Rīgas 64. Vidusskola

AUTOMĀTISKA SLIMĪBU NOTEIKŠANAS SISTĒMA.

Programmatūras specifikācija

Autors: **Gatis Petrovskis**

Darba vadītājs: **Edvards Bukovskis**

Rīga 2023

Saturs

Definīcijas, akronīmi un saīsinājumi

ASNS	Automātiska slimību noteikšanas sistēma
API	Saīsinājums no angļu valodas "*Application Programming Interface*" jeb latviski "Lietojumprogrammas saskarne".
Copyleft	Atvērtā pirmkoda licence, kas garantē brīvību izmantot, modificēt un izplatīt programmatūru ar nosacījumu, ka uz modificēto versīju tiks attiecināti tādi paši noteikumi, kā uz sākotnējo programmatūras versīju.
Permissive	Atvērtā pirmkoda licence, kas garantē brīvību izmantot, modificēt un izplatīt, vienlaikus atļaujot arī patentēt modificētos darbus
PPS	Programmatūras prasību specifikācija.
Python	Augsta līmeņa interpretējama objektorientētā programmēšanas valoda.
MVP	Agrīns izstrādes produkts, kas izpilda nosacītās pamatfunkcijas, no angļu valodas "*Minimum Viable Product* ".
(API) token	Unikāls lietotāja API identificētājs, kas ļauj gūt piekļuvi API datiem un funkcijām.

1. Ievads

1.1. Nolūks

Ņemot vērā pēdējo gadu tendenci uz sabiedrības straujo velmi pēc veselīga un aktīva dzīvesveida, it īpaši pēc Covid-19 pandēmijas sākuma, kā arī arvien pieaugošo medicīnas nozares noslodzi ar pacientiem, kuru diagnozēm nebūtu nepieciešama ārsta vizīte. Problēmas risināšanai ir nepieciešama automatizēta diagnožu noteikšanas sistēma, kura balstoties uz ievadītajiem simptomiem spēj noteikt diagnozi un, ja nepieciešams, ārstēšanai nepieciešamās zāles.

Dokuments ir sagatavots programmēšanas II kursa pielaides darbs. Dokumentā tiek aprakstīta ASNS struktūra un funkcionalitāte, kā arī dokumetns sevī ietver gatavā produkta redzējumu un tā daļēju tehnisko izpildījumu MVP formātā, kas būtu kā pamats turpmākai sistēmas attīstībai.

1.2. Darbības sfēra

ASNS ir paredzēts intergrēt Valsts medicīnas sistēmā, kā vienu no palīdzības sniegšanas veidiem gadījumos, kad ārsta vizīte nav pieejama pārredzamā laika periodā vai arī tad, kad pacients vēlas uzzināt papildus informāciju par simptomu iespējami izraisītajām slimībām un to ārstēšanai nepieciešamajām zālēm. ASNS būtu apakšsistēma jau esošai “e-veselības” sistēmai.

1.3. Saistība ar citiem dokumentiem

1. Dokumenta PPS noformēšanā ievērotas standarta LVS 68:1996 prasības.

2. Dokumenta PPS kalpo kā atsauce turpmākiem sistēmas uzlabojumiem.

1.3. Pārskats

Pirmajā daļā ir ievadinformācija un vispārējs apraksts, kas satur dokumenta nolūku, definīciju skaidrojumu, saistību ar citiem dokumentiem un sistēmas darbības sfēru.

Otrā daļa sniedz pārskatu par sistēmas esošo stāvokli, sistēmas perspektīvu un vispārēju sistēmas funkcionalitāti un būtību.

Trešajā daļā tiek aprakstītas funkcionālās prasības un nefunkcionālās prasības

Ceturtajā daļā ir piemērojamās licences pamatojums.

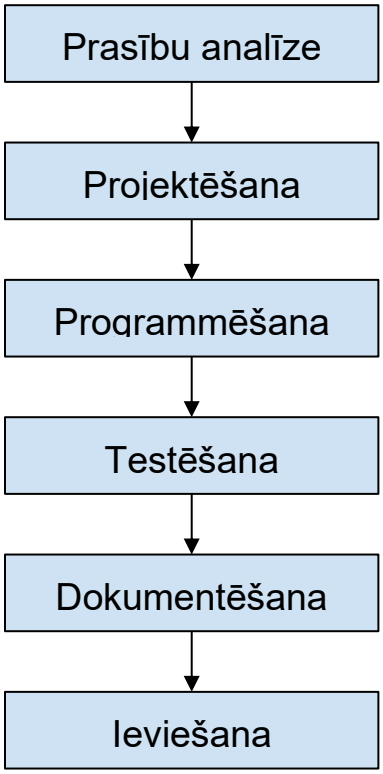
Piektajā daļā tiek aprakstīts programmatūras darbības un ieviešanas plāns.

