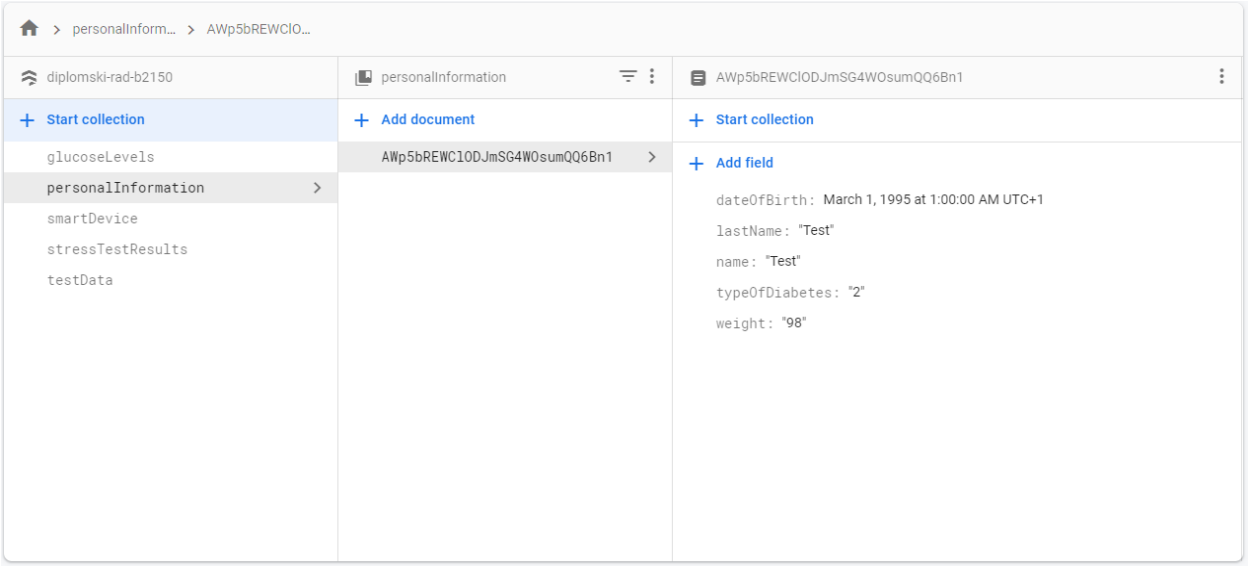
Programski kod 5.16. Funkcija za dohvaćanje korisničkih podataka.

Programski kod 5.17 prikazuje funkciju za unos i promjenu podataka korisnika. Asinkronoj funkciji potrebno je predati id korisnika i objekt. Objekt sadrži ime, prezime, tip dijabetesa, težinu i datum rođenja.

```
async function updatePersonalData(userId, userData) {
  db.collection("personalInformation").doc(userId).set({
    name: userData.name,
    lastName: userData.lastName,
    typeOfDiabetes: userData.typeOfDiabetes,
    weight: userData.weight,
    dateOfBirth: userData.dateOfBirth,
  });
}
```

Programski kod 5.17. Funkcija za unos i promjenu korisničkih podataka.

Slika 5.3 prikazuje spremljene podatke korisnika na firestore bazi podataka.



Slika 5.3. Prikaz spremljenih podataka korisnika.

5.3.1 Spremanje razine glukoze i podataka s pametne narukvice

Za spremanje razine glukoze korištenja je asinkrona funkcija koja kao parametre prima id korisnika i *glucoseLevels* objekt. U zbirku *glucoseLevels* dodaje se dokument koji ima naziv isti kao i id korisnika i onda se dodaje zbirka *levels* koja sadrži dokumente s podatcima. Programski kod 5.18 prikazuje funkciju za spremanje razine glukoze, a programski kod 5.19 prikazuje klasu, čiji se objekt predaje funkciji za spremanje razine glukoze.

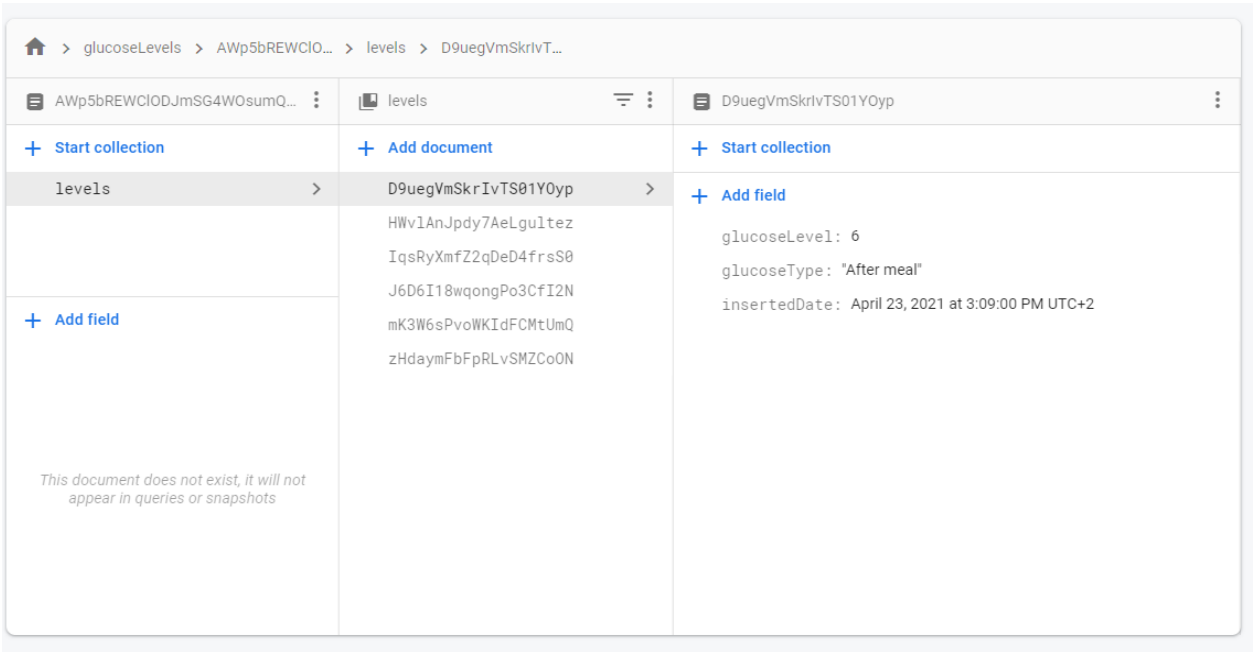
```
async function insertGlucoseData(userId, glucoseLevel) {
  db.collection("glucoseLevels")
    .doc(userId)
    .collection("levels")
    .add({
      glucoseLevel: Number(glucoseLevel.glucoseLevel),
      glucoseType: glucoseLevel.glucoseType,
      insertedDate: glucoseLevel.insertedDate,
    });
}
```

Programski kod 5.18. Funkcija za spremanje razine glukoze.

```
export class GlucoseLevel {
  constructor(glucoseLevel, glucoseType, insertedDate) {
    this.glucoseLevel = glucoseLevel;
    this.glucoseType = glucoseType;
    this.insertedDate = insertedDate;
  }
}
```

Programski kod 5.19. Objekt korišten za razinu glukoze.

Slika 5.4 prikazuje spremljene podatke na Firestore bazi podataka pomoću *insertGlucoseData()* funkcije.



Slika 5.4 Prikaz spremljenih podataka razine glukoze.

