Web aplikacija za vođenje osobnog zdravstvenog dnevnika

Rastović, Milan

Sveučilište u Rijeci
Fakultet informatike
i digitalnih tehnologija

Sveučilišni prijediplomski studij Informatika

Milan Rastović

Web aplikacija za vođenje osobnog zdravstvenog dnevnika

Završni rad

Mentor: doc.dr.sc. Lucia Načinović Prskalo

Rijeka, lipanj 2024.

Sveučilište u Rijeci
Fakultet informatike i digitalnih tehnologija

Rijeka, 9. travnja 2024.

Zadatak za završni rad

Pristupnik/ica: Milan Rastović

Naziv završnog rada: Web aplikacija za vođenje osobnog zdravstvenog dnevnika

Naziv završnog rada na engleskom jeziku: Web application for managing personal health diary

Sadržaj zadatka: Zadatak završnog rada je izraditi web aplikaciju za vođenje osobnog zdravstvenog dnevnika. U radu će detaljno biti opisane sve tehnologije koje će se koristiti za izradu aplikacije, postupak izrade aplikacije te sve funkcionalnosti koje aplikacija pruža.

Mentorica
Doc. dr. sc. Lucia Načinović Prskalo

Načinović Prskalo

Voditelj za završne radove
Izv. prof. dr. sc. Miran Pobar

Zadatak preuzet: 9.4.2024.

(potpis pristupnika/ice)

Radmile Matejčić 2,
51000 Rijeka, Hrvatska

T: +385 (0)51/584-700
E: ured@inf.uniri.hr
www.inf.uniri.hr

OIB: 64218323816
IBAN: HR1524020061400006966

Sažetak

Tema ovoga rada je izrada i implementacija web aplikacije u obliku dnevnika za vođenje zdravlja. U web aplikaciji implementirane su iznimno korisne stavke za buduće korisnike. Korisniku je omogućeno da na vrlo jednostavan, intuitivan i zanimljiv način prati svoje zdravlje, poput unosa vrijednosti vitalnih znakova, kontaktiranja liječnika specijaliste, pregled statistike vrijednosti vitalnih znakova i slično. Za izradu web aplikacije korišteno je sveukupno znanje stečeno prilikom akademskog obrazovanja. U radu se koriste programski jezici JavaScript, HTML, CSS, Python i Django čime je osigurana funkcionalna platforma za korisnike. Web aplikacija je dizajnirana s naglaskom na korisničko iskustvo, pružajući jednostavan i intuitivan način za unos i pregled zdravstvenih podataka, kao i mogućnost komunikacije s medicinskim specijalistima.

Ključne riječi

Web aplikacija, dnevnik za vođenje zdravlja, baza podataka, korisnik, JavaScript, HTML, testiranje, CSS, Python, Django, API, RESTful, Postman, zdravstvena statistika, zdravstvena informatika, responzivni dizajn, frontend, backend, terapije, nalazi, vitalni znakovi, Figma

Sadržaj

1. Uvod

U suvremenom društvu, zdravlje je jedno od najvažnijih aspekata ljudskog života, a vođenje osobnog zdravstvenog dnevnika postaje sve značajnije za praćenje i održavanje zdravstvenog stanja. S razvojem tehnologije i povećanjem dostupnosti digitalnih alata, omogućeno je stvaranje aplikacija koje korisnicima olakšavaju upravljanje vlastitim zdravstvenim podacima. Ovaj rad detaljno prikazuje proces razvoja web aplikacije za vođenje osobnog zdravstvenog dnevnika koristeći HTML, CSS, JavaScript.

Cilj ovog rada jest demonstrirati kako se suvremene web tehnologije mogu upotrijebiti za izradu funkcionalne aplikacije koja odgovara potrebama korisnika u području zdravlja. Kroz konkretan primjer razvoja aplikacije, obuhvaćen je cjelokupan postupak, od dizajna u alatu Figma, do same implementacije ideje.

Osnovna namjena aplikacije je omogućiti korisnicima da jednostavno i pregledno upravljaju svojim zdravstvenim podacima. Aplikacija omogućuje unos i pregled medicinskih nalaza, praćenje vrijednosti vitalnih znakova, evidenciju lijekova i terapija, te bilježenje važnih zdravstvenih događaja. Također aplikacija nudi vizualizaciju zdravstvenih podataka kroz grafove i analize i kontaktiranje doktora.

Kroz detaljan opis razvoja i snimke ekrana prilikom postupka izrade, ovaj rad pruža cjelovitu sliku kako od ideje doći do realizacije funkcionalne web aplikacije. Na taj način, rad služi kao vodič i inspiracija za sve one koji žele istražiti mogućnosti web razvoja u svrhu poboljšanja osobnog zdravlja i kvalitete života. Naposljetku, prikazan je konačan izgled aplikacije, te su opisane njezine funkcionalnosti.

2. Pregled literature

2.1. Pregled postojećih rješenja za praćenje zdravstvene povijesti

U suvremenom digitalnom dobu, praćenje zdravstvene povijesti postalo je značajno lakše zahvaljujući razvoju različitih aplikacija i alata koji pomažu korisnicima u održavanju i upravljanju njihovim zdravstvenim podacima. Različita rješenja nude razne funkcionalnosti, a neka od najistaknutijih uključuju:

- *MyChart*: Ova aplikacija, razvijena od strane Epic Systems, omogućuje pacijentima pristup njihovim elektroničkim zdravstvenim zapisima, zakazivanje pregleda, dobivanje rezultata laboratorijskih testova, te komunikaciju s liječnicima. MyChart integrira podatke iz različitih bolničkih sustava, omogućujući cjelovit pregled zdravstvenog stanja pacijenata [1]
- *Apple Health*: Aplikacija koja dolazi s iOS uređajima, Apple Health integrira podatke iz raznih zdravstvenih i fitness aplikacija, kao i s medicinskih uređaja. Korisnicima omogućuje praćenje širokog spektra zdravstvenih podataka, uključujući fizičku aktivnost, prehranu, san i vitalne znakove [2].
- *Google Fit*: Googleova aplikacija fokusira se na praćenje fizičke aktivnosti i vitalnih znakova, kao što su otkucaji srca i koraci. Aplikacija se integrira s različitim uređajima i aplikacijama, pružajući korisnicima sveobuhvatan pregled njihovog zdravstvenog i fitness stanja [3].
- *Medisafe*: Specijalizirana aplikacija za praćenje uzimanja lijekova. Korisnicima omogućuje unos podataka o lijekovima, primanje podsjetnika za uzimanje doza, te praćenje njihovog napretka. Medisafe također pruža obavijesti o interakcijama lijekova i savjete za pravilno uzimanje terapije [4].
- *HealthVault:* Microsoftov alat za praćenje zdravstvenih podataka koji omogućuje korisnicima pohranu i upravljanje njihovim zdravstvenim informacijama, uključujući medicinske nalaze, imunizacije i povijest bolesti. HealthVault također omogućava dijeljenje podataka s liječnicima i zdravstvenim ustanovama [5].

Iako ova rješenja nude širok spektar funkcionalnosti, većina njih je usmjerena na specifične aspekte zdravlja ili integraciju s već postojećim zdravstvenim sustavima. Web aplikacija za vođenje osobnog zdravstvenog dnevnika nastoji kombinirati najbolje značajke ovih rješenja, pružajući sveobuhvatan alat za upravljanje osobnim zdravstvenim podacima. Fokus je na jednostavnosti korištenja i sveobuhvatnosti funkcionalnosti koje zadovoljavaju potrebe korisnika.

2.2. Tehnologije i alati korišteni za slične projekte

Razvoj web aplikacija za praćenje zdravstvene povijesti zahtijeva korištenje suvremenih tehnologija i alata koji omogućuju učinkovitu i sigurnu obradu podataka. Najčešće korištene tehnologije i alati u sličnim projektima uključuju:

- **JavaScript:** Popularan programski jezik za razvoj web aplikacija, JavaScript omogućuje dinamičko i interaktivno korisničko iskustvo. Upotreba okvira kao što su React.js, Angular i Vue.js dodatno olakšava razvoj složenih korisničkih sučelja, omogućujući brzu izradu responzivnih i skalabilnih aplikacija [6].
- **HTML i CSS:** Osnovne tehnologije za izradu i stiliziranje web stranica, HTML (HyperText Markup Language) i CSS (Cascading Style Sheets) omogućuju strukturiranje sadržaja i dizajn korisničkog sučelja. HTML se koristi za označavanje sadržaja, dok CSS osigurava vizualni izgled aplikacije [7] [8].
- **Figma:** Alat za dizajn korisničkih sučelja koji omogućuje jednostavnu vizualizaciju i prototipiranje aplikacija. Figma podržava suradnju u stvarnom vremenu, što olakšava timski rad na dizajnu i omogućuje brze iteracije prema povratnim informacijama korisnika [9].
- **Node.js:** Platforma za razvoj serverskih aplikacija koristeći JavaScript, Node.js omogućuje izradu skalabilnih i visokoučinkovitih serverskih aplikacija. Node.js se često koristi za backend razvoj, podržavajući upravljanje bazama podataka, autentifikaciju korisnika i obradu zahtjeva [10].
- **MongoDB:** NoSQL baza podataka koja je popularna zbog svoje fleksibilnosti i skalabilnosti. MongoDB omogućuje pohranu raznovrsnih podataka, uključujući korisničke profile, medicinske nalaze i druge zdravstvene informacije, te podržava dinamičko prilagođavanje strukture podataka [11].
- **RESTful API:** Arhitektonski stil za dizajn mrežnih aplikacija koji omogućuje komunikaciju između klijenta i servera. Korištenje RESTful API-ja omogućuje integraciju različitih komponenti aplikacije i razmjenu podataka, osiguravajući modularnost i skalabilnost sustava [12].
- **OAuth:** Standardni protokol za autorizaciju koji omogućuje sigurnu razmjenu podataka između aplikacija bez dijeljenja korisničkih vjerodajnica. OAuth se koristi za autentifikaciju korisnika i zaštitu njihovih podataka, osiguravajući da samo ovlašteni korisnici mogu pristupiti određenim informacijama i funkcionalnostima [13].