2. Vispārējais apraksts

2.1. Programmatūras izstrādes plāns

Programmatūras izstrāde tiek veikta pēc lineārā ūdenskrituma modeļa, jo katrs no izstrādes posmiem ir jādara secīgi, tādējādi izstrādes procesu padarot ne tikai loģiskāku, bet arī samazinot iespejamību pieļaut kļūdas programmas izstrādē un paaugstinot izstrādes kvalitāti, jo pēc katra izstrādes posma tiek analizēti iegūtie rezultāti.

Izstrādāta programmatūra pēc būtības ir mazapjomīgs projekts, kā arī tās izstrādē nav daudz iesaistīto pušu un nav nepieciešama sistēmas lietotāju iesaiste, rezultējoši izvēlētais modelis ir visatbilstošākais (skat. att. 1).



Att. 1. Programmatūras izstrādes modelis

2.2 Detalizēti izstrādes posmi

1. Prasību analīzē - tiktu veikta aptauja par vispārējām problēmām, ar kurām iedzīvotāji saskārušies iespējamās slimības noteikšanas procesā, apmeklējot ārstu. Kā arī noskaidrotu iespējamo sistēmas lietotāju attieksmi par konkrētās sistēmas izstrādāšanu un ieviešanu. Šis solis aizņemtu aptuveni 1 mēnesi, jo būtu nepieciešams aptauju saskaņot ar attiecīgajām varas iestādēm un slimnīcu vadību. Aptauju veiktu ikviens simnīcu klients, kā arī tās tiktu publicētas “e-veselības” interneta platformā.
2. Projektēšanā - analizēti aptaujas rezultāti, līdz ar to arī teorētiski izstrādātas galvenās programmas funkcionalitātes, kuras nepieciešams integrēt sistēmā. Projektēšana aizņemtu

aptuveni 2 līdz 3 nedēļas, jo šajā posmā būtu jāapkopo dati, kam nebūtu nepieciešams atvēlēt lielu laika periodu.

3. Programmēšana - balstoties uz iepriekšējā izstrādes posmā teorētiski izstrādātajām programmas funkcijām, praktiska to realizēšana "Python" programmēšanas valodā. Atkarībā no izstrādātās programmas sarežģītības, programmēšana būtu viens no laikietilpīgākajiem posmiem, jo tiktu izstrādāta programmas pamatbāze. Līdz ar to pilnvērtīga koda izstrādāšana varētu aizņemt vismaz 1 mēnesi.
4. Testēšana - tiktu pārbaudīta programmas funkcionalitāte. Atkarībā no tā, kādi papildus uzlabojumi būtu jāievieš, konkrētais izstrādes posms aizņemtu aptuveni 2 līdz 3 nedēļas.
5. Dokumentēšana - tā kā deļēji visas sistēmas funkcijas būtu jau dokumentētas projektēšanas posmā, tad šajā izstrādes posmā tiktu apkopota visa informācija, kam nepieciešamas aptuveni 2 nedēļas.
6. Ieviešana - pēc veiksmīgas programmas izstrādes un iegūta gala rezultāta, sākotnējo programmas versīju padarītu pieejamu ikvienam lietotājam internetā. Katram no programmas lietotājiem būtu iespēja sniegt savus iespaidus par programmu, tās trūkumiem un nepieciešamajiem uzlabojumiem, kā arī sniegt kopējo vērtējumu par tās nepieciešamību un lietderību. Pēc tās sākotnējās versījas pilnveidošanas, programma tiktu integrēta "e-veselibas" sistēmā, kā viena no tās papildus funkcijām. Ieviešanas posmam nepieciešamais termiņš nav paredzams, jo tas ir atkarīgs no tā, cik programma stabili funkcionēs un cik apjomīgi uzlabojumi tai būs nepieciešami pēc sakotnējās versījas publicēšanas.

2.3. Produkta perspektīva

Sistēma pati par sevi ir neatkarīga, tāpēc to ir iespējams pielāgot un modificēt citām vajadzībām. Konkrētajā gadījumā, kad sistēma kalpo kā apakšsistēma jau esošai sistēmai, tai ir iespējams pievienot papildu funkcionalitātes. Kā, piemēram, sistēmā būtu iespējams pievienot detalizētākus simptomus, lai lietotājs iegūtu prezīzākus rezultātus, kā arī pievienot ne tikai ārstēšanai nepieciešamās zāles, bet arī detalizētus ārstēšanas veidus.