Za spremanje unesenih podataka s narukvice korištena je asinkrona funkcija koja kao parametre prima id korisnika i objekt *smartDevice*. U zbirku *smartDevice* dodaje se dokument koji za naziv ima id korisnika i u taj dokument se dodaje zbirka *data*. Programski kod 5.20 prikazuje funkciju za spremanje podataka s pametne narukvice, a programski kod 5.21 prikazuje klasu, čiji se objekt predaje funkciji za spremanje podataka s pametne narukvice.

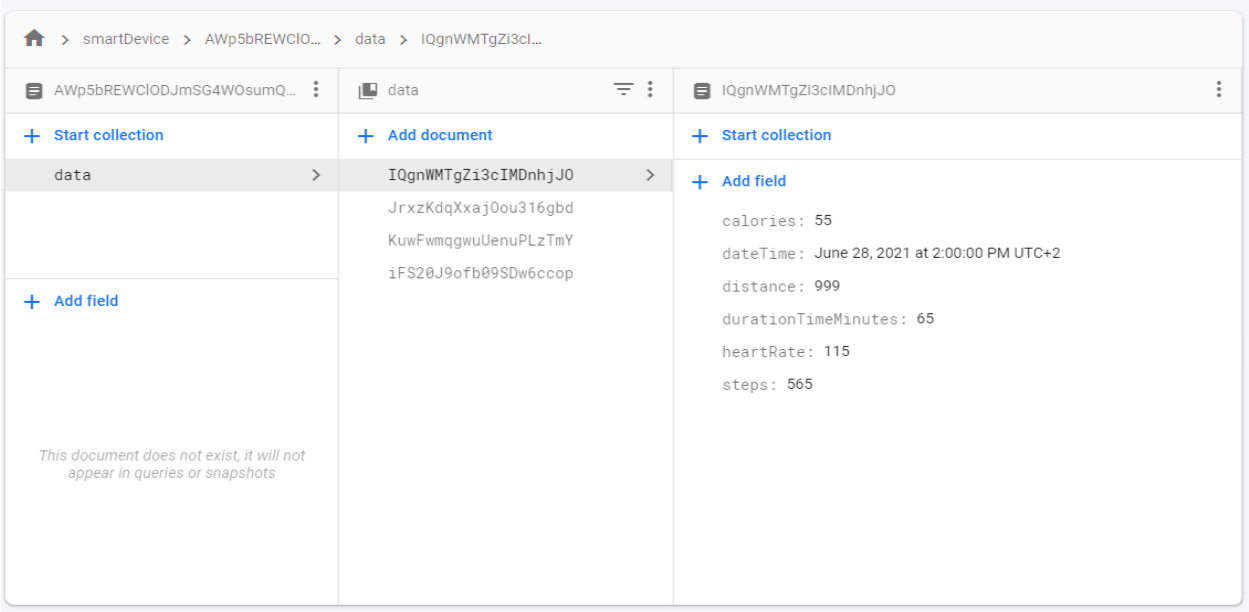
```
async function insertSmartDeviceData(userId, smartDevice) {
  db.collection("smartDevice")
    .doc(userId)
    .collection("data")
    .add({
      heartRate: Number(smartDevice.heartRate),
      distance: Number(smartDevice.distance),
      steps: Number(smartDevice.steps),
      calories: Number(smartDevice.calories),
      durationTimeMinutes: Number(smartDevice.durationTimeMinutes),
      dateTime: smartDevice.dateTime,
    });
}
```

Programski kod 5.20. Funkcija za spremanje podataka sa pametne narukvice.

```
export class SmartDevice {
  constructor(
    calories,
    distance,
    durationTimeMinutes,
    heartRate,
    steps,
    dateTime
  ) {
    this.calories = calories;
    this.distance = distance;
    this.durationTimeMinutes = durationTimeMinutes;
    this.heartRate = heartRate;
    this.steps = steps;
    this.dateTime = dateTime;
  }
}
```

Programski kod 5.21. Objekt tipa SmartDevice.

Slika 5.5 prikazuje spremljene podatke pomoću funkcije *insertSmartDeviceData()* na Firestore bazu podataka.



Slika 5.5. Prikaz spremljenih podataka s pametne narukvice u Firestore bazu podataka.

5.4 Utjecaj agilnog razvoja i razvoja pokretanog testiranjem na programsku podršku

Tijekom razvoja web aplikacije, za svaku funkcionalnost razvijali su se i jedinični testovi. Za svaku funkcionalnost aplikacije prvo su pisani jedinični testovi, zatim bi se svi testovi pokrenuli, ali kako nema programskog koda, testovi bi pali. Nakon toga bi se pisao programski kod kako bi testovi uspješno prošli. Navedeni postupci dio su razvoja programske podrške pokretane testiranjem kao što je ranije spomenuto. Korištena je automatizirana okolina za skupno pokretanje testova pomoću naredbe *npm test -- --coverage --watchAll*. Zbog tog načina razvoja, programska podrška ima visoku razinu pokrivenosti programskog koda testovima. Ovakav način razvoja je kompleksan i značajno povećava vrijeme razvoja programske podrške, međutim, nagrađuje trud na način da testovi pomažu u razumijevanju programskog koda, pronalaženju grešaka, sprječavanju dupliciranja programskog koda te omogućuju lakše refaktoriranje koda.

File		Statements		Branches		Functions		Lines	
features/auth/forgotPassword		100%	15/15	100%	4/4	100%	2/2	100%	15/15
features/auth/login		100%	15/15	100%	2/2	100%	2/2	100%	15/15
features/auth/signup		100%	18/18	100%	4/4	100%	2/2	100%	18/18
features/dataForm		100%	62/62	92.86%	26/28	100%	9/9	100%	62/62
features/dataForm/models		100%	17/17	100%	0/0	100%	4/4	100%	17/17
features/glucoseData		100%	3/3	100%	0/0	100%	2/2	100%	3/3
features/home		100%	1/1	100%	0/0	100%	1/1	100%	1/1
features/layout		100%	12/12	100%	8/8	100%	4/4	100%	12/12
features/mydata		94.74%	54/57	83.33%	10/12	95%	19/20	94.55%	52/55
features/profile		100%	1/1	100%	10/10	100%	1/1	100%	1/1
features/result		100%	16/16	100%	2/2	100%	6/6	100%	14/14
features/smartDeviceGraph		100%	4/4	100%	0/0	100%	2/2	100%	4/4
features/statistic		100%	65/65	100%	20/20	100%	14/14	100%	64/64
features/stressTestData		100%	5/5	100%	0/0	100%	3/3	100%	5/5
features/test		70.33%	64/91	56.25%	18/32	74.07%	20/27	68.97%	60/87
firebase		100%	3/3	100%	0/0	100%	0/0	100%	3/3

Slika 5.6. Prikaz pokrivenosti programskog koda jediničnim testovima.

Kako je već ranije opisano, agilni razvoj predstavlja razvoj pojedinih funkcionalnosti programske podrške u iteracijama te ranu dostupnost proizvoda. Aplikacija je podijeljena na pojedine funkcionalnosti, koje su zasebno razvijane, i nakon razvoja pojedine funkcionalnosti odmah je bila dostupna korisniku za korištenje. U slučaju nezadovoljstva trenutnim stanjem aplikacije, aplikacija se bez problema mogla izmijeniti. Kako je aplikacija razvijana pokretanim testiranjem, svaka funkcionalnost sadrži potrebne jedinične testove, te nakon razvoja pojedine funkcionalnosti, provodilo bi se ručno testiranje. Ako bi se tijekom razvoja programske podrške pronašla pogreška, bilo ju je jednostavno ispraviti zbog toga što je bilo potrebno pronaći pogrešku samo u programskom kodu te funkcionalnosti. Agilni razvoj podrške doveo je do toga da smo, već u ranom razvoju programske podrške, imali dostupne funkcionalnosti aplikacije, a ako je bilo potrebno, izmjene su se mogle učiniti odmah, što je jednostavnije nego da su se radile na gotovom proizvodu.