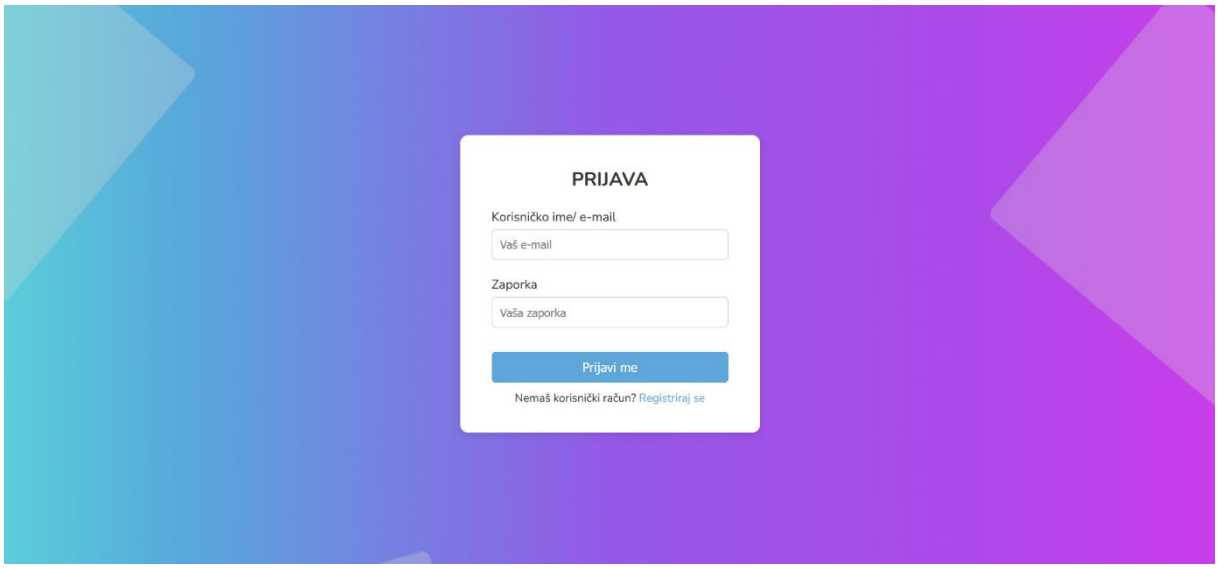
Integracija ovih tehnologija osigurava da aplikacija može zadovoljiti sve potrebne zahtjeve u pogledu performansi, sigurnosti i korisničkog iskustva.

3. Struktura i funkcionalnosti aplikacije

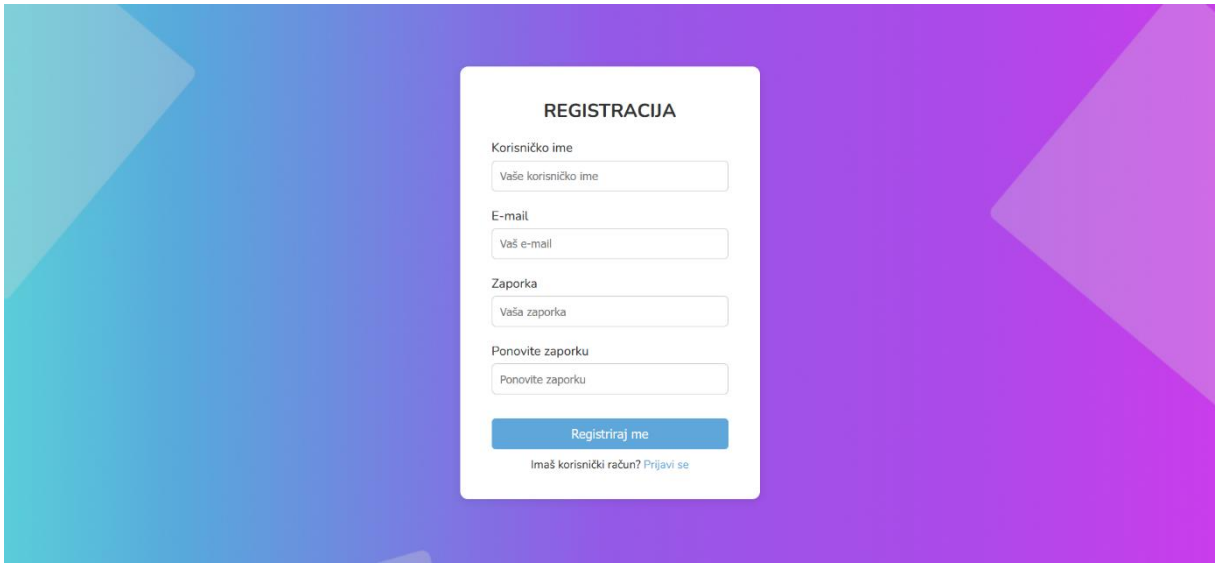
Web aplikacija za vođenje osobnog zdravstvenog dnevnika osmišljena je kao inovativno rješenje za korisnike, omogućavajući im unos, pregled i upravljanje pregledom vitalnih znakova, medicinskih pregleda, terapije koju koriste i nalaza.

Za izradu aplikaciju korištene su moderne tehnologije, garantirajući tako visoku funkcionalnost, sigurnost i kvalitetno korisničko iskustvo.

Da bi korisničko iskustvo bilo što kvalitetnije, web aplikacija ima vrlo intuitivnu i jednostavnu strukturu. Jednu od početnih komponenti sučelja čini stranica za prijavu (Slika 1) i registraciju korisnika (Slika 2), gdje korisnici unose svoje podatke poput email-a i zaporke da bi se uspješno ulogirali u aplikaciju.

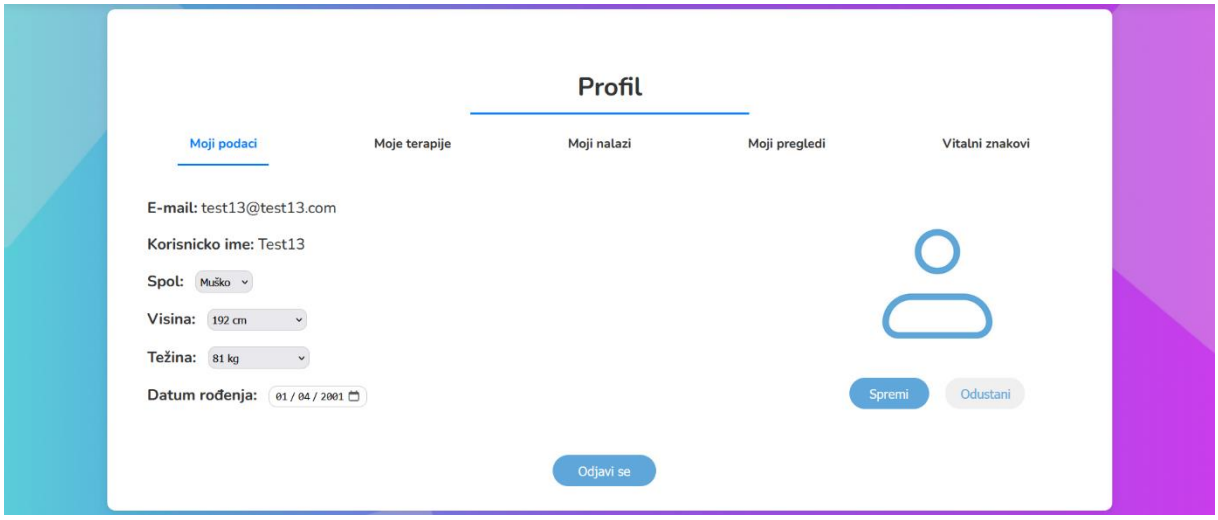


Slika 1 Prijava korisnika



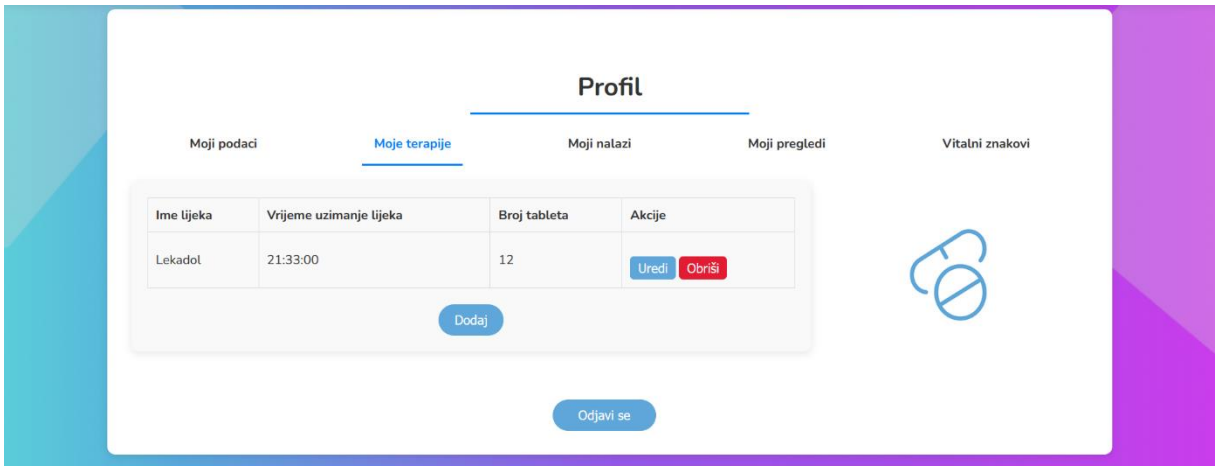
Slika 2 Registracija korisnika

Korisnici također imaju i mogućnost pregledavanja i uređivanja svojih profila unutar stranice „Profil“ (Slika 3), gdje se prikazuju informacije o samome korisniku – email, korisničko ime, spol, visina težina i datum rođenja.



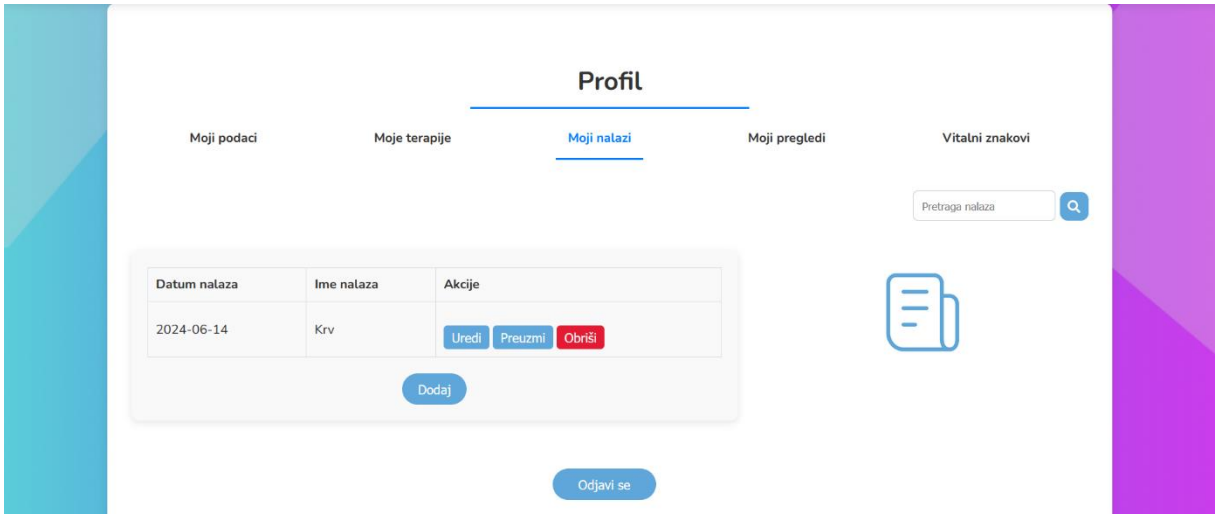
Slika 3 Profil korisnika

Stranica „Moje terapije“ (Slika 4) posebno je korisna za korisnike aplikacije obzirom na implementirano vrijeme za uzimanje lijekova. Na taj način, ovom stranicom je web aplikacija osigurala korisnicima pouzdano rješenje za uzimanje svojih lijekova na vrijeme.