2.4. Produkta funkcijas

Diagnozes atpazīšana balstoties uz vairākiem lietotāja ievadītajiem faktoriem - slimības simptomiem, vecuma un dzimuma.

Atkarībā no iegūtas diagnozes noteikt pie kādas medicīnas nozares speciālistiem lietotājam jāvēršas turpmākai palīdzībai.

2.5. Sistēmas funkcionalitāte

1. Datu ieguve
2. Datu pārbaude
3. Datu apstrāde
4. Atbildes sniegšana

2.6. Lietotāja raksturiezīmes

Sistēmas tiešie lietotāji ir ikviens sabiedrības loceklis, kuru skārušas veselības problēmas vai nepieciešama papildus informācija par to ārstēšanas iespējām un iespējamo diagnozi.

2.7. Vispārējie ierobežojumi

Sistēmas darbībai ir nepieciešami ievaddati no lietotāja un interneta savienojums.

2.8. Pieņēmumi un atkarības

ASNS darbībai tiek pieņemts, ka lietotāja ievadītie dati ir patiesi. Sistēmas funkciju veikšana ir atkarīga no *APIMedic* API.

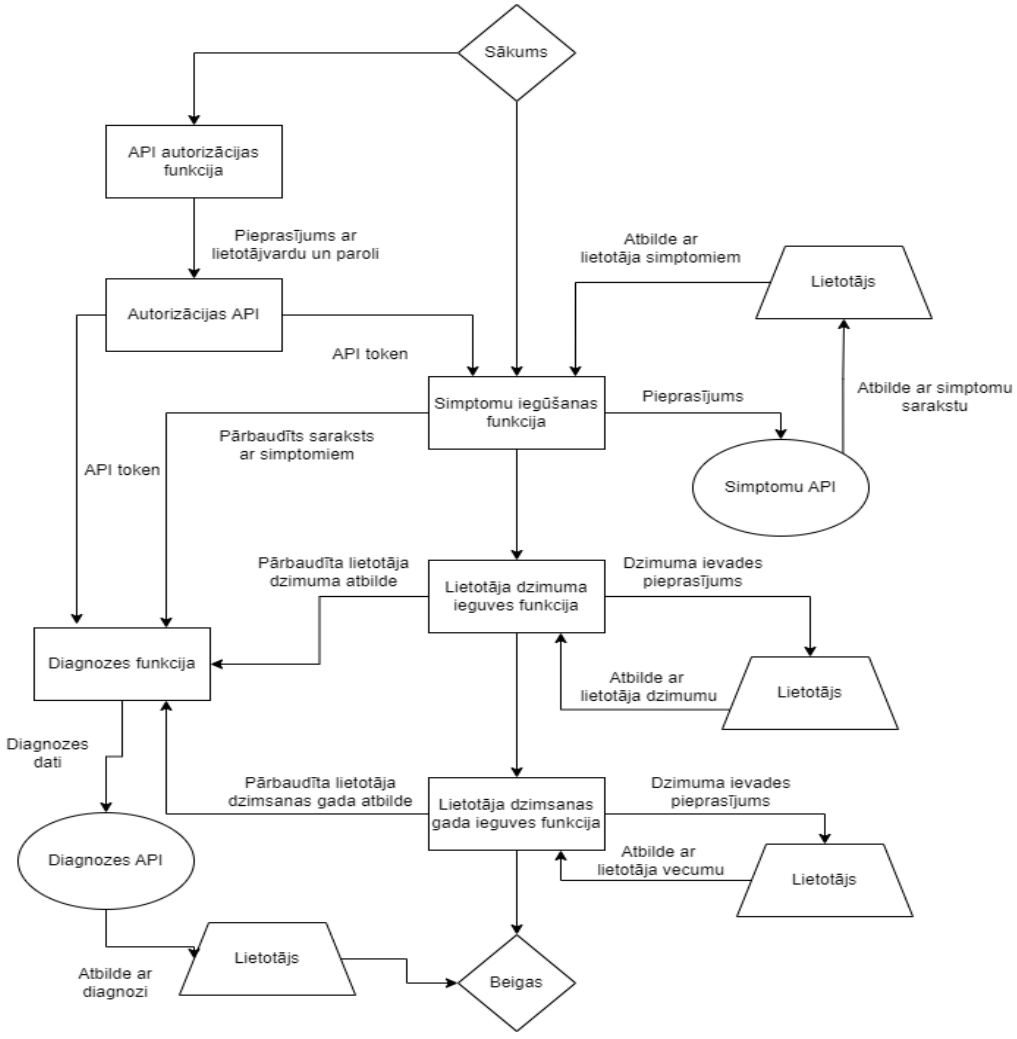
3. Konkrētās prasības

3.1. Funkcionālās prasības

ASNS būtība ir veikt medicīnisko diagnozi balstoties uz lietotāja iepriekš ievadītajiem datiem. Šīs prasības izpildīšanā tika izmantots *APIMedic* diagnozes un simptomu API uz kā tiek balstīta galvenā MVP funkcionalitāte.[2.][3.]