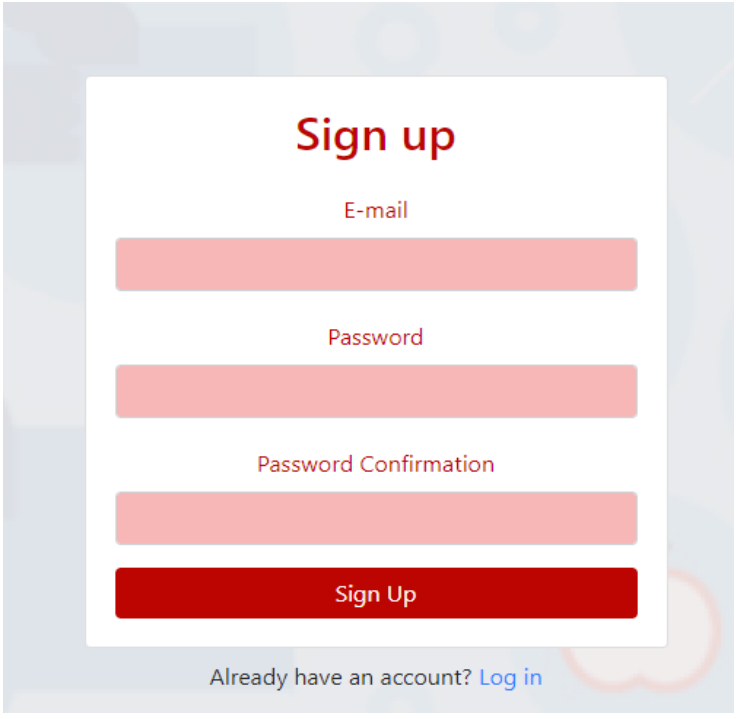
6 OPIS I PRIKAZ NAČINA RADA SUSTAVA

U ovom poglavlju detaljno je opisan i prikazan način rada sustava.

6.1 Opis načina rada web sustava

6.1.1 Registriranje korisnika

Kako bi korisnik mogao unositi razinu glukoze prije i poslije testa, tijekom vremena, kao i podatke s pametne narukvice, potrebno je napraviti korisnički račun. Sučelju za registriranje novog korisnika može se pristupiti klikom na *Sign up* na navigacijskoj traci. Na sučelju je prikazana forma za registriranje novog korisnika koja se sastoji od e-mail-a, lozinke te potvrde lozinke. U slučaju da korisnik nije popunio neko od polja, registriranje se neće odraditi, te će korisniku biti prikazana poruka da treba ispuniti prazno polje. Korisnik može pristupiti sučelju za prijavu klikom na *Login* poveznicu. Slika 6.1 prikazuje izgled forme za registriranje novog korisnika.

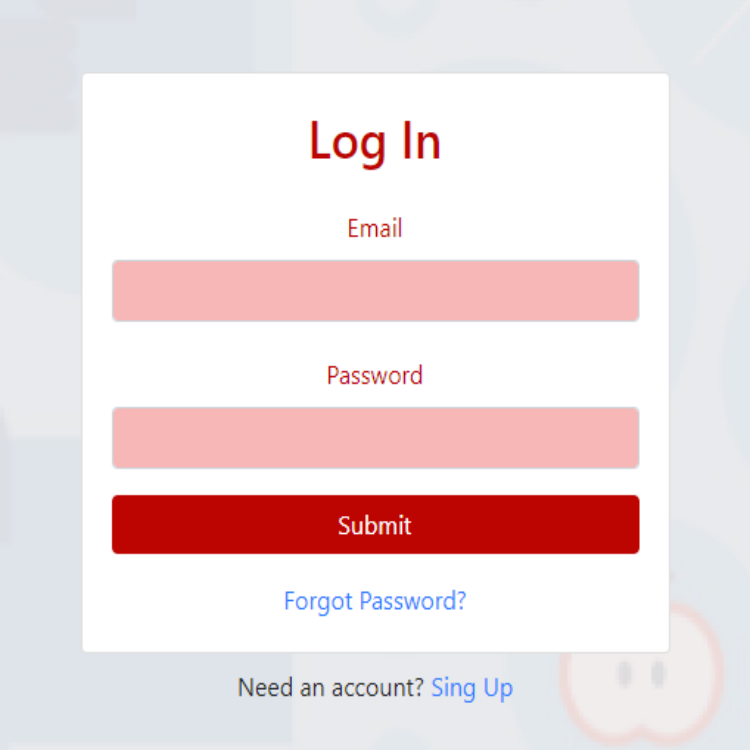


Slika 6.1. Izgled forme za registriranje novog korisnika.

6.1.2 Prijava korisnika i promjena lozinke

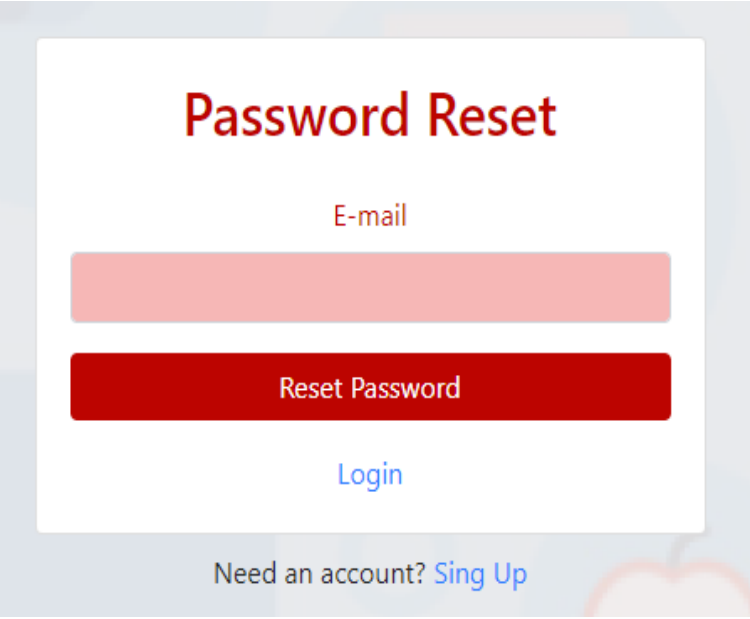
Ako korisnik već ima registriran korisnički račun, može se prijaviti u sustav. Sučelju može pristupiti klikom na *Login* na navigacijskoj traci ili iz sučelja za registraciju. Sučelje za prijavu sastoji se od forme s poljima za unos e-mail-a i lozinke. Ako se e-mail i lozinka podudaraju s

postojećim u bazi podataka, korisnik je prijavljen u sustav te će biti preusmjeren na početnu stranicu. Korisnik iz ovog sučelja može pristupiti sučelju za promjenu lozinke. Sučelje za promjenu lozinke prikazuje formu za unos e-mail-a, nakon čega će korisnik dobiti e-mail s uputama za izmjenu lozinke. Slika 6.2 prikazuje izgled forme za prijavu korisnika.



Slika 6.2. Izgled forme za prijavu korisnika.