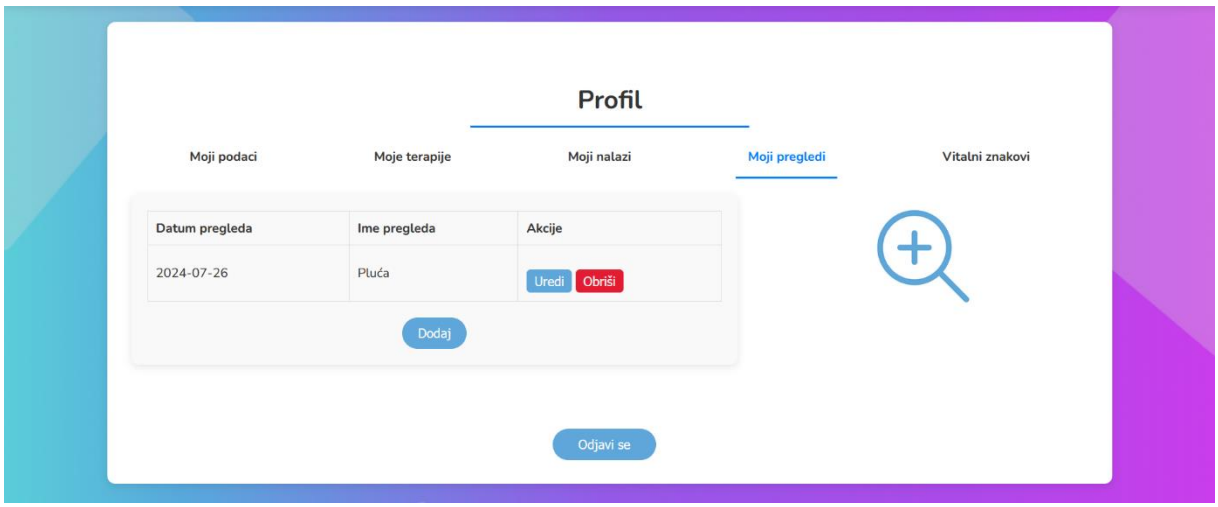
Slika 4 Moje terapije

„Moji nalazi“ stranica (Slika 5) također je dio aplikacije koji svojim sučeljem omogućava unos i pregled medicinskih nalaza na vrlo jednostavan način. Osim toga, ovaj dio korisničkog sučelja je izuzetno pregledan za korisnika i intuitivan. Korisnik na lak način može unijeti novi nalaz, zabilježiti datum izrade nalaza i dodati kratki opis nalaza. Također, korisniku je u bilo kojemu trenutku omogućeno preuzimanje bilo kojeg od unesenih nalaza.



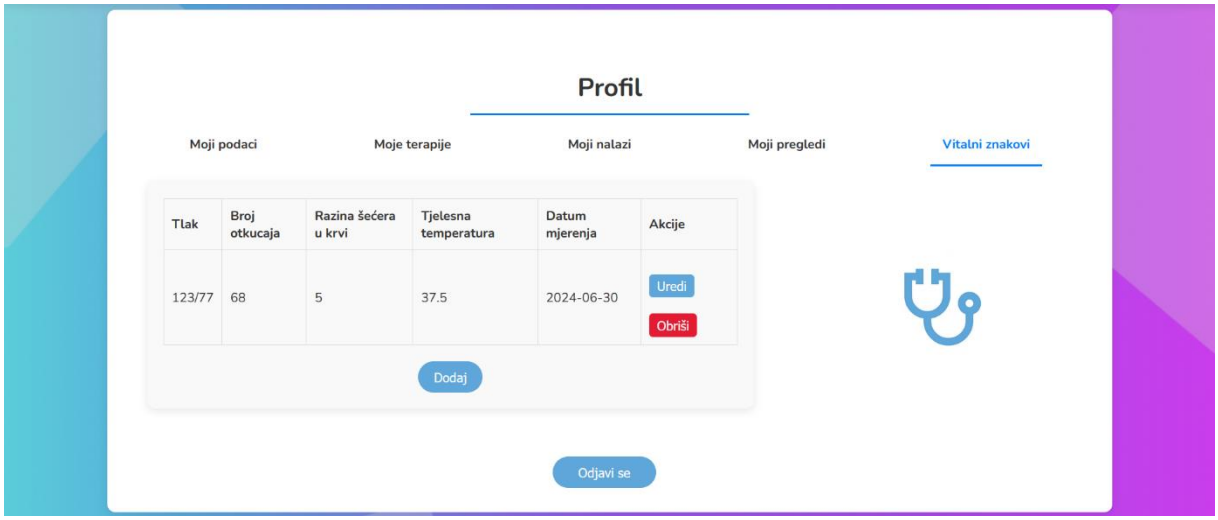
Slika 5 Moji nalazi

Stranicom „Moji pregledi“ (Slika 6) korisniku je omogućeno da na lak način vodi evidenciju obavljenih i nadolazećih pregleda. Stranica je strukturirana na način da korisnik unosi ime pregleda, datum pregleda i opis pregleda, te kada pregled unese, on se pojavljuje u strukturiranom poretku pregleda.



Slika 6 Moji pregledi

Vrijednosti vitalnih znakova poput krvnog tlaka, broja otkucaja srca, tjelesne temperature i razine šećera u krvi također se mogu unositi u ovoj web aplikaciji, a sve unutar stranice „Vitalni znakovi“ (Slika 7). Ovom stranicom omogućeno je da korisnik svakodnevno može voditi dnevnik nekih od vitalnih znakova, što je od izuzetne koristi za nadolazeće liječničke preglede u slučaju bolovanja od neke bolesti.



Slika 7 Vitalni znakovi

Nadovezujući se na stranicu „Vitalni znakovi“, stranica „Statistika“ (Slika 8) iz tablice parametara, koja prikazuje vrijednosti vitalnih znakova (Slika 9), računa prosjek svih vitalnih znakova unazad sedam dana i grafičkim prikazima sugerira na stanje istih putem tri moguća stanja smiješak, tužni i ozbiljni emoji. Korisnik također može provjeriti tablicu parametara koja je informativnog karaktera (Slika 9).

Slika 8 Statistika krvnog tlaka

Kategorija	Sistolni tlak (mmHg)	Dijastolni tlak (mmHg)
Niski (Hipotenzija)	<80	<50
Sniženi	80-89	50-59
Normalan	90-120	60-80
Povišeni	121-139	80-89
Visoki (Hipertenzija)	>140	>90

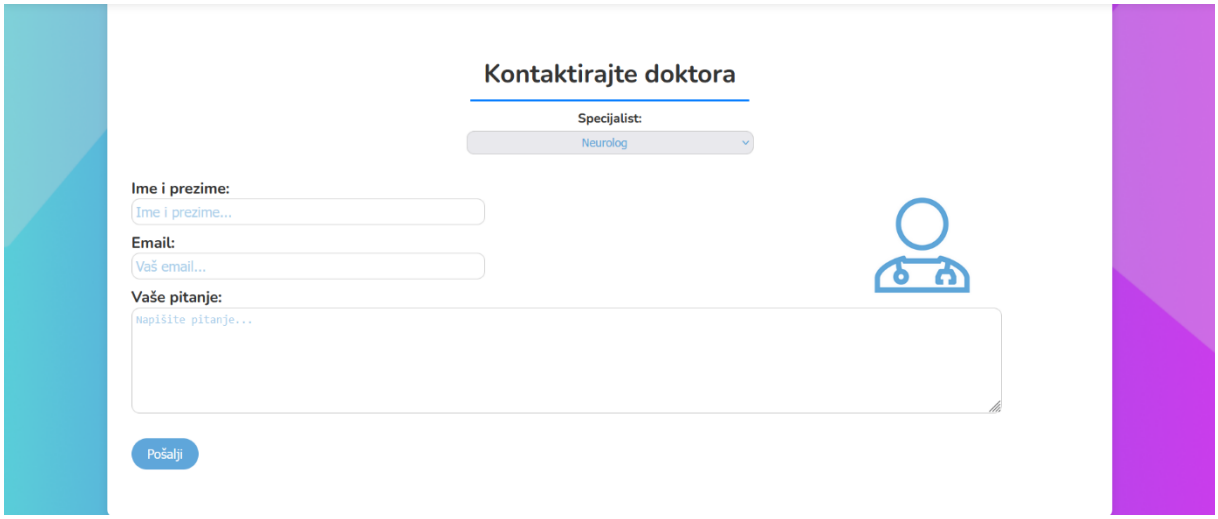
Kategorija	Razina šećera u krvi (mmol/L)
Niska (Hipoglikemija)	<3.9
Snižena	3.9-5.0
Normalna	5.0-7.8
Povišena (Predijabetes)	7.8-11.1
Visoka (Dijabetes)	>11.1

Kategorija	Broj otkucaja srca u minuti (BPM)
Niski (Bradikardija)	<60
Normalni	60-100
Povišeni (Tahikardija)	>100

Kategorija	Tjelesna temperatura (°C)
Hipotermija	<35.0
Normalna	35.0-37.5
Povišena (Subfebrilna)	37.5-38.0
Groznica (Febrilna)	>38.0
Visoka groznica	>39.0