Vissvarīgākās ASNS programmas funkcionālās prasības ir API token pieprasījuma veikšana un iegūšana tālākai izmantošanai autorizācijā, *APIMedic* pieejamo simptomu saraksta sagatave, lietotāja simptomu datu ievākšana un pārbaude, lietotāja dzimuma datu ievākšana un atbildes pārbaude, lietotāja vecuma ievākšana un pārbaude, ievākto datu apstrāde, sagatave un interneta adreses veidošana, kā arī diagnozes datu iegūšana, apstrāde un izvade.

Balstoties uz šīm prasībām tika izstrādāts MVP, tas tika veidots Python programmēšanas valodā ar mērķi izpildīt iepriekš nosauktās prasības.



Att. 3. ASNS darbības modelis

API token iegūšanas funkcija

Mērķis:
Izveidot API token, lai veiktu autorizāciju un varētu izmantot *APIMedic* funkcijas.
Ievaddati:
API autorizācijas saites adrese, API piešķirtais lietotājvārds un parole.
Apstrāde:
Tiek izveidots šifrēts pieprasījums uz autorizācijas saiti ar lietotāja informāciju, izmantojot *requests* bibliotēku.
Izvaddati:
API token.

Tabula 1

Simptomu iegūšanas funkcija

Mērķis:
Iegūt sarakstu ar lietotāja simptomiem no API simptomu saraksta.
Ievaddati:
API token, Simptomu API interneta adrese, lietotāja izvēlētie simptomi.
Apstrāde:
Apkopo un parāda tabula ar simptomiem no kā lietotājam jāizvēlas, izvēlētie simptomi tiek ielikti sarakstā un tiek pārbaudīta atbilstība ar tabulā esošiem simptomiem.
Izvaddati:
Saraksts ar lietotāja izvēlētajiem simptomiem.

Tabula 2

Lietotāja bioloģiskā dzimuma iegūšanas funkcija

Mērķis:
Iegūt lietotāja bioloģisko dzimumu
Ievaddati:
Lietotāja ievadītais dzimums.
Apstrāde:
Lietotāja izvēlētā atbilde tiek pārbaudīta, vai atbilst nosacījumiem (Vīrietis vai Sieviete).
Izvaddati:
Lietotāja bioloģiskais dzimums.

Tabula 3

Dzimšanas gada ieguves funkcija

Mērķis:
Iegūt lietotāja dzimšanas gadu.
Ievaddati:
Lietotāja ievadītais vecums.
Apstrāde:
Lietotāja atbilde tiek pārbaudīta, nosacījumam vesels skaitlis robežās no 0 līdz 100, iegūts dzimšanas gads no vecuma.
Izvaddati:
Lietotāja dzimšanas gads.

Tabula 4

Diagnozes funkcija

Mērķis:

Parādīt lietotājam iegūto diagnozi no ievadītajiem datiem.
Ievaddati:
API token, diagnozes API interneta adrese, lietotāja simptomi, dzimums un dzimšanas gads.
Apstrāde:
Lietotāja ievadītie dati tiek ievietoti API, iegūst atbildi no diagnozes API. Atbilde tiek pārveidota un apkopota pārskatāmā tabulā.
Izvaddati:
Lietotājam izvadīta diagnozes tabula ar slimību nosaukumiem, aprakstu un id, kā arī simptomu procentuālo atbilstību, un ārstējošo medicīnas nozari.

Tabula 5

3.2. Veiktspējas prasības

Sistēmai ir jādarbojas ar augstu precizitāti, jo pretēja gadījuma sistēmas sniegtais atbalsts lietotājam nepaātrinās informācijas ieģūšanas ātrumu, līdz ar to būs līdzvērtīgs ar laiku, kurā iespējams apmeklēt ārstu, kā arī neatvieglos ne lietotāja, ne ārsta ikdienu, kuru darba slogs līdz ar to paliktu nemainīgi augsts.

Tā kā programma balstās uz API datiem nevis datu bāzē glabātiem datiem, kā arī izveidotais kods nav apjomīgs, tad sistēmas veiktspēja ir paredzama kā ļoti optimāla.

3.3. Atribūti

3.3.1. Drošība

Sistēmas lietotāja ievadītājiem datiem ir jābūt droši glabātiem, lai tiem nav iespējams piekļūt no ārējām sistēmām. Ievadītie dati automātiski jāizdzēš pēc lietotāja atslēgšanās no sistēmas.