Slika 6.3 prikazuje izgled forme za promjenu lozinke.



Slika 6.3. Izgled forme za promjenu lozinke.

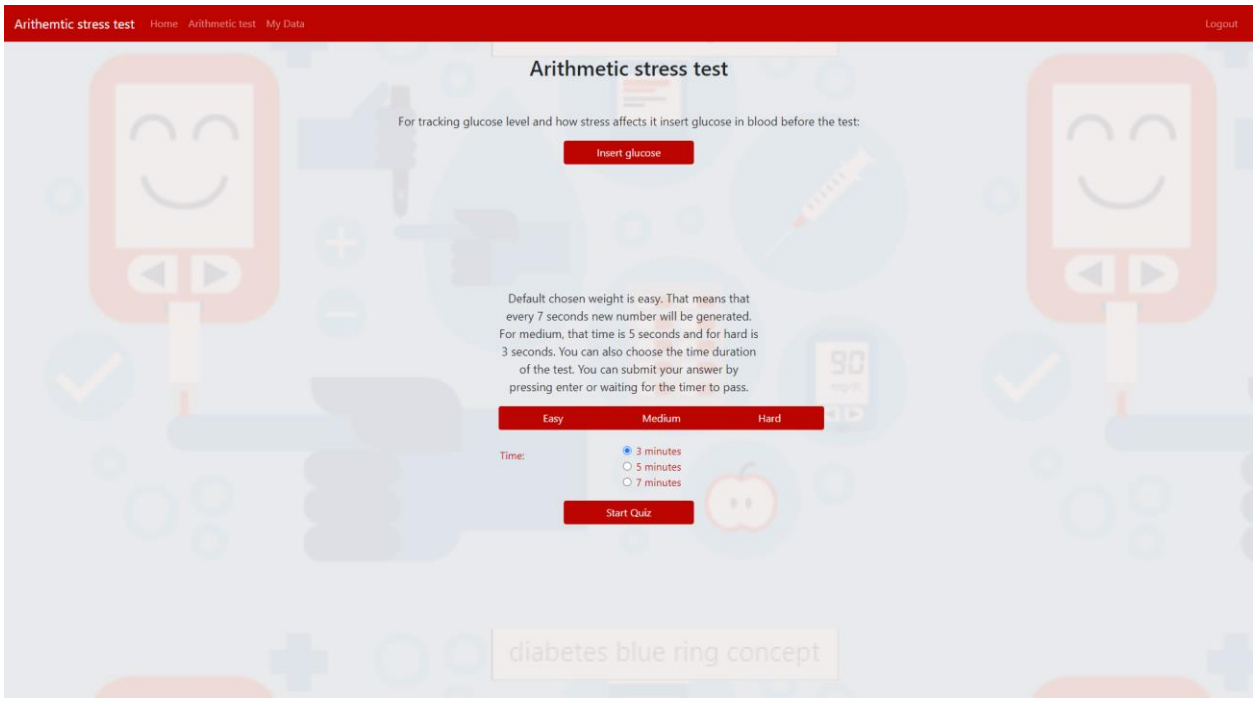
6.1.3 Početna stranica

Na početnoj stranici (slika 6.4) prikazani su simptomi dijabetesa. Korisnik može klikom na gumb *Start arithmetic test* otvoriti sučelje za pokretanje aritmetičkog testa. Početnoj stranici može se pristupiti klikom na gumb *Home* iz navigacijske trake. Sučelje je dostupno i prijavljenim i neprijavljenim korisnicima.

Slika 6.4. Izgled početne stranice.

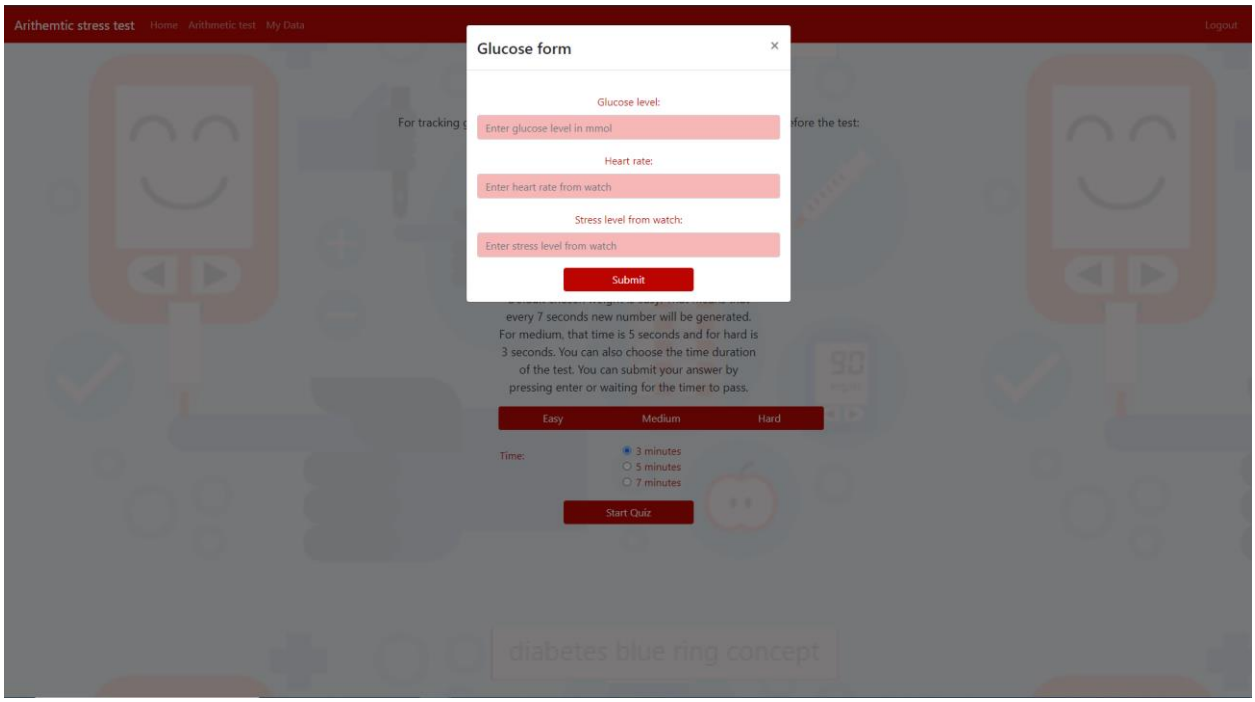
6.1.4 Sučelje za prikaz aritmetičkog testa

Sučelje za prikaz aritmetičkog testa nudi korisniku mogućnost unosa razine glukoze prije testa (slika 6.5). Klikom na gumb *Insert glucose* otvara se modalni prozor koji sadrži formu koja se sastoji od polja za unose razine glukoze u mmol, a ako korisnik ima pametni sat, može unijeti otkucaje srca i razinu stresa sa sata. Klikom na gumb *Submit*, modal se zatvara te se sprema u lokalni spremnik aplikacije.



Slika 6.5. Izgled sučelja za pokretanje testa.