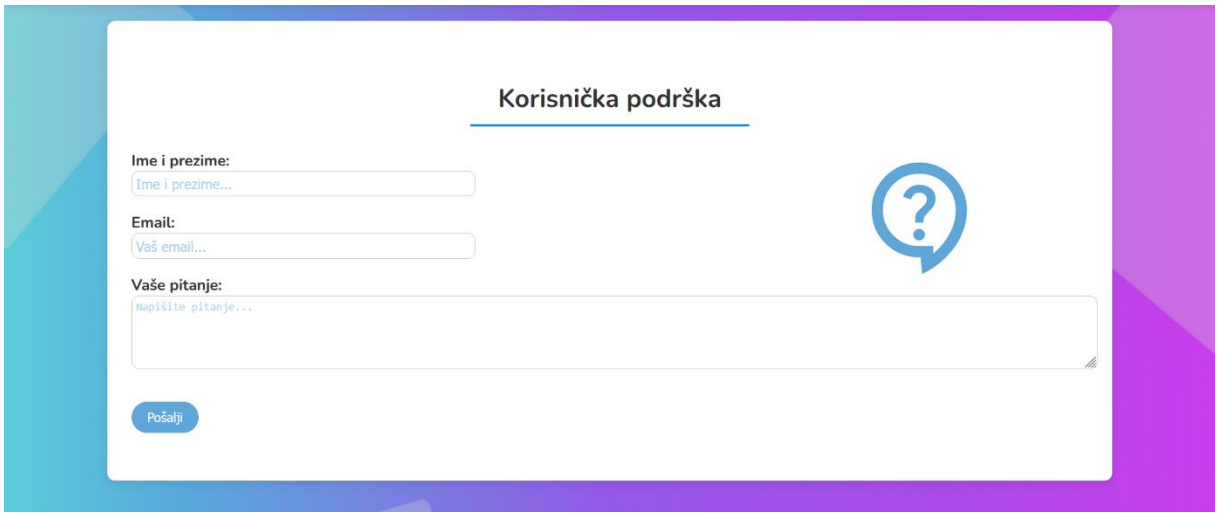
Slika 9 Tablica parametara

„Kontakt“ stranicom (Slika 10) korisniku je omogućeno da kontaktira svog liječnika specijalistu, a na način da iz padajućeg izbornika odabere specijalističko područje koje mu je potrebno, unese ime i prezime, email na koji će mu liječnik povratno odgovoriti i konkretni upit za liječnika. Klikom na gumb „Pošalji“, korisnikovo pitanje šalje se liječniku na mail.



Slika 10 Kontaktiranje doktora

Ova web aplikacija korisniku omogućuje i kontaktiranje korisničke podrške u slučaju bilo kakvih problema s aplikacijom. Stranica korisničke podrške (Slika 11) nije vidljiva na glavnoj stranici aplikacije, već se nalazi u podnožju iste i dostupna je za sve korisnike.



Slika 11 Kontaktiranje korisničke podrške

3.1. Dizajn aplikacije

Razvoj dizajna web aplikacije za vođenje osobnog zdravstvenog dnevnika je izrađen pomoću alata Figma. Figma je alat za dizajn koji pruža detaljnu i preciznu izradu dizajna korisničkog sučelja sa brojnim mogućnostima. Za logo aplikacije i ikone unutar aplikacije korišten je program Canva, jednostavan alat s mnogo predložaka i opcija za prilagođavanje.

Kao ključna karakteristika dizajna ističe se jednostavnost i intuitivnost. Prilikom same izrade sučelja, fokus je stavljen na stvaranje jasnih i lako razumljivih putanja do informacija za korisnike, čime je smanjena potreba za učenjem i usvajanjem aplikacije.

Obrasci za prijavu i registraciju korisnika dizajnirani su na način kako bi bili što jednostavniji, s minimalnim brojem koraka. Korisničkom profilu omogućeno je pregledavanje i uređivanje osobnih podataka, a za svaki unos podataka dan je jasan i precizan opis da ne bi došlo do pogrešaka prilikom upisivanja.

Vizualizacija podataka još je jedna od značajki dizajna ove web aplikacije. Grafički prikazi vrijednosti vitalnih znakova i terapija omogućuju korisnicima da lako prate promjene unutar svog zdravstvenog stanja kroz određeno vremensko razdoblje. Grafovima i statističkim prikazima također je osigurana dosljednost i lakoća pri interpretaciji podataka.

Boje koje su korištene prilikom izrade dizajna pažljivo su odabrane da bi estetski dojam bio što bolji, kao i korisničko iskustvo. Primarna boja ove web aplikacije je smirujuća nijansa plave boje koja je korištena u glavnim elementima sučelja – logo, gumbi, ikone i dijelovi teksta. Plava boja u aplikaciji asocira na povjerenje i profesionalnost, što je važno u kontekstu zdravstvenih aplikacija.

Sekundarne boje koje su korištene za dizajn ove web aplikacije su različite nijanse sive, ljubičaste, zelene i crvene boje koje osiguravaju bolju preglednost stavki unutar web aplikacije. Upotreba svijetlih i tamnih sivih nijansi koriste se za tekst kako bi osigurali lakše čitanje informacija i snalaženje u aplikaciji.

Dizajn u Figmi omogućio je detaljan i kolaborativan pristup, rezultirajući intuitivnim korisničkim sučeljem koje zadovoljava potrebe korisnika u kontekstu praćenja zdravstvenih parametara i komunikacije sa zdravstvenim stručnjacima.

4. Metodologija izrade i arhitektura aplikacije

4.1. Pristup izradi

Web aplikaciji za vođenje dnevnika zdravlja pristupilo se kombiniranjem linearnog i iterativnog procesa razvoja. Iterativni proces razvoja posebno se odnosi na dizajn korisničkog sučelja i cjelokupan proces integracije funkcionalnosti koje aplikacija nudi, uključujući i njihovo testiranje. Izrada stranica koje aplikacija nudi (Moj profil, Moji nalazi itd.) temeljena je na kombinaciji linearnog i iterativnog pristupa – svaka stranica razvijana je korak po korak i zasebno.

Nakon implementacije cjelokupnog „frontend-a“ aplikacije, uslijedila je implementacija „backend“ funkcionalnosti koje stranice aplikacije koriste. Na kraju, slijedi postupak testiranja aplikacije.

4.2. „Frontend“ tehnologije

„Frontend“ aplikacije za vođenje osobnog zdravstvenog dnevnika razvijen je koristeći HTML, CSS i JavaScript. Ovim tehnologijama omogućena je izrada dinamičkog i responzivnog korisničkog sučelja koje je jednostavno, intuitivno i estetski ugodno.

HTML (HyperText Markup Language) standardni je programski jezik korišten za izradu, oblikovanje i strukturiranje sadržaja na web stranicama aplikacije [7].

Stranice aplikacije su izrađene pomoću HTML programskog jezika. Upravo njime određena je osnovna struktura i hijerarhija elemenata na stranici, te je korisnicima omogućena interakcija s aplikacijom putem formi, tablica i grafova.

Ovim HTML kodom definirana je struktura početne stranice web aplikacije.

Prvotno, „DOCTYPE“ deklaracija označava da se radi o HTML5 dokumentu. Zatim se otvara HTML element s atributom „lang="hr"“, što postavlja jezik dokumenta na hrvatski.

U „head“ sekciji definirani su osnovni meta podaci i linkovi na vanjske resurse (Slika 12). Prvo se postavlja skup znakova na „UTF-8“ kako bi se podržali svi znakovi i simboli. Zatim, meta oznakom „viewport“ osigurano je da stranica bude ispravno prikazana.

Nadalje, definira se naslov stranice pomoći „title“ elementa. Kroz „link“ elemente, dodaju se vanjski CSS stilovi, uključujući stilove definirane u datoteci styles.css i font "Nunito" preuzet s Google Fonts. Također, postavljena je ikona stranice koja se prikazuje na kartici preglednika.

„Body“ sekcija sadrži glavni sadržaj aplikacije. Elementi stranice definirani su po klasama koje su uređene u styles.css datoteci, a na isti način su rađene sve ostale stranice. Pomoću toga se organizira raspored sadržaja na stranici. Unutar svakog „div“ kontejnera nalaze se elementi poput slika (<img>), dugmadi (<button>) i tekstova različite svrhe (<p>, <h1>, <h2>, <h3>) koji su dodatno stilizirani u styles.css datoteci.

```html
<!DOCTYPE html>
<html lang="hr">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Početna Stranica</title>
    <link rel="stylesheet" href="/static/css/styles.css">
    <link href="https://fonts.googleapis.com/css2?family=Nunito:wght@400;600;700&display=swap" rel="stylesheet">
    <link rel="icon" sizes="32x32" href="/static/images/LogoZavrsni.png" id="faviconTag" type="image/png">
</head>
<body>
    <div class="intro">
        <div class="intro-content">
            <img src="/static/images/LogoZavrsni.png" alt="Health Journal Icon" class="icon">
            <h1>Vaš Osobni Zdravstveni Dnevnik</h1>
            <p>Pratite sve aspekte svog zdravlja na jednom mjestu.</p>
            <button onclick="window.location.href='{% url 'prijava' %}'">Započni</button>
        </div>
    </div>
```

Slika 12 Dio HTML koda početne stranice web aplikacije

CSS (Cascading Style Sheets) korišten je za uređivanje HTML elemenata i osiguravanje estetski ugodnog vizualnog dizajna aplikacije (Slika 13). Dizajn aplikacije izrađen je na temelju definiranih predložaka u Figmi, s naglaskom na korištenje fonta „Nunito“ i palete boja koja uključuje nijanse plave, sive, bijele i crvene boje za dugmad i linkove. CSS-om je omogućen responzivni dizajn koji osigurava optimalno prikazivanje sadržaja na različitim uređajima, uključujući desktop računala, tablete i mobilne uređaje. Korištenjem CSS „media queries“ (Slika 14), „layout“ je prilagođen ovisno o veličini ekrana. „Media query“ koji omogućava prilagodbu dizajna stranice za različite veličine zaslona (Slika 14) (Slika 15), osigurava responzivnost i optimalno korisničko iskustvo na mobilnim uređajima i tabletima. Svaka klasa definirana je kako se treba ponašati na manjim uređajima [8].

```css
body {
    margin: 0;
    padding: 0;
    box-sizing: border-box;
    font-family: 'Nunito', sans-serif;
    background: url('../images/pozadina.png') no-repeat center center fixed;
    background-size: cover;
    color: #333;
}

.intro {
    height: 100vh;
    display: flex;
    align-items: center;
    justify-content: center;
    text-align: center;
    color: white;
}

.intro-content {
    max-width: 600px;
    padding: 20px;
    background: rgba(0, 0, 0, 0.3);
    border-radius: 20px;
}

.icon {
    width: 150px;
    height: 150px;
}
```

Slika 13 Dio koda style.css datoteke

Dio koji se odnosi na „body“ element postavlja osnovne stilove za cijelu stranicu. „Margin“ i „padding“ su postavljeni na nulu kako bi se uklonili svi vanjski i unutarnji razmaci. „box-sizing: border-box“ osigurava da „padding“ i „border“ ne povećavaju ukupne dimenzije elemenata. Pozadinska slika se postavlja pomoću URL-a do slike „pozadina.png“, koja je centrirana i fiksirana, te prekriva cijelu pozadinu. Boja teksta je postavljena na tamno sivu.