Piekļuvi sistēmas lietotāju ievadītajiem datiem, kā arī to nodošana iesaistītajām pusēm vai trešajām peronām netiek nodrošināta.

No lietotāja ievāktie personas dati tiek šifrēti, lai nodrošinātu lielāku datu aizsardzību un izvairītos no personas datu aizsardzības likuma laušanas.

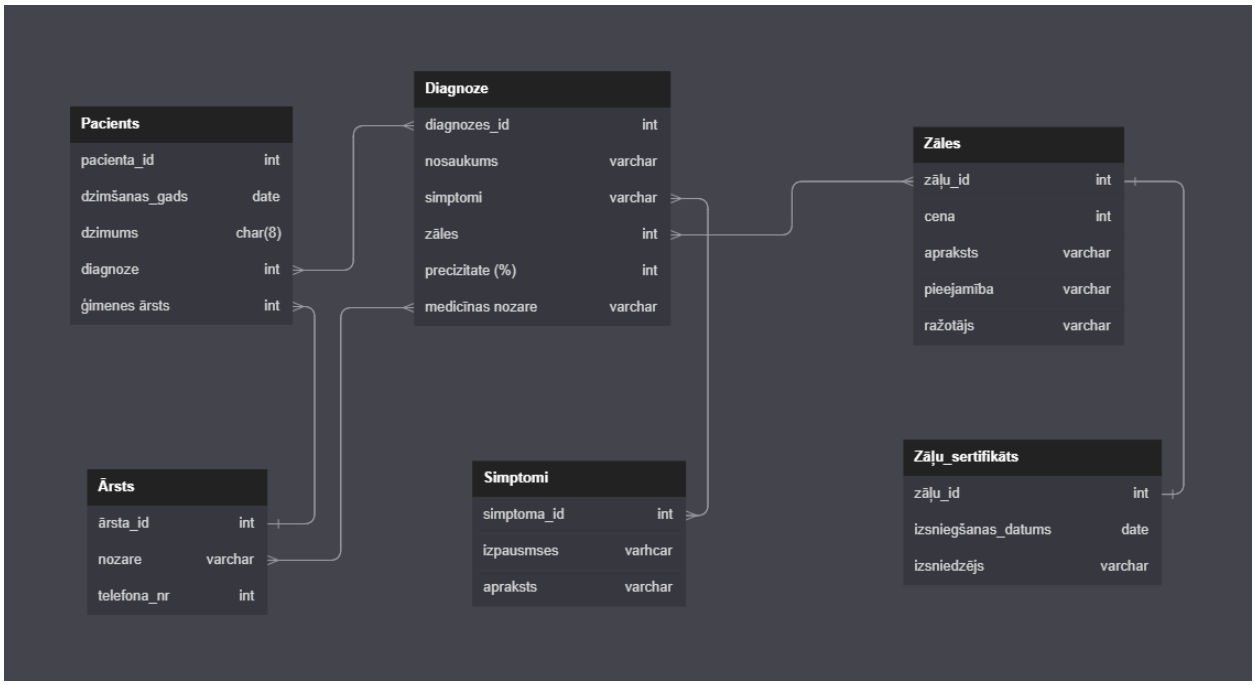
3.3.2. Uzturamība

Programmas kodam ir jābūt strukturētam, viegli uztveramam un būtiskākajām koda rindām - paskaidrotām, kā arī jāievēro labs programmēšanas stils.

3.4. Citas prasības

3.4.1. Datu bāze

ASNS gala versijā tiks iesaistīta datu bāze, lai uzglabātu vajadzīgos datus un nodrošinātu vieglāku informācijas apmaiņu diagnozes sistēmā. Datu bāze tiks veidota ar datu bāžu pārvaldības sistēmu PostgreSQL, jo tā spēj uzglabāt lielu datu apjomu un ar lielu drošības pakāpi, tādēļ ir piemērota šim gadījumam.



Att. 3. ASNS Datu bāzes relāciju modelis

3.5. Nefunkcionālās prasības

Tā kā izstrādātās programmas mērķis ir atvieglot medicīnas nozares noslodzi un paātrināt slimību noteikšanai nepieciešamo laiku, tad izstrādātajai sistēmai ir jābūt ērti lietojamai un viegli uztveramai programmas sniegtajai informācijai ikvienam lietotājam saprotamai un skaidrai.