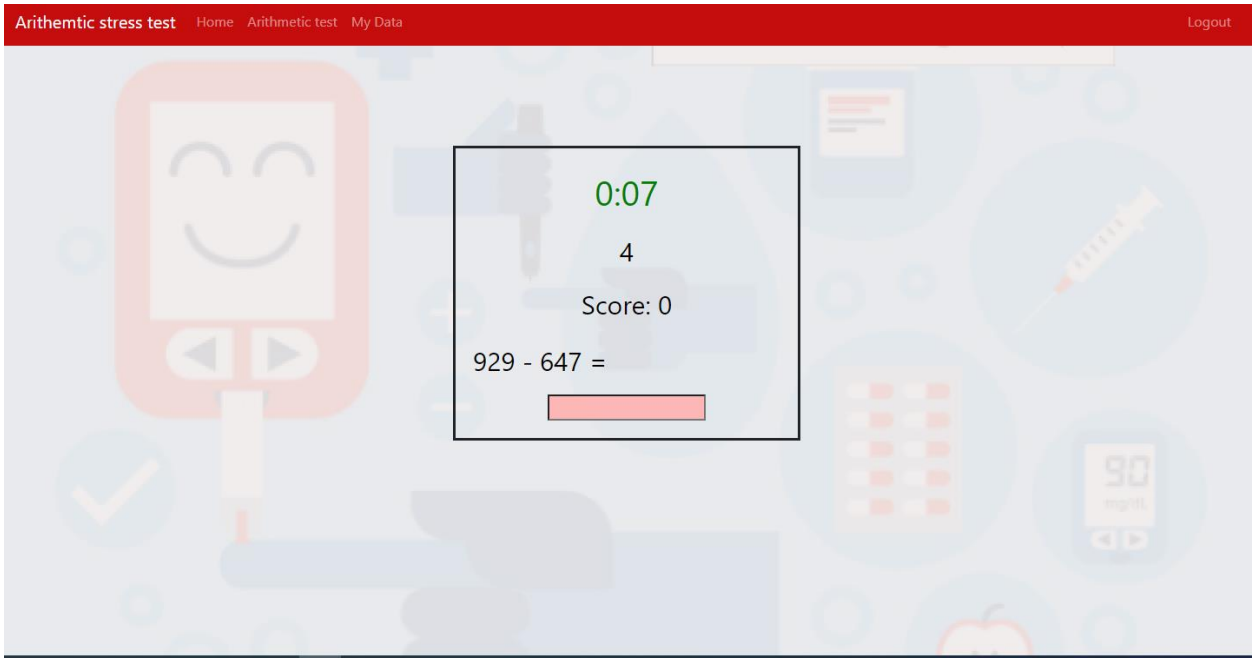
Slika 6.6 prikazuje izgled forme za unos podataka prije testa.



Slika 6.6. Izgled forme za unos podataka prije testa.

Zatim korisnik može odabrati težinu aritmetičkog testa, a opcije su *Easy*, *Medium* i *Hard*. To se odnosi na brzinu generiranja novih brojeva. *Easy* postavka znači da će se novi broj generirati svakih 7 sekundi, *medium* svakih 5 i *hard* svake 3 sekunde. Može se odabrati i vrijeme trajanja

testa koje može biti 3, 5 ili 7 minuta. Klikom na gumb *Start Quiz* započinje test. Slika 6.7 prikazuje sučelje za rješavanje aritmetičkog testa.



Slika 6.7. Izgled sučelja za prikaz testa.

Brojač zelene boje predstavlja odbrojavanje ukupnog vremena trajanja testa. Brojač ispod toga predstavlja vrijeme do generiranja novih brojeva. Također, korisniku su prikazana dva broja te operacija oduzimanja. U ponuđeno polje korisnik treba unijeti odgovor, zatim može stisnuti enter, te će se odgovor zaprimiti i generirat će se novi brojevi. Korisnik može i sačekati da prođe vrijeme do generiranja novog broja, te će bilo kakav unos biti zaprimljen kao odgovor. U slučaju neispravnog odgovora korisnik će biti obavješten zvučnim signalom.

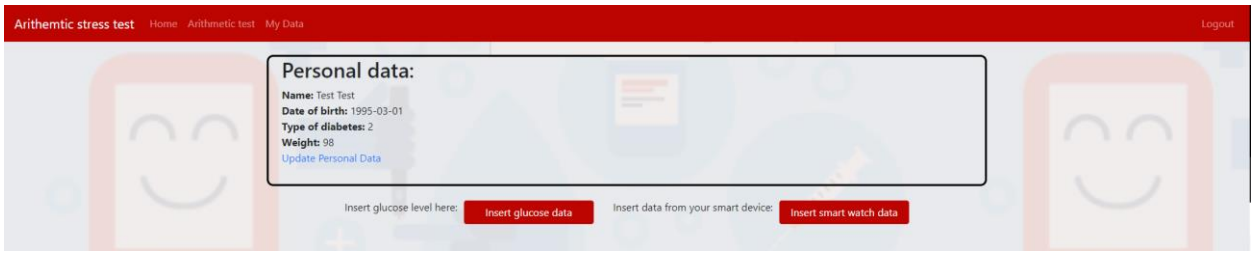
6.1.5 Sučelje za prikaz rezultata testa

Sučelje za prikaz rezultata omogućuje prikaz broja točnih odgovora te ukupan broj pitanja na koja je dan odgovor. Na slici 6.8 prikazana je lista rezultata poredanih po datumu od svih korisnika. Korisniku se nudi mogućnost unosa razine glukoze nakon testa, a ako korisnik unese vrijednost, bit će spremljena vrijednost prije i poslije testa u bazu podataka.

Slika 6.8. Sučelje za prikaz rezultata testa.

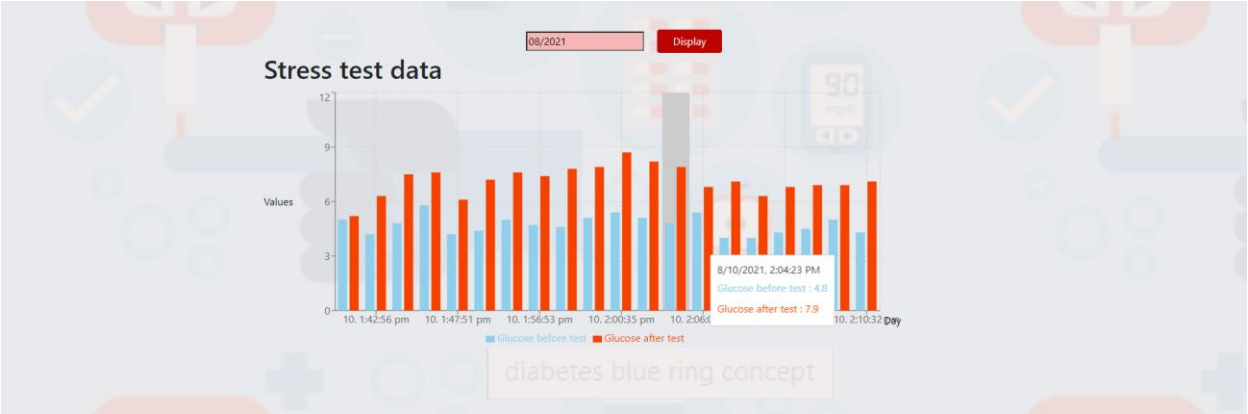
6.1.6 Sučelje za prikaz korisničkih podataka

Na ovom sučelju, na samom početku, prikazani su osobni podatci korisnika (slika 6.9). Korisnik klikom na poveznicu *Update Personal Data* otvara sučelje za unos i izmjenu osobnih podataka. Nakon toga, korisniku su prikazana dva gumba koji vode na posebna sučelja za unos podataka.