„intro“ klasa stilizira uvodni dio stranice tako da zauzima cijelu visinu preglednika (100vh), a sadržaj unutar tog dijela je centriran vertikalno i horizontalno pomoću „flexbox-a“. Tekst unutar „intro“ sekcije je centriran i obojan bijelo.

„intro-content“ klasa stilizira sadržaj unutar uvodnog dijela tako da ima maksimalnu širinu od 600px, unutarnji razmak od 20px, te pozadinu s poluprozirnim crnim slojem (rgba(0, 0, 0, 0.3)) kako bi se tekst istaknuo na pozadini. Rubovi sadržaja su zaobljeni s „border-radius“ od 20px.

„icon“ klasa stilizira ikonu unutar uvodnog dijela tako da ima širinu i visinu od 150px.

```
@media (max-width: 768px) {
    .add-therapy-content {
        flex-direction: column;
        align-items: center;
    }

    .add-therapy-info, .add-therapy-details {
        width: 90%;
    }

    .add-therapy-details {
        align-items: center;
        margin-top: 20px;
    }

    .therapy-picture img {
        width: 120px;
        height: 120px;
    }

    .therapy-actions {
        width: 100%;
        display: flex;
        justify-content: space-evenly;
    }

    .save-button, .cancel-button {
        width: 40%;
    }
}
```

Slika 14 Kod za osiguravanje responzivnog dizajna kod uređaja od 768 piksela

```
@media (max-width: 480px) {
    .add-therapy-info {
        width: 100%;
    }

    .therapy-picture img {
        width: 100px;
        height: 100px;
    }

    .therapy-actions {
        flex-direction: column;
        gap: 10px;
    }

    .save-button, .cancel-button {
        width: 80%;
    }
}
```

Slika 15 Kod za osiguravanje responzivnog dizajna kod uređaja od 480 piksela

Na slične načine su stilizirane sve ostale klase i „media query“ optimizacije kod ostalih stranica unutar web aplikacije.

JavaScript je korišten za dodavanje interaktivnosti i dinamike aplikaciji. Skripta omogućuje manipulaciju DOM-om (Document Object Model) i asinkronu komunikaciju s „backend“ serverom. Funkcionalnosti implementirane JavaScript-om uključuju validaciju formi, navigaciju i interakciju te komunikaciju s „backend-om“ putem AJAX poziva. Na primjer, stranice poput „Prijava“ i „Registracija“ koriste JavaScript za provjeru ispravnosti

unosa korisničkih podataka prije slanja na server, osiguravajući da su svi potrebni podaci ispravno uneseni [14].

Primjeri različitih stranica dodatno ilustriraju kako su ove tehnologije integrirane. Na primjer, stranica za prijavu sadrži formu za unos email-a i zaporke, stiliziranu pomoću CSS-a, dok JavaScript skripta osigurava validaciju podataka prije slanja. Stranica za registraciju omogućava kreiranje novog korisničkog računa, dok stranica za dodavanje terapije pruža intuitivno sučelje za unos informacija o terapiji, uključujući naziv lijeka, vrijeme uzimanja i broj tableta. Stranica za korisničku podršku omogućava korisnicima da postavljaju pitanja i dobiju odgovore putem email-a, a stranica za statistiku vizualizira zdravstvene podatke korisnika putem interaktivnih grafova. JavaScript osigurava da korisnici koji nemaju korisnički račun ili se nisu prijavili u aplikaciju, ne mogu pristupiti stranicama poput profila, statistike i kontakta specijaliste (Slika 16).

```
document.addEventListener('DOMContentLoaded', function () {
    const token = localStorage.getItem('token');
    if (!token) {
        window.location.href = '/prijava/';
    }
});
```

Slika 16 Kod za zabranu pristupa stranicama

Cijeli proces razvoja „frontend-a“ temelji se na dizajnu kreiranom u Figmi, osiguravajući konzistentan i estetski ugodan korisnički doživljaj. Ovaj pristup osigurava da je „frontend“ aplikacije vizualno usklađen, pružajući korisnicima intuitivno iskustvo korištenja aplikacije za vođenje osobnog zdravstvenog dnevnika.

4.3. „Backend“ tehnologije

„Backend“ aplikacije za vođenje osobnog zdravstvenog dnevnika razvijen je korištenjem Django „framework-a“ i PostgreSQL baze podataka.

Django je visokorazinski Python „web framework“ koji omogućava čist i brz razvoj s vrlo praktičnim dizajnom. Iznimno je koristan ukoliko se koristi za izradu složenih stranica, obzirom da donosi brojne ugrađene funkcionalnosti – autentifikacija korisnika, administracijski „interface“, rukovanje URL-ovima i slično [15].

Prvotno, postavljeno je radno okruženje za Django projekt. Instaliran je Django „framework“ zajedno s potrebnim paketima koristeći „pip“, Python-ov alat za upravljanje paketima.

Konfiguracija Django projekta započinje u datoteci settings.py, gdje su definirane osnovne postavke projekta, uključujući instalirane aplikacije (Slika 17), middleware komponente (Slika 17), putanje do statičkih i medijskih datoteka (Slika 18), te postavke baze podataka (Slika 19). U ovom slučaju, koristi se PostgreSQL baza podataka.

PostgreSQL moćan je „open-source“ sustav za upravljanje relacijskim bazama podataka. Izuzetno je stabilan, skalabilan i podržava napredne funkcionalnosti poput složenih upita, transakcija, indeksiranja i slično. Podržava sve standarde SQL jezika i podršku za JSON podatke pa je samim time fleksibilan i koristan za različite vrste aplikacija [16].

Detalji konfiguracije baze podataka uključuju naziv baze podataka, korisničko ime, lozinku, „host“ i „port“.

```
10  INSTALLED_APPS = [
11      'django.contrib.admin',
12      'django.contrib.auth',
13      'django.contrib.contenttypes',
14      'django.contrib.sessions',
15      'django.contrib.messages',
16      'django.contrib.staticfiles',
17      'myapp',
18      'rest_framework',
19      'django.contrib.sites',
20      'allauth',
21      'allauth.account',
22      'allauth.socialaccount',
23      'rest_framework.authtoken',
24      'dj_rest_auth',
25      'dj_rest_auth.registration',
26  ]
27
28  MIDDLEWARE = [
29      'myapp.middleware.DisableCSRFForAuth',
30      'django.middleware.security.SecurityMiddleware',
31      'django.contrib.sessions.middleware.SessionMiddleware',
32      'django.middleware.common.CommonMiddleware',
33      'django.middleware.csrf.CsrfViewMiddleware',
34      'django.contrib.auth.middleware.AuthenticationMiddleware',
35      'django.contrib.messages.middleware.MessageMiddleware',
36      'django.middleware.clickjacking.XFrameOptionsMiddleware',
37      'allauth.account.middleware.AccountMiddleware',
38  ]
```

Slika 17 Kod za instalirane aplikacije i middleware

```
92  STATIC_URL = '/static/'
93  STATICFILES_DIRS = [
94      Path('C:/Users/Korisnik/Desktop/Stranica/static'),
95  ]
```

Slika 18 Putanja do svih statičkih datoteka unutar stranice

```
WSGI_APPLICATION = 'myproject.wsgi.application'

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': 'myproject',
        'USER': 'myprojectuser',
        'PASSWORD': 'test12345',
        'HOST': 'localhost',
        'PORT': '5432',
    }
}
```

Slika 19 Podaci za bazu podataka u PostgreSQL

Nakon postavljanja radnog okruženja, započinje razvoj „backend“ komponenata. Modeli su definirani u datoteci models.py.

Svaki model predstavlja tablicu u bazi podataka (Slika 20). Na primjer, model „UserProfile“ povezan je s korisnikom i sadrži polja kao što su spol, visina, težina i datum rođenja koja daju informacije o korisniku koji se služi aplikacijom. Model „Therapy“ bilježi podatke o terapijama korisnika, uključujući naziv lijeka, vrijeme uzimanja, dozu i opis. Modelom „MedicalReport“ pohranjeni su medicinski nalazi korisnika, dok model „Exam“ bilježi podatke o pregledima. Model „VitalSign“ prati vitalne znakove korisnika, uključujući krvni tlak, broj otkucaja srca, razinu šećera u krvi i tjelesnu temperaturu.

```
class UserProfile(models.Model):
    user = models.OneToOneField(User, on_delete=models.CASCADE)
    gender = models.CharField(max_length=10, blank=True, null=True)
    height = models.IntegerField(blank=True, null=True)
    weight = models.IntegerField(blank=True, null=True)
    birthdate = models.DateField(blank=True, null=True)

    def __str__(self):
        return self.user.username


class Therapy(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    medicine_name = models.CharField(max_length=100)
    medicine_time = models.TimeField()
    medicine_dosage = models.IntegerField()
    medicine_description = models.TextField(blank=True, null=True)

    def __str__(self):
        return self.medicine_name


class MedicalReport(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    report_name = models.CharField(max_length=255)
    report_date = models.DateField()
    report_description = models.TextField()
    report_file = models.FileField(upload_to='reports/')

    def __str__(self):
        return self.report_name
```

Slika 20 Primjer modela za bazu podataka

Za svaki model kreirani su odgovarajući „serializer“ u datoteci serializers.py (Slika 21). Serijalizeri omogućavaju pretvorbu složenih podataka poput upita baze podataka u Python podatkovne tipove koji su zatim lako konvertibilni.

Na primjer, „UserProfileSerializer“ serijalizira podatke modela „UserProfile“, dok „TherapySerializer“, „MedicalReportSerializer“, „ExamSerializer“ i „VitalSignSerializer“ serijaliziraju podatke svojih odgovarajućih modela navedenih u kodu.

```python
class CustomLoginSerializer(LoginSerializer):
    username = None
    email = serializers.EmailField(required=True, allow_blank=False)


class UserProfileSerializer(serializers.ModelSerializer):
    class Meta:
        model = UserProfile
        fields = ['gender', 'height', 'weight', 'birthdate']


class TherapySerializer(serializers.ModelSerializer):
    class Meta:
        model = Therapy
        fields = ['id', 'medicine_name', 'medicine_time', 'medicine_dosage', 'medicine_description']

class MedicalReportSerializer(serializers.ModelSerializer):
    class Meta:
        model = MedicalReport
        fields = ['id', 'report_name', 'report_date', 'report_description', 'report_file']

class ExamSerializer(serializers.ModelSerializer):
    class Meta:
        model = Exam
        fields = ['id', 'exam_name', 'exam_date', 'exam_description']

class VitalSignSerializer(serializers.ModelSerializer):
    class Meta:
        model = VitalSign
        fields = ['id', 'blood_pressure', 'heart_rate', 'blood_sugar', 'body_temperature', 'measurement_date']
```

Slika 21 Serijalizeri za odgovarajuće modele

U datoteci views.py definirane su „view“ funkcije i klase koje odgovaraju na HTTP zahtjeve. Koriste se Django „REST framework“ generičke klase kao što su „ListCreateAPIView“ i „RetrieveUpdateDestroyAPIView“ za implementaciju osnovnih CRUD (Create, Read, Update i Delete) operacija.