4. Piemērotā licence

No šobrīd diviem galvenajiem atvērtā koda licenču veidiem - “copyleft” un “permissive” - darba autors kā atbilstošako izstrādātajai programmatūrai uzskata “copyleft” licenci [1]. Tā kā izstrādātā programma ir ar mērķi uzlabot un atvieglot sabiedrības ikdienu medicīnas nozarē, tad jebkurai modificētajai programmai būtu jābūt ar mērķi nevis katru tās versīju patentēt, kā tas ir “permissive” licences gadījumā, bet gan katru nākošo modificēto kodu ar tādiem pašiem noteikumiem kā sākotnējo kodu padarīt pieejamu ikvienam [1]. Tā kā galvenais mērķis izveidotajai programmai ir nevis komerciāls, bet gan uz kopējo sabiedrības ieguvumu vērsts, tad vispiemērotākais licences veids ir “copyleft”, kur ikvienam kas vēlas būtu iespējams iesaistīties programmas attīstīšanā.

5. Atkļūdošanas un akcepttestēšanas pārskats.

ASNS izstrādē tika veikti testi, lai pārbaudītu sistēmas darbību. Testi tika veikti funkcijās ar lielāko kļūdu iespējamību, kuras prasa lietotāja ievadītus datus.

ID	Ievaddati	Sagaidāmais rezultāts	Reālais rezultāts
simptomu_iegusana	Ievadē ir skaitļu virkne vai skaitlis, kur visi iekļauti simptomu sarakstā (44,9,101) (263)	Funkcija atgriež sarakstu	Izpildās
Apraksts	Ievadē ir skaitļu virkne, kur ne visi ir iekļauti simptomu sarakstā (136,234,10000) (5000, 11, 203)	Funkcija neatgriež sarakstu, izvada paziņojumu, ka vēlreiz vērtības jāievada	Izpildās
Iegūst sarakstu ar lietotāja simptomiem no API simptomu saraksta (tiek gaidits skaitlis vai vairāki skaitļi atdalīti ar komatu, kuri atrodas simptomu sarakstā, piemēram - 9,10,11)	Ievadē ir virkne, kas satur vārdu nevis skaitli. (101,piecdesmit,123) (157, 221,piemers)	Funkcija neatgriež sarakstu, izvada paziņojumu, ka vēlreiz vērtības jāievada	Izpildās

ID	Ievaddati	Sagaidāmais rezultāts	Reālais rezultāts
dzimuma_iegusana	Ievadē ir vārds, kas atbilst nosacījumiem. (vīrietis) (sieviete)	Funkcija atgriež lietotāja dzimumu	Izpildās
Apraksts	Ievadē ir vārds, kas neatbilst nosacījumiem. (cilvēks) (vilciens)	Funkcija neatgriež lietotāja dzimumu, izvada paziņojumu, ka vēlreiz vērtība jāievada	Izpildās

Tiek prasīts lietotājam ievadīt bioloģisko dzimumu (vīrietis vai sieviete)	Ievade saturs skaitli (5) (42)	Funkcija neatgriež lietotāja dzimumu, izvada paziņojumu, ka vēlreiz vērtība jāievada	Izpildās

ID	Ievaddati	Sagaidāmais rezultāts	Reālais rezultāts
dz_gada_iegusana	Ievade satur skaitli noteiktajās robežās (9) (65)	Tiek atgriezts lietotāja vecums	Izpildās
Apraksts	Ievade satur skaitli, kas nav noteiktajās robežās vai vesels (-2) (150) (5,5)	Netiek atgriezts lietotāja vecums, rādās paziņojums, ka vecums vēlreiz jāievada	Izpildās
Iegūst vecuma datus no lietotāja, pārvērš tos uz dzimšanas gadu, tiek gaidīts vecums no 0 līdz 100, kā vesels skaitlis	Ievade satur vārdu (divdesmit) (gadi)	Netiek atgriezts lietotāja vecums, rādās paziņojums, ka vecums vēlreiz jāievada	Izpildās

6. Lietotāja ceļvedis

6.1. Vienkāršots programmas darbības plāns

1. Programmā lietotājs izvēlas un apstiprina slimības simptomus no saraksta, ievada vecumu un dzimumu.
2. Programma sniedz lietotājam atbildi par slimībām ar vislielāko varbūtību, balstoties uz ievāktajiem lietotāja datiem, kā arī infromācija pie kādas nozares speciālistiem jāvēršas.

6.2. Programmatūras ieviešanas plāns

1. Pēc veiksmīgas programmas izstrādes un iegūta gala rezultāta, sākotnējo programmas versīju padarītu pieejamu ikvienam lietotājam internetā. Katram no programmas lietotājiem būtu iespēja sniegt savus iespaidus par programmu, tās trūkumiem un nepieciešamajiem uzlabojumiem, kā arī sniegt kopējo vērtējumu par tās nepieciešamību un lietderību.
2. Izstrādātās programmatūras, pēc tās sākotnējās versījas pilnveidošanas, integrēšana "e-veselibas" sistēmā, kā viena no tās papildus funkcijām, ja lietotājam nepieciešama steidzama slimības noteikšana un ja nepieciešams arstēšanai nepieciešamo zāļu noteikšana.