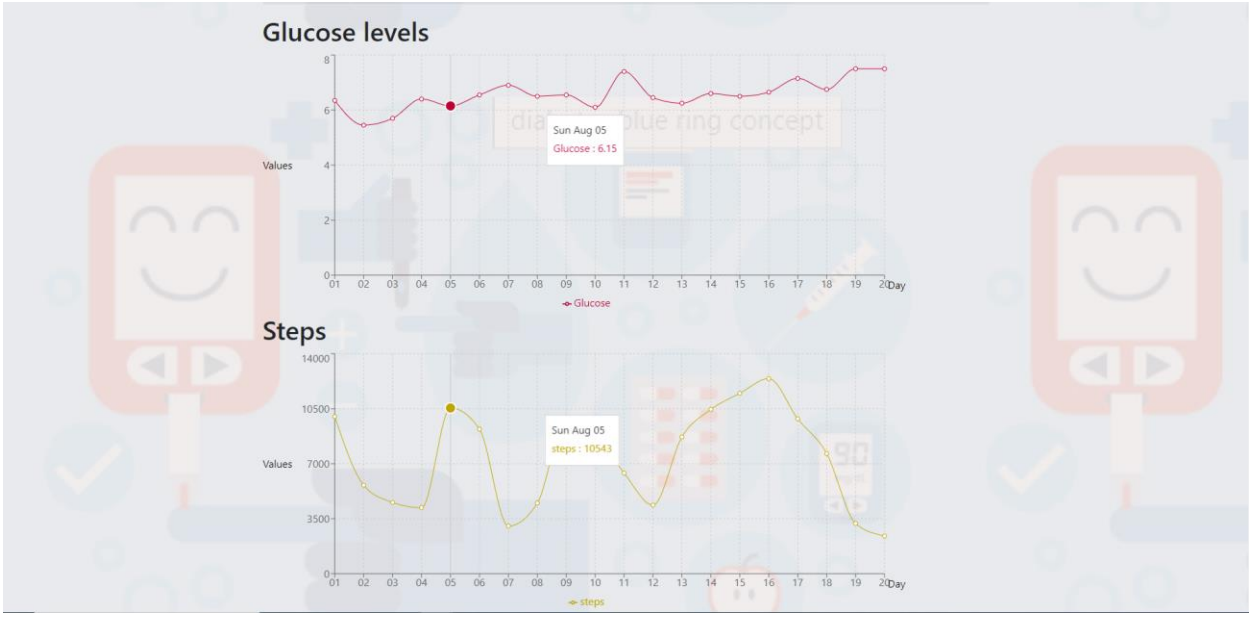
Slika 6.9. Prikaz korisničkih podataka.

Korisnik može filtrirati podatke po mjesecu i godini. Pristupom sučelju prikazani su podatci za trenutni mjesec (slika 6.10).



Slika 6.10. Grafički prikaz razine glukoze prije i poslije testa.

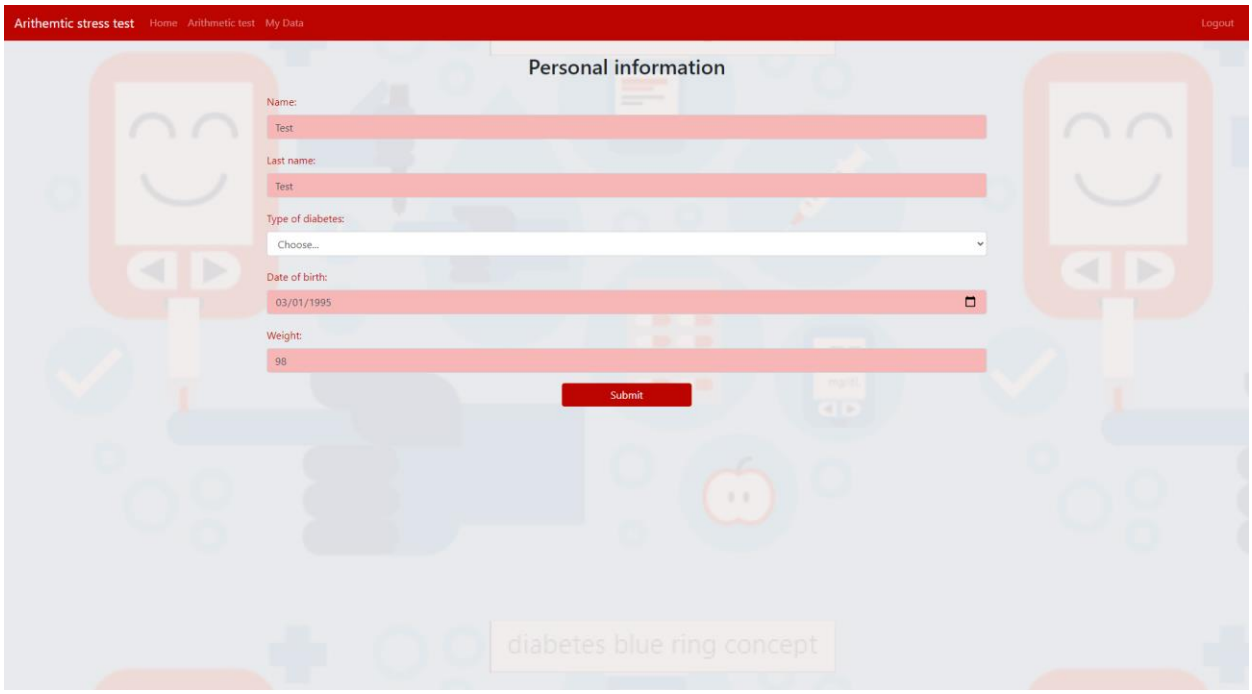
Klikom na gumb *Glucose levels*, na slici 6.11 prikazuju se dva grafa „*Glucose levels*“ i „*Steps*“. Prvi graf prikazuje razine glukoze u nekom vremenu, dok drugi graf prikazuje broj koraka u nekom vremenu.



Slika 6.11. Grafički prikaz razine glukoze i koraka.

6.1.7 Sučelje za unos korisničkih podataka

Sučelje prikazano na slici 6.12 sastoji se od polja ime, prezime, tip dijabetesa, datum rođenja i težina te na njemu korisnik unosi i mijenja osobne podatke.



Slika 6.12. Forma za unos i promjenu korisničkih podataka.