Na primjer, klasom „TherapyListCreateView“ omogućeno je dohvaćanje i kreiranje terapija za prijavljenog korisnika (Slika 22), dok se „TherapySerializer“ koristi za pretvorbu podataka, te je pristup omogućen samo autentificiranim korisnicima putem „TokenAuthentication“. Metodom „get_queryset“ osigurano je dohvaćanje samo terapije trenutnog korisnika, dok „perform_create“ postavlja korisnika kao vlasnika terapije prije spremanja.

Klasa „TherapyDetail“ omogućava dohvaćanje, ažuriranje i brisanje pojedinačne terapije za prijavljenog korisnika (Slika 22), te također koristi „TherapySerializer“ i zahtijeva autentifikaciju putem tokena. Metoda „get_queryset“ filtrira terapije prema trenutnom korisniku, osiguravajući da korisnici mogu pristupiti samo svojim podacima. Slične klase su implementirane za medicinske nalaze, preglede i vitalne znakove.

```python
class TherapyListCreateView(generics.ListCreateAPIView):
    serializer_class = TherapySerializer
    permission_classes = [IsAuthenticated]
    authentication_classes = [TokenAuthentication]

    def get_queryset(self):
        return Therapy.objects.filter(user=self.request.user)

    def perform_create(self, serializer):
        serializer.save(user=self.request.user)



class TherapyDetail(generics.RetrieveUpdateDestroyAPIView):
    serializer_class = TherapySerializer
    permission_classes = [IsAuthenticated]
    authentication_classes = [TokenAuthentication]

    def get_queryset(self):
        return Therapy.objects.filter(user=self.request.user)
```

Slika 22 „View“ funkcija za terapije

Dodatno, implementirane su specifične „view“ funkcije za podršku korisnicima i komunikaciju sa specijalistima.

Funkcija „ContactSpecialistView“ omogućava korisnicima slanje email-a specijalistu (Slika 23), dok „ContactSupportView“ omogućava korisnicima slanje upita korisničkoj podršci (Slika 24). Obje funkcije koriste Django „send_mail“ funkciju za slanje email poruka.

```python
class ContactSpecialistView(APIView):
    def post(self, request, *args, **kwargs):
        specialist = request.data.get('specialist')
        subject = request.data.get('subject')
        message = request.data.get('message')
        sender_email = request.data.get('email')

        specialist_emails = {
            'neurolog': 'neurolog@gmail.com',
            'oftamolog': 'oftamolog@gmail.com',
            'kardiolog': 'kardiolog@gmail.com',
            'dermatolog': 'dermatolog@gmail.com'
        }
```

Slika 23 Funkcija za kontaktiranje specijalista

```python
class ContactSupportView(APIView):
    permission_classes = [IsAuthenticated]

    def post(self, request):
        data = request.data
        sender_name = data.get('name')
        sender_email = data.get('email')
        message = data.get('message')

        if sender_name and sender_email and message:
            send_mail(
                'Upit korisničke podrške',
                f"Email: {sender_email}\n\n{message}",
                sender_email,
                ['health.journal2000@gmail.com'],
                fail_silently=False,
            )
            return JsonResponse({'message': 'Email poslan.'}, status=200)
        return JsonResponse({'error': 'Nevažeći podaci.'}, status=400)
```

Slika 24 Funkcija za kontaktiranje korisničke podrške

Svi URL obrasci definirani su u datoteci urls.py. Ova datoteka sadrži putanje koje povezuju URL-ove s odgovarajućim „view“ funkcijama i klasama. Na primjer, putanja „path ('profile/', UserProfileDetail.as_view(), name='user-profile')“ povezuje URL „profile/“ s klasom „UserProfileDetail“, koja obrađuje zahtjeve za dohvat i ažuriranje korisničkog profila (Slika 25).

```
urlpatterns = [
    path('profile/', UserProfileDetail.as_view(), name='user-profile'),
    path('therapy/', TherapyListCreateView.as_view(), name='therapy-list-create'),
    path('therapy/<int:pk>/', TherapyDetail.as_view(), name='therapy-detail'),
    path('reports/', MedicalReportListCreateView.as_view(), name='report-list-create'),
    path('reports/<int:pk>/', MedicalReportDetailView.as_view(), name='report-detail'),
    path('reports/<int:pk>/download/', MedicalReportDownloadView.as_view(), name='report-download'),
    path('exams/', ExamListCreateView.as_view(), name='exam-list-create'),
    path('exams/<int:pk>/', ExamDetailView.as_view(), name='exam-detail'),
    path('vital_signs/', VitalSignListCreate.as_view(), name='vital_signs'),
    path('vital_signs/<int:pk>/', VitalSignRetrieveUpdateDestroy.as_view(), name='vital_sign_detail'),
    path('statistics/<str:measurement_type>/', StatisticsView.as_view(), name='statistics'),
    path('api/contact-specialist/', ContactSpecialistView.as_view(), name='contact_specialist'),
    path('api/contact-support/', ContactSupportView.as_view(), name='contact_support'),
]
```

Slika 25 Putanje za povezivanje stranica sa klasama

Integracija „backend-a“ s „frontend-om“ ostvarena je putem „REST API-ja“. „Frontend“ koristi AJAX pozive za slanje i dohvaćanje podataka s „backend“ servera. Na primjer, prilikom prijave korisnika, „frontend“ šalje „POST“ zahtjev koji obrađuje „backend“ koristeći validaciju korisničkih podataka i autentifikaciju korisnika, te vraća odgovor „frontend-u“ i na temelju tog odgovora se provjerava je li korisnik uspješno prijavljen ili nije (Slika 26).

```
function loginUser(email, password) {
    fetch('http://127.0.0.1:8000/api/auth/login/', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json',
        },
        body: JSON.stringify({
            email: email,
            password: password
        })
    })
    .then(response => {
        return response.text().then(text => {
            return { status: response.status, body: text }
        });
    })
    .then(result => {
        if (result.status === 200) {
            const data = JSON.parse(result.body);
            if (data.key) {
                localStorage.setItem('token', data.key);
                window.location.href = '/profil/';
            } else {
                alert('Prijava nije uspjela!');
            }
        } else {
            alert('Greška pri prijavi: ' + result.body);
        }
    })
    .catch(error => {
        alert('Greška: ' + error);
    });
}
```

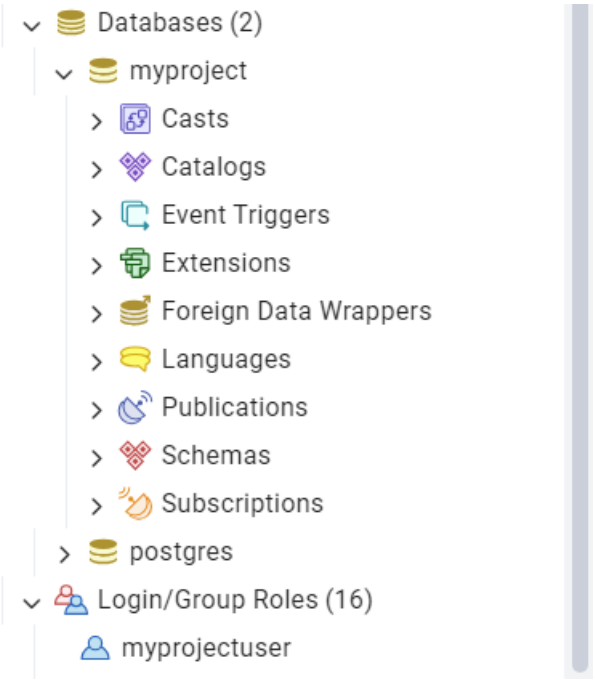
Slika 26 Slanje zahtjeva za prijavu korisnika i pohrana sigurnosnog tokena

4.4. Implementacija baze podataka

„Backend“ aplikacije za vođenje osobnog zdravstvenog dnevnika koristi PostgreSQL bazu podataka za pohranu i upravljanje podacima. PostgreSQL je odabrana zbog svoje robusnosti i podrške za složene upite.

4.4.1. Postgre SQL

Postavljanje PostgreSQL baze podataka započinje instalacijom PostgreSQL servera. Nakon instalacije, kreira se nova baza podataka i korisnik s odgovarajućim privilegijama. Konfiguracija baze podataka u Django projektu definirana je u datoteci settings.py gdje su navedeni svi potrebni parametri za povezivanje s bazom podataka (Slika 27Slika 1).



Slika 27 Baza podataka i korisnik u pgAdmin-u

Nakon postavljanja konfiguracije, kreiraju se migracije za modele definirane u datoteci models.py. Migracije omogućavaju da Django automatski kreira odgovarajuće tablice u bazi podataka na temelju definicija modela. Migracije se generiraju naredbom „python manage.py makemigrations“, a primjenjuju se naredbom „python manage.py migrate“. Za svaki model što se dodatno uredi, doda ili obriše potrebno je prije pokretanja servera napraviti migracije kako bi se te iste promijene spremile u bazi podataka.

Modeli podataka u Django projektu definirani su u datoteci models.py. Svaki model predstavlja tablicu u PostgreSQL bazi podataka. Na primjer, model „UserProfile“ povezan je s korisnikom i sadrži polja kao što su spol, visina, težina i datum rođenja. Model „Therapy“ bilježi podatke o terapijama korisnika, uključujući naziv lijeka, vrijeme uzimanja, dozu i opis, model „MedicalReport“ pohranjuje medicinske izvještaje korisnika, model „Exam“ bilježi podatke o pregledima, model „VitalSign“ pohranjuje vitalne znakove korisnika, uključujući krvni tlak, broj otkucaja srca, razinu šećera u krvi i tjelesnu temperaturu.

Ovi modeli definiraju strukturu podataka u bazi podataka i omogućavaju aplikaciji upravljanje korisničkim podacima i prikazivanjem istih.