Izmantotās literatūras un avotu saraksts

1. “Atvērtā koda programmatūra”, bez autora, 2009. gads [Skatīts 2023. gada 17. martā]. Pieejams: Atvērtā pirmkoda produkta trūkumihttps://profizgl.lu.lv › mod › resource › view
2. APIMedic diagnozes API [Tiešsasiste] Pieejams: https://healthservice.priaid.ch/diagnosis
3. APIMedic simptomu API [Tiešsaiste] Pieejams: https://healthservice.priaid.ch/symptoms

4. “APIMedic API autorizācijas dokumentācija” [Tiešsaiste]. Pieejams: http://authservice.priaid.ch/docs.html
5. “APIMedic API dokumentācija” [Tiešsaiste]. Pieejams: http://healthservice.priaid.ch/docs.html

Pielikumi

1. Pielikums - ASNS pirmkods, Python programmēšanas valoda

```
import requests
import pandas as pd
import hmac
import base64
import json

#=================API token iegūšanas funkcija=====================
def token_iegusana(url, lietotajvards, parole):
    rawHashString      =      hmac.new(bytes(parole,      encoding='utf-8'),
url.encode('utf-8')).digest()
    computedHashString = base64.b64encode(rawHashString).decode()
    lietotaja_info = lietotajvards + ':' + computedHashString
    galvene = {
        'Authorization':  'Bearer  {}'.format(lietotaja_info)  #informācija
tiek ielikta galvenē pec piemēra api dokumentācijā
    }
    atbilde_token = requests.request("POST", url, headers=galvene)  #Veikts
pieprasījums uz APIMedic autorizācijas API
    dati_token = json.loads(atbilde_token.text)
    token = dati_token["Token"]
    return token    #Atgriezts API token


autorizacija_url = "https://authservice.priaid.ch/login"
lietotajvards = "lietotājvārds"     #APIMedic lietotājvārds
parole = "parole"   #APIMedic parole
token = token_iegusana(autorizacija_url, lietotajvards, parole) #Izmantota
api token iegūšanas funkcija

#==================Simptomu iegūšanas funkcija===========
def simptomu_iegusana(token):
    token = str(token)
    url_simptomi   =   "https://healthservice.priaid.ch/symptoms?token="   +
token + "&language=en-gb"
    response_simpt = requests.get(url_simptomi)
    simpt = response_simpt.json()
```

```
    df = pd.DataFrame(simpt)
    df.columns = ["Simptoma ID", "Simptoms"]
    pd.set_option('display.max_rows', None)
    pd.set_option('display.max_columns', None)   #Izveidots un pārtaisīts
dataframe ar simptomiem un to id
    print(df) #Parādīta tabula lietotājam
    simptomu_saraksts = []
    pacienta_simptomi  = input("Ludzu ievadiet jusu simptomu id ar komatu
bez atstarpes , piemeram (9,10,11): ").split(",")
    while True: #Tiek prasīta un pēc tam pārbaudīta lietotāja atbilde pēc
nosacījumiem ar while ciklu
        try:
            for a in pacienta_simptomi:
                a = int(a)
                if a in range(0, 1100):
                    simptomu_saraksts.append(a)
        except ValueError:
            print("Radusies kluda, atbilde neatbilst nosacijumiem \n")
            pacienta_simptomi  = input("Ludzu ievadiet jusu simptomu id ar
komatu bez atstarpes , piemeram (9,10,11): ").split(",")
        except len(simptomu_saraksts) == 0:
            print("Radusies kluda, atbilde neatbilst nosacijumiem \n")
            pacienta_simptomi  = input("Ludzu ievadiet jusu simptomu id ar
komatu bez atstarpes , piemeram (9,10,11): ").split(",")
        else:
            break
    return simptomu_saraksts    #Atgriezts lietotāja saraksts ar simptomiem
pēc pārbaudes

pacienta_simptomi  =  simptomu_iegusana(token)        #Izmantota  simptomu
iegūšanas funkcija


#=================Lietotāja dzimuma iegūšanas funkcija==============
def dzimuma_iegusana():
    x = True
    while x == True:
        pacienta_dzimums = input("Ievadiet dzimumu(Virietis vai Sieviete):
") #Ievākta lietotāja atbilde
```

```
        if pacienta_dzimums == "Virietis" or pacienta_dzimums == "virietis"