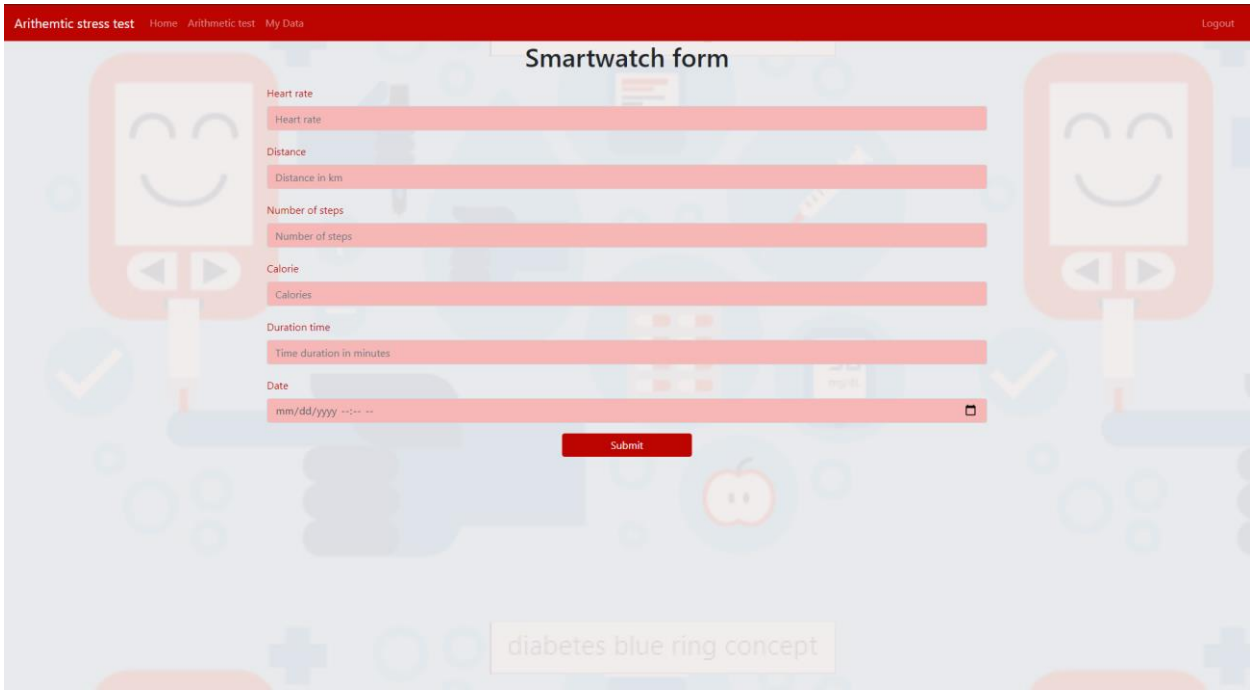
6.1.8 Sučelje za unos glukoze

U sučelju prikazanom na slici 6.13, korisnik unosi razinu glukoze, odabire je li mjereno prije ili poslije jela, te odabire datum i vrijeme kada je mjerenje odrađeno.

Slika 6.13. Forma za unos razine glukoze.

6.1.9 Sučelje za unos podataka sa pametnog sata

Forma prema slici 6.14 omogućuje korisniku unos podataka s pametnog sata. Forma sadrži polja kao što su otkucaji srca, udaljenost, broj koraka, potrošene kalorije, trajanje i datum.



Slika 6.14. Forma za unos podataka s pametnog uređaja.

6.2 Analiza rezultata učinka aritmetičkog stres testa

Korisnik pokreće aritmetički test s postavkom težine „Easy“ u trajanju od 3 minute, izmjerena razina glukoze prije testa je 5.0 mmol. Nakon provedenog aritmetičkog testa razina izmjerena razina glukoze je 6.0. Drugi slučaj korisnik pokreće test s postavkom težine „Easy“ ali u vremenu trajanja od 5 minuta. Izmjerena razina glukoze prije testa iznosila je 5.0 mmol, a nakon testa iznosila je 6.5 mmol. Tablica 6.1 prikazuje sve slučajeve provedbe aritmetičkih testova. Iz toga se može vidjeti da svi testovi podižu razinu glukoze u krvi. Na razinu glukoze u krvi, najmanje utječu testovi postavke „Easy“, neovisno o vremenu trajanja. „Medium“ postavka trajanja izaziva povećanje razine glukoze u krvi više nego „Easy“, ali manje od „Hard“ težine, neovisno o vremenu trajanja. Značajno povišenje razine glukoze u krvi uzrokuje postavka težine „Hard“ na svim vremenskim trajanjima. Iz navedenog se može zaključiti da stres utječe na razinu glukoze u krvi, a koliko će se povisiti razina glukoze u krvi, ovisi o intenzitetu i vremenu trajanja utjecaja stresa izazvanog aritmetičkim stres testom.

Tablica 6.1. Rezultati aritmetičkog testa.

Slučaj	Postavke težine	Vrijeme	Glukoza prije mmol	Glukoza poslije mmol
1	„Easy“	3 minute	5.0	6.0
2	„Easy“	5 minuta	5.0	6.5
3	„Easy“	7 minuta	5.1	6.9
4	„Medium“	3 minute	4.8	7.0
5	„Medium“	5 minuta	5.0	7.3
6	„Medium“	7 minuta	5.3	7.8
7	„Hard“	3 minute	5.0	8.0
8	„Hard“	5 minuta	4.7	8.4
9	„Hard“	7 minuta	5.3	9.0

7 ZAKLJUČAK

Web sustav za praćenje oboljelih od dijabetesa u ovisnosti o izazvanom stresu prikazuje prednosti primjene računalnih tehnologija u medicini. Uglavnom korisnici ovog sustava mogu biti osobe oboljele od dijabetesa te im sustav pomaže u praćenju parametara dijabetesa i pokazuje koliko izazvani stres utječe na njihovu razinu glukoze. Također, korisnik može unositi broj koraka tokom dana i razinu glukoze te na taj način može pratiti postoji li poveznica između kretanja tokom dana i razine glukoze.

Kako bi programsko rješenje bilo što bolje izrađeno, definirani su funkcionalni i nefunkcionalni zahtjevi koje aplikacija treba sadržavati. Prema tim zahtjevima, agilnim načinom, razvijeno je idejno rješenje korištenjem tehnologija React i Firestore. Kako je razvoj aplikacije temeljen na pristupu razvoja pokretanog testiranjem, sustav ima visoku razinu točnosti i ispravnosti prikaza podataka na sučelju. Sustav omogućuje korisnicima izazivanje stresa aritmetičkom metodom oduzimanja brojeva te im omogućuje unos razine glukoze i kretanja tokom dana s pametnog sata. Na osnovu unesenih podataka, korisnik ima uvid u grafički prikaz tih podataka. Sučelje sustava jednostavno je za korištenje, što poboljšava korisničko iskustvo te omogućuje korištenje sustava većem broju korisnika.

Razvijeni web sustav može se unaprijediti dodavanjem novih načina izazivanja stresa, te unaprjeđivanjem postojećih aritmetičkih testova. Također, sustav se može unaprijediti dodavanjem novih oblika analize grafičkog prikaza podataka.

LITERATURA

[1] R. Williams, S. Colagiuri, IDF Diabetes atlas, 2019.

[2] D. UK, »Complications of Diabetes,« Diabetes UK, [Mrežno]. Available: https://www.diabetes.org.uk/guide-to-diabetes/complications. [Pokušaj pristupa 6. 6. 2021.].

[3] M. Clinic, »A1C test,« Mayo Clinic, 2019. [Mrežno]. Available: https://www.mayoclinic.org/tests-procedures/a1c-test/about/pac-20384643. [Pokušaj pristupa 6. 6. 2021.].

[4] C. Lloyd, J. Smith and K. Weinger, »Diabetes Spectrum,« *Stress and Diabetes: A Review of the Links*, svez. 18, br. 2, pp. 121-127, 2005.

[5] »Stress and Diabetes,« Diabetes UK, [Mrežno]. Available: https://www.diabetes.org.uk/guide-to-diabetes/emotions/stress. [Pokušaj pristupa 6. 6. 2021.].

[6] »Stress: How It Affects Diabetes and How to Decrease It,« Healthline, 2020. [Mrežno]. Available: https://www.healthline.com/health/diabetes-and-stress#types-of-stress. [Pokušaj pristupa 6. 6. 2021.].

[7] S. R. Colberg, R. J. Sigal, J. E. Yardley, M. C. Riddell, D. W. Dunstan, P. C. Dempsey, E. S. Horton, K. Castorino and D. F. Tate, »Diabetes Care,« *Physical Activity/Exercise and Diabetes: A Position Statement of the American Diabetes Association,* svez. 39, br. 11, pp. 2065-2079, 2016.