5. Implementacija i razvoj aplikacije po funkcionalnostima

5.1. „Splash“ stranica

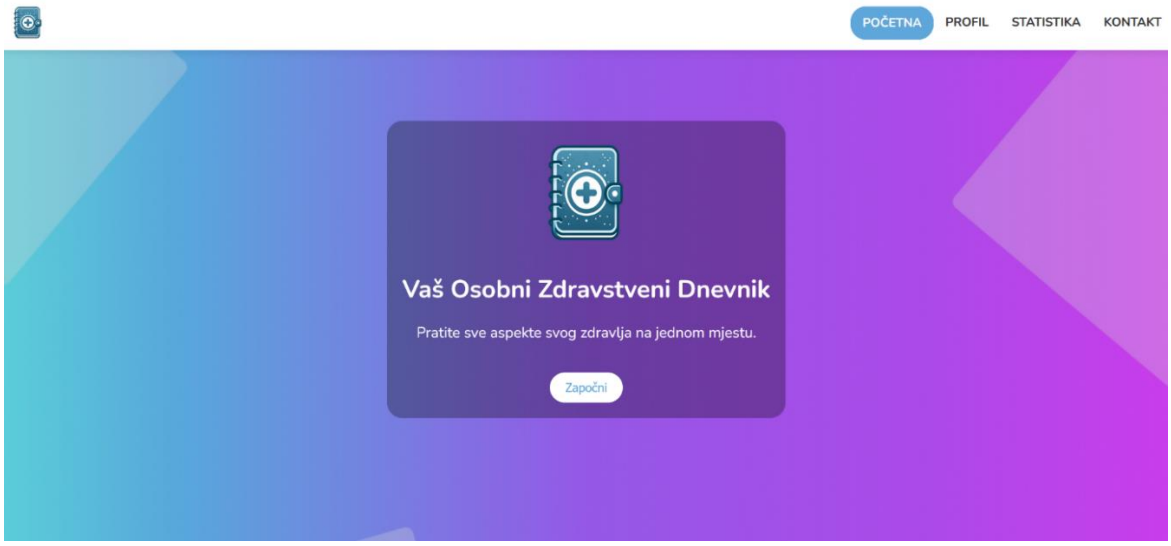
„Splash“ stranica (početna stranica) aplikacije za vođenje osobnog zdravstvenog dnevnika predstavlja korisnicima kratak uvid u aplikaciju i njene osnovne značajke. Dizajnirana je s ciljem da privuče pažnju korisnika i zainteresira ga za korištenje aplikacije.

Na samom vrhu stranice nalazi se dio koji uključuje logotip aplikacije, naslov "Vaš Osobni Zdravstveni Dnevnik", kratak opis funkcionalnosti aplikacije te gumb za prijavu (Slika 28). Ovim dijelom aplikacije, korisnicima je omogućen brz i jednostavan pristup osnovnim informacijama o aplikaciji, kao i mogućnost da se odmah prijave ili registriraju ako nemaju napravljen korisnički račun te zatim započnu s korištenjem.

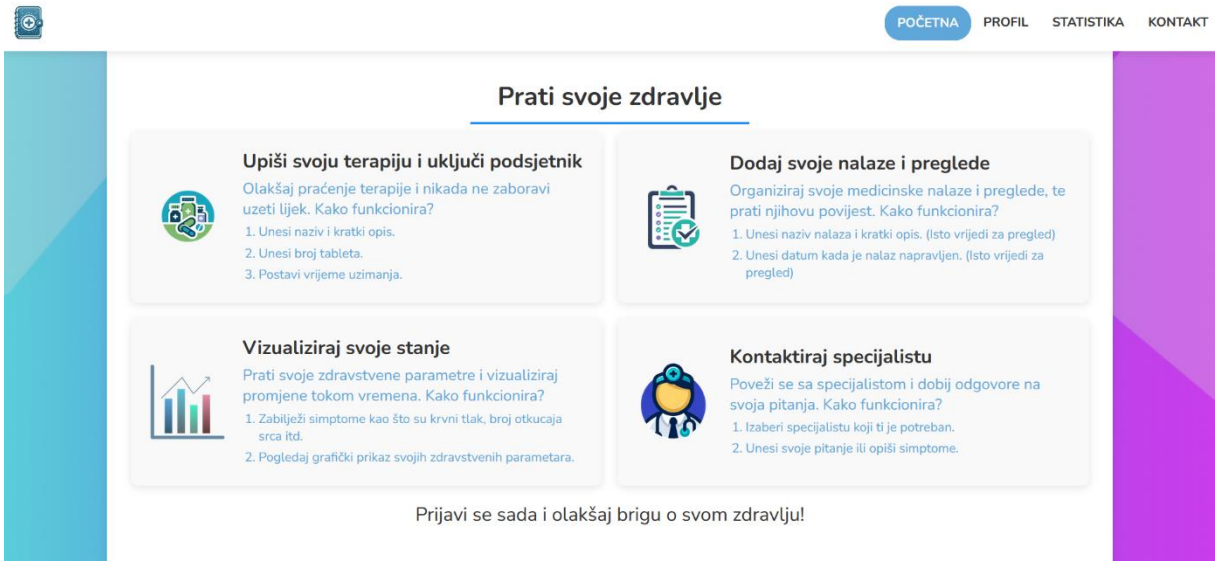
Navigacijska traka koja se nalazi ispod uvodnog dijela, omogućava jednostavnu navigaciju između različitih dijelova aplikacije. Traka sadrži glavne linkove kao što su „Početna“, „Profil“, „Statistika“ i „Kontakt“, čime korisnicima omogućava brz pristup ključnim funkcionalnostima aplikacije (Slika 28). Također, navigacijska traka uključuje i hamburger meni koji omogućava preglednu navigaciju na mobilnim uređajima.

Središnji dio stranice posvećen je predstavljanju četiri glavne funkcionalnosti aplikacije (Slika 29).

U podnožju stranice nalaze se linkovi za brzi pristup stranicama statistika, kontakt specijaliste, profilnim sekcijama i korisničkoj podršci (Slika 30).



Slika 28 Početna stranica



Slika 29 Opis funkcionalnosti web aplikacije

Slika 30 „Podnožje“

5.2. Stranica za registraciju i prijavu korisnika

Stranica za prijavu korisnika unutar aplikacije omogućava korisnicima da se prijave i pristupe svojim osobnim zdravstvenim podacima, a ukoliko nemaju svoj korisnički račun mogu ga napraviti preko stranice za registraciju.

Centralni dio stranice zauzima forma za prijavu. Forma sadrži polje za unos email adrese i zaporke, te gumb za prijavu. Unosom svojih podataka u ova polja, korisnici mogu jednostavno i brzo pristupiti svom profilu. Polja su označena odgovarajućim oznakama, što korisnicima jasno daje do znanja koji podaci su potrebni za prijavu (Slika 31).

```
<body>
    <div class="login">
        <div class="login-container">
            <h2>PRIJAVA</h2>
            <form id="login-form" method="POST">
                <label for="email">Korisničko ime/ e-mail</label>
                <input type="email" id="email" name="email" placeholder="Vaš e-mail" required>
                <label for="password">Zaporka</label>
                <input type="password" id="password" name="password" placeholder="Vaša zaporka" required>
                <button type="submit" class="login-button">Prijavi me</button>
            </form>
            <p class="register-link">Nemaš korisnički račun? <a href="{% url 'registracija' %}">Registriraj se</a></p>
        </div>
    </div>

    <script src="/static/js/auth.js"></script>
</body>
```

Slika 31 Html kod za formu prijavljivanja korisnika

Kada se forma ispuni i stisne se dugme za prijavu, podaci se šalju na „backend“ server putem JavaScript funkcije koja se koristi „fetch API-jem“. Server provjera i validira podatke koji su uneseni, koristeći se prilagođenim serijalizatorom. Ukoliko su podaci ispravni, serijalizer vraća autentifikacijski token koji se pohranjuje u „localStorage“, omogućavajući na taj način korisniku da pristupi zaštićenim dijelovima aplikacije.

Ispod forme za prijavu korisnika u aplikaciju, nalazi se poveznica za registraciju, koji upućuje nove korisnike na stranicu za kreiranje novog korisničkog računa. Proces registracije unutar aplikacije za vođenje osobnog zdravstvenog dnevnika omogućava novim korisnicima da kreiraju račune kako bi mogli pristupiti svojim osobnim zdravstvenim podacima i funkcionalnostima aplikacije.

Forma za registraciju slična je formi za prijavu korisnika. Prvi korak u procesu registracije je unos podataka od strane korisnika. Korisnik unese svoje korisničko ime, email adresu, zaporku i potvrdu zaporke u formu za registraciju na stranici (Slika 32). Sva polja je obavezno ispuniti kako bi se osigurali svi potrebni podaci za kreiranje računa.

```
<body>
    <div class="registration">
        <div class="registration-container">
            <h2>REGISTRACIJA</h2>
            <form id="registration-form" method="POST" action="">
                <label for="username">Korisničko ime</label>
                <input type="text" id="username" name="username" placeholder="Vaše korisničko ime" required>
                <label for="email">E-mail</label>
                <input type="email" id="email" name="email" placeholder="Vaš e-mail" required>
                <label for="password1">Zaporka</label>
                <input type="password" id="password1" name="password1" placeholder="Vaša zaporka" required>
                <label for="password2">Ponovite zaporku</label>
                <input type="password" id="password2" name="password2" placeholder="Ponovite zaporku" required>
                <button type="submit" class="registration-button">Registriraj me</button>
            </form>
            <p class="login-link">Imaš korisnički račun? <a href="{% url 'prijava' %}">Prijavi se</a></p>
        </div>
    </div>
    <script src="/static/js/auth.js" defer></script>
</body>
```

Slika 32 Html kod za formu registracije korisnika

Kada korisnik ispuni formu i stisne gumb za registraciju, podaci se šalju na „backend“ server. Na „frontend“ strani, JavaScript funkcija koristi se također „fetch API-jem“ za slanje „POST“ zahtjeva na „backend endpoint“ koji je zadužen za registraciju. „POST“ zahtjev sadrži korisničke podatke u JSON formatu (Slika 33). Takvim pristupom omogućena je sigurna komunikacija između klijentske i serverske strane, osiguravajući tako da su svi podaci ispravno poslani i primljeni.

```
function registerUser(username, email, password1, password2) {
    fetch('http://127.0.0.1:8000/api/auth/registration/', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json',
        },
        body: JSON.stringify({
            username: username,
            email: email,
            password1: password1,
            password2: password2
        })
    })
    .then(response => {
        if (response.status === 204) {
            alert('Registracija uspješna! Preusmjeravam na profil.');
            window.location.href = '/profil/';
            return;
        }
        return response.text();
    })
    .then(data => {
        if (data) {
            try {
                const jsonData = JSON.parse(data);
                if (jsonData.key) {
                    localStorage.setItem('token', jsonData.key);
                    window.location.href = '/profil/';
                } else {
                    alert('Registracija nije uspjela! ' + JSON.stringify(jsonData));
                }
            } catch (e) {
                alert('Greška pri parsiranju odgovora: ' + e.message + '\nOdgovor: ' + data);
            }
        }
    })
```

Slika 33 Slanje zahtjeva za registraciju korisnika

Na „backend“ strani, Django „REST Auth“ korišten je za obradu zahtjeva za registraciju. Paket automatski pruža „view“ funkciju koja provjera i validira podatke koje je korisnik unio prilikom registracije. Validacija uključuje provjeru da su unesene zaporke jednake i da se email adresa ne nalazi već u bazi podataka, odnosno da nije u upotrebi. Ako su svi podaci provjereni i prihvaćeni, kreira se novi korisnički račun. Django „REST Auth“ olakšava ovaj proces pružajući gotove alate za validaciju i kreiranje korisničkih računa, što ubrzava razvoj i povećava sigurnost aplikacije.