or pacienta_dzimums == "vīrietis" or pacienta_dzimums == "Vīrietis":   #Tiek
pārbaudīta lietotāja dzimuma atbilde
            pacienta_dzimums = "Male"
            x = False
        elif  pacienta_dzimums  ==  "Sieviete"  or  pacienta_dzimums  ==
"sieviete":
            pacienta_dzimums = "Female"
            x = False
        else:
            print("Radusies kluda, atbilde neatbilst nosacijumiem \n")
            x = True
    return pacienta_dzimums #atgriezts lietotāja dzimums

pacienta_dzimums  =  dzimuma_iegusana()      #Izmantota  lietotāja  dzimuma
iegūšanas funkcija


#============Lietotāja dzimšanas gada ieguves funkcija=========
def dz_gada_iegusana():
    while True:
        dz_gads = input("Ievadiet vecumu: ")    #Prasīts lietotaja vecums
        if dz_gads.isdigit() == True:
            dz_gads = int(dz_gads)
            if dz_gads in range(0, 100):
                break
            else:
                print("Radusies kluda, atbilde neatbilst nosacijumiem \n")

        else:
            print("Radusies  kluda,  atbilde  neatbilst  nosacijumiem  \n")
#Parbaudīta atbilde pēc nosacījumiem ar while ciklu
    dz_gads = 2023 - dz_gads    #Aprēķināts lietotāja dzimšanas gads
    return dz_gads  #Atgriezts dzimšanas gads

dz_gads = dz_gada_iegusana()     #Tiek izmantota lietotāja dzimšanas gada
ieguves funkcija


#===============Diagnozes funkcija=======================
```

```
def iegut_diagnozi(token, simptomi, dzimums, dz_gads):
    simptomi = str(simptomi)
    dz_gads = str(dz_gads)
    url_diagnoze    =    "https://healthservice.priaid.ch/diagnosis?token="+
token + "&language=en-gb&symptoms=" + simptomi + "&gender=" + dzimums +
"&year_of_birth=" + dz_gads #Visi iegūtie dati ielikti API saitē
    response_diagnoze = requests.get(url_diagnoze)
    json_diagnoze = response_diagnoze.json()
    if len(json_diagnoze) == 0: #Pārbauda vai lietotāja simptomi atbilst
kādai diagnozei
        print("Netika atrasta diagnoze, kas atbilst simptomiem")
        return json_diagnoze
    saraksts_diagnozes  =  []  #Saraksti  vajadzīgajām  vērtībām  dataframe
izveidei
    saraksts_slimibas = []
    saraksts_slimibas_id = []
    saraksts_arstejosa_nozare = []
    saraksts_atbilstiba = []
    saraksts_vieta = []
    saraksts_apraksts = []
    for i in json_diagnoze: #Vērtības tiek ieliktas sarakstā ar for cikliem
        diagnoze = i["Issue"]
        saraksts_diagnozes.append(diagnoze)

    for i in saraksts_diagnozes:
        slimibas = i["Name"]
        saraksts_slimibas.append(slimibas)

    for i in saraksts_diagnozes:
        slimibas_id = i["ID"]
        saraksts_slimibas_id.append(slimibas_id)

    for i in saraksts_diagnozes:
        precizitate = i["Accuracy"]
        saraksts_atbilstiba.append(precizitate)

    for i in saraksts_diagnozes:
        vieta = i["Ranking"]
        saraksts_vieta.append(vieta)
```

```
    for i in saraksts_diagnozes:
        vieta = i["IcdName"]
        saraksts_apraksts.append(vieta)

    for i in json_diagnoze:
        nozare = i["Specialisation"][0]["Name"]
        saraksts_arstejosa_nozare.append(nozare)

    diagnozes_dati = {  #Izveidota vārdnīca no kā tiks veidots dataframe
        "Atbilstiba(%)": saraksts_atbilstiba,
        "Arstejosa nozare": saraksts_arstejosa_nozare,
        "Apraksts": saraksts_apraksts,
        "Slimiba_id": saraksts_slimibas_id,
        "Slimiba": saraksts_slimibas
    }
    df_diagnoze = pd.DataFrame(diagnozes_dati, index = saraksts_vieta)
#Izveidota diagnozes tabula
    df_diagnoze = df_diagnoze.reindex(columns=['Slimiba', 'Slimiba_id',
'Apraksts', 'Atbilstiba(%)', 'Arstejosa nozare'])
    print(df_diagnoze)  #Lietotājam tiek parādīta diagnoze
    diagnoze = df_diagnoze
    return diagnoze


diagnoze = iegut_diagnozi(token, pacienta_simptomi, pacienta_dzimums,
dz_gads)  #Izmantota diagnozes funkcija
```