[8] M. Nakao, T. Shimosawa, S. Nomura, T. Kuboki, T. Fujita and K. Murata, »American Journal of Hypertension,« *Mental Arithmetic Is a Useful Diagnostic Evaluation in White Coat Hypertension,* svez. 11, br. 1, pp. 41-45, 1998.

[9] W. L. Wasmund, E. C. Westerholm, D. E. Watenpaugh, S. L. Wasmund and M. L. Smith, »Interactive mental and physical stress,« *Interactive effects of mental and physical stress on cardiovascular control,* p. 1828–1834, 2002.

[10] P. Armario, R. H. del Rey, M. Martin-Baranera, M. C. Almendros, L. M. Ceresuela & H. Pardell, »Journal of Human Hypertension volume,« *Blood pressure reactivity to mental stress task as a determinant of sustained hypertension after 5 years of follow-up,* pp. 181-186, 2003.

[11] A. Cavka, M. Stupin, A. Panduric, A. Plazibat, A. Cosic, L. Rasic, Z. Debeljak, G. Martinovic and I. Drenjancevic, »Regulation of Cardiovascular Metabolism by Hormones and Growth Factors,« *Adrenergic System Activation Mediates Changes in Cardiovascular and Psychomotoric Reactions in Young Individuals after Red Bull© Energy Drink Consumption,* 2015.

[12] Wikipedia, »Blood glucose monitoring,« [Mrežno]. Available: https://en.wikipedia.org/wiki/Blood_glucose_monitoring. [Pokušaj pristupa 6. 6. 2021].

[13] »FreeStyle Libre,« Diabetes.co.uk, 2019. [Mrežno]. Available: https://www.diabetes.co.uk/blood-glucose-meters/abbott-freestyle-libre.html. [Pokušaj pristupa 6. 6. 2021].

[14] »Smartwatch,« Wikipedia, [Mrežno]. Available: https://en.wikipedia.org/wiki/Smartwatch. [Pokušaj pristupa 6. 6. 2021.].

[15] »Beyond Blood Sugar: How My Smartwatch Helps Me Manage My Diabetes,« OnTruck Diabetes, [Mrežno]. Available: https://www.ontrackdiabetes.com/live-well/diabetes-management/beyond-blood-sugar-how-my-smartwatch-helps-me-manage-my-diabetes. [Pokušaj pristupa 6. 6. 2021.].

[16] »Agile Methodology: What is Agile Software Development Model?,« Guru99, [Mrežno]. Available: https://www.guru99.com/agile-scrum-extreme-testing.html. [Pokušaj pristupa 6. 6. 2021.].

[17] M. C. Layton and S. J. Ostermiller, Agile Project Management for dummies, New Jersey, 2017.

[18] »Principles behind the Agile Manifesto,« Agilemanifesto, [Mrežno]. Available: https://agilemanifesto.org/principles.html. [Pokušaj pristupa 6. 6. 2021.].

[19] Guru99, »What is Software Testing? Definition, Basics & Types in Software Engineering,« [Mrežno]. Available: https://www.guru99.com/software-testing-introduction-importance.html. [Pokušaj pristupa 9 7 2021].

[20] S. t. material, »What Is Software Testing | Everything You Should Know,« [Mrežno]. Available: https://www.softwaretestingmaterial.com/software-testing/#h-unit-testing. [Pokušaj pristupa 9 7 2021].

[21] »What Is Software Testing | Everything You Should Know,« Software Testing Material, [Mrežno]. Available: https://www.softwaretestingmaterial.com/software-testing/. [Pokušaj pristupa 6. 6. 2021.].

[22] Wikipedia, »Software testing,« [Mrežno]. Available: https://en.wikipedia.org/wiki/Software_testing. [Pokušaj pristupa 9 7 2021].

[23] I. Sommerville, Software Engineering, Boston, 2011.

[24] J. Qiu, Matering Test Driven Development with React, 2019..

[25] »Introduction to Test Driven Development (TDD),« Agile data, 2003.. [Mrežno]. Available: http://agiledata.org/essays/tdd.html. [Pokušaj pristupa 6. 6. 2021.].

[26] »Test-driven development,« Wikipedia, [Mrežno]. Available: https://en.wikipedia.org/wiki/Test-driven_development?fbclid=IwAR2D9xPO8djoNHgExKbYHt6mOGfHPAjjaXwj54FnOYDACHKj-YMi9Zw789c. [Pokušaj pristupa 6. 6. 2021.].

[27] »Cloud Firestore,« Firebase, [Mrežno]. Available: https://firebase.google.com/docs/firestore. [Pokušaj pristupa 6. 6. 2021.].

[28] »Testing Library,« [Mrežno]. Available: https://testing-library.com/docs/. [Pokušaj pristupa 15 8 2021].

[29] »Jest,« [Mrežno]. Available: https://jestjs.io/. [Pokušaj pristupa 15 8 2021].

SAŽETAK

U ovom diplomskom radu razvijen je web sustav za praćenje stanja oboljelih od dijabetesa u ovisnosti o izazvanom stresu, s ciljem da pomogne i omogući uvid u utjecaj stresa razinu glukoze. Aplikacija omogućuje korisniku spremanje razine glukoze i podataka o aktivnosti korisnika s pametnog uređaja te njihov grafički prikaz. Korisnik na grafičkom prikazu može vidjeti utjecaj stresa na razinu glukoze, kao i postoji li povezanost između kretanja i razine glukoze. Tijekom razvoja aplikacije, korištena su dva pristupa, a to su agilni razvoj i razvoj pokretan testiranjem. Na kraju razvoja svake funkcionalnosti provedeno je ručno testiranje. Aplikacija je razvijana koristeći razvojni okvir React, a za potrebe spremanja podatka korištena je baza podataka u oblaku računala Firestore. Za potrebe razvoja i pokretanja testova korištena su dva okvira Jest i React Testing Library. U radu je također opisana problematika dijabetesa kao i pristupi i tehnologije korišteni u razvoju web aplikacije.

Ključne riječi: agilni razvoj, aritmetički stres test, dijabetes, razvoj pokretan testiranjem, web aplikacija.

ABSTRACT

Title: Agile development of a web system for monitoring the condition of diabetics depending on the stress caused

In this thesis, a web system was developed to monitor the condition of diabetics depending on the stress caused, with the aim of helping and providing insight into the impact of stress on glucose levels. The application allows the user to save glucose levels and user activity data from a smart device and display them graphically. The user can see the effect of stress on glucose levels on the chart, as well as whether there is a relationship between movement and glucose levels. During application development, two approaches were used, namely agile development and testing-driven development. At the end of the development of each functionality, manual testing was conducted. The application was developed using the React development framework, and the Firestore database in the cloud was used to store the data. Jest and React test library frames were used to develop and run the tests. The paper also describes the issue of diabetes, as well as the approaches and technologies used in web application development.

Keywords: agile development, arithmetic stress test, diabetes, testing driven development, web application.

ŽIVOTOPIS

Mario Džoić rođen je 30. lipnja 1995. godine u Vinkovcima. Završio je osnovnu školu Mate Lovraka u Županji. Nakon osnovne škole, upisuje srednju Elektrotehničku školu u Županji, a po završetku i Fakultet elektrotehnike, računarstva i informacijskih tehnologija na Sveučilištu J. J. Strossmayera u Osijeku. Na četvrtoj godini zapošljava se u tvrtki Mono, a na petoj godini u tvrtki Kod savjetovanje gdje radi i danas.

PRILOZI

1. Dokument rada u formatu docx
2. Dokument rada u formatu pdf
3. Programski kod web sustava i testova u .zip datoteci