5.3. Profil korisnika

Stranica za profil korisnika jest glavna stranica aplikacije koja omogućava pregled i upravljanje svim osobnim zdravstvenim podacima korisnika koje on unese. Stranica je dizajnirana i osmišljena kako bi korisnicima pružila iskustvo lakog upravljanja vlastitim podacima, a da je pritom vrlo jednostavna, pregledna i intuitivna.

Navigacijska traka na vrhu korisnicima omogućava jednostavnu navigaciju kao i kod početne stranice.

Centralni dio stranice posvećen je korisničkim podacima i podijeljen je u nekoliko sekcija, od kojih svaka pruža nekoliko aspekata zdravstvenih informacija korisnika. Prva sekcija, "Moji podaci", omogućava korisnicima pregled i uređivanje osnovnih podataka kao što su spol, visina, težina i datum rođenja. Korisnici mogu ažurirati ove informacije, a sve promjene mogu spremiti klikom na gumb "Spremi" ili ih poništiti klikom na gumb "Odustani" (Slika 3).

5.3.1. Sekcije

Kroz intuitivno sučelje aplikacije, korisnici mogu pregledavati, dodavati, ažurirati i brisati podatke vezane za osobne terapije, medicinske nalaze, preglede i vitalne znakove. Svaka

stranica unutar aplikacije slijedi sličan obrazac interakcije između korisnika i sustava, što uključuje:

1. Dohvaćanje podataka:

Kada korisnik pristupi određenoj sekciji aplikacije, aplikacija šalje „GET“ zahtjev na „backend“ server da bi dohvatila sve relevantne podatke povezane s korisničkim računom koji trenutno želi pristupiti aplikaciji. „Frontend“ strana sa JavaScript funkcijom koristi „fetch“ za slanje „GET“ zahtjeva na odgovarajući „API endpoint“, npr. „/api/user/reports/“ za nalaze. „Backend“ se koristi Django „REST framework-om“ i odgovarajućim „view“ klasama kao što su „ListCreateAPIView“ ili specijalizirane klase poput „MedicalReportListCreateView“. Tim klasama podaci su filtrirani na temelju prijavljenog korisnika koristeći se sa „self.request.user“. Odgovarajući podaci zatim se vraćaju na „frontend“ stranu u JSON formatu. Nakon što su podaci zaprimljeni, „frontend“ ih prikazuje u tabličnom ili drugom odgovarajućem obliku kao za sekciju moji podaci, omogućujući korisniku jednostavan pregled i interakciju s podacima.

2. Spremanje novih podataka:

Korisnici mogu dodavati nove zapise unutar aplikacije klikom na gumb "Dodaj", koji ih zatim preusmjerava na stranicu za unos novih podataka. Na stranici za unos novih podataka, korisnici unose nove željene informacije u formu. Nakon što su uneseni svi podaci tj. informacije, „frontend“ JavaScript funkcijom šalje „POST“ zahtjev na odgovarajući „API endpoint“, npr. „/api/user/reports/“ za nalaze. Zahtjev uključuje JSON objekt s podacima koje je korisnik unio. „Backend“ ima ulogu obrade tih zahtjeva koristeći se metodom „perform_create“ u „view“ klasama, tako da automatski povezuje nove zapise s trenutnim korisničkim računom. „Backend“ također i validira podatke koristeći serijalizer poput „MedicalReportSerializer“. Ukoliko je proces validacije uspješan, novi zapis se pohranjuje i odgovarajući odgovor se vraća „frontend-u“.

3. Ažuriranje i brisanje postojećih podataka:

Svaki prikazani zapis ima opciju za uređivanje i brisanje unesenih podataka. Klikom na gumb za uređivanje, korisnici mogu izmijeniti postojeće podatke. Unesene promjene od strane korisnika putem „PUT“ zahtjeva šalju se na „backend“ na odgovarajući „API endpoint“, npr. „/api/user/reports/$<reportId>/“, gdje „<reportId>“ predstavlja jedinstveni identifikator zapisa. Gledajući „frontend“ stranu, JavaScript funkcija serijalizira ažurirane podatke u JSON formatu i šalje ih na server. „Backend“ koristi metodu „perform_update“ za spremanje promjena koje su unesene. Upravo time ostvarena je validacija podataka koristeći serijalizer prije ažuriranja zapisa u bazi podataka. Funkcija za brisanje radi na jednak način samo što se koristi metodom „perform_destroy“ kako bi zapis bio uklonjen iz baze podataka u kojoj se nalazio. Nakon uspješnog ažuriranja ili brisanja, „backend“ vraća ažurirane podatke „frontend-u“, koji zatim osvježava prikaz za korisnika sa najnovijim unesenim informacijama.

4. Preuzimanje:

Korisnici također mogu i preuzeti medicinske nalaze koje su pohranili u aplikaciju. Klikom na gumb za preuzimanje dokumenta, pokreće se „GET“ zahtjev na odgovarajući „API endpoint“ „/api/user/reports/$<reportId>/download/“. „GET“ zahtjev vraća datoteku koja se može preuzeti i pohraniti lokalno na uređaju kojeg korisnik trenutno koristi. „Backend“ koristi „FileResponse“ iz Django HTTP modula za vraćanje datoteke korisniku te osigurava da se ta ista datoteka isporuči za korisnika u ispravnome formatu. „Frontend“ JavaScript funkcijom automatski pokreće preuzimanje datoteke željene od strane korisnika.

5.3.1.1. Moji podaci

"Moji podaci", omogućava korisnicima pregled i uređivanje osnovnih podataka kao što su spol, visina, težina i datum rođenja (Slika 34). Korisnici mogu ažurirati ove informacije, a sve promjene moguće je spremiti klikom na gumb "Spremi" (Slika 35) ili ih poništiti klikom na gumb "Odustani". Ova sekcija koristi model „UserProfile“ za pohranu podataka (Slika 37), a serijalizer „UserProfileSerializer“ osigurava da je validacija uspješna (Slika 38).

```
41 ▼            <div id="mydata" class="profile-section active">
42 ▼                <div class="profile-content">
43 ▼                    <div class="profile-info">
44                          <p><strong>E-mail:</strong> <span id="email"></span></p>
45                          <p><strong>Korisnicko ime:</strong> <span id="name"></span></p>
46 ▼                        <p><strong>Spol:</strong>
47 ▼                            <select id="gender">
48                                  <option value="male">Muško</option>
49                                  <option value="female">Žensko</option>
50                              </select>
51                          </p>
52 ▼                        <p><strong>Visina:</strong>
53 ▼                            <select id="height">
54                                  <option value="">Odaberite visinu</option>
55                              </select>
56                          </p>
57 ▼                        <p><strong>Težina:</strong>
58 ▼                            <select id="weight">
59                                  <option value="">Odaberite težinu</option>
60                              </select>
61                          </p>
62 ▼                        <p><strong>Datum rođenja:</strong>
63                              <input type="date" id="birthdate">
64                          </p>
65
66                      </div>
```

Slika 34 HTML kod za prikaz osnovnih podataka o korisniku

Ako se korisnik odluči za odbacivanje promjena koje je unio, klikom na gumb "Odustani", „frontend“ JavaScript funkcija restartira formu na originalne podatke koje je dohvatila prilikom inicijalnog učitavanja stranice (Slika 36).

```
    document.getElementById('save-button').addEventListener('click', function() {
        const profileData = {
            gender: document.getElementById('gender').value,
            height: document.getElementById('height').value,
            weight: document.getElementById('weight').value,
            birthdate: document.getElementById('birthdate').value
        };

        fetch('/api/user/profile/', {
            method: 'PUT',
            headers: {
                'Authorization': `Token ${token}`,
                'Content-Type': 'application/json',
                'X-CSRFToken': csrftoken
            },
            body: JSON.stringify(profileData)
        })
        .then(response => {
            if (response.ok) {
                alert('Promjene su spremljene.');
            } else {
                alert('Greška prilikom spremanja promjena.');
            }
        })
        .catch(error => console.error('Error:', error));
    });
```

Slika 35 Spremanje podataka spola, visine, težine i datum rođenja korisnika

```
    document.getElementById('cancel-button').addEventListener('click', function() {
        alert('Promjene su odbačene.');
        fetch('/api/user/profile/', {
            method: 'GET',
            headers: {
                'Authorization': `Token ${token}`,
                'Content-Type': 'application/json'
            }
        })
        .then(response => response.json())
        .then(data => {
            document.getElementById('gender').value = data.gender || '';
            document.getElementById('height').value = data.height || '';
            document.getElementById('weight').value = data.weight || '';
            document.getElementById('birthdate').value = data.birthdate || '';
        })
        .catch(error => console.error('Error:', error));
    });
});
```

Slika 36 Odbacivanje promjena i dohvaćanje starih vrijednosti

```
class UserProfile(models.Model):
    user = models.OneToOneField(User, on_delete=models.CASCADE)
    gender = models.CharField(max_length=10, blank=True, null=True)
    height = models.IntegerField(blank=True, null=True)
    weight = models.IntegerField(blank=True, null=True)
    birthdate = models.DateField(blank=True, null=True)

    def __str__(self):
        return self.user.username
```

Slika 37 Model za "moji podaci"

```python
class UserProfileSerializer(serializers.ModelSerializer):
    class Meta:
        model = UserProfile
        fields = ['gender', 'height', 'weight', 'birthdate']
```

Slika 38 Serializer za "moji podaci"

5.3.1.2. Moje terapije

Sekcija "Moje terapije" (Slika 39) omogućava korisnicima da pregledavaju (Slika 40), dodaju (Slika 41 Dodavanje nove terapije), ažuriraju i upravljaju svojim terapijama (Slika 42). Model „Therapy“ definira strukturu terapija (Slika 43), dok „TherapySerializer“ osigurava pravilnu validaciju (Slika 44).

```html
<div id="therapies" class="profile-section">
    <div class="therapies-content">
        <div class="therapies-container">
            <table class="therapies-table">
                <thead>
                    <tr>
                        <th>Ime lijeka</th>
                        <th>Vrijeme uzimanje lijeka</th>
                        <th>Broj tableta</th>
                        <th>Akcije</th>
                    </tr>
                </thead>
                <tbody>

                </tbody>
            </table>
            <button class="add-button" onclick="window.location.href='{% url 'terapija' %}'">Dodaj</button>
        </div>
        <div class="therapy-image">
            <img src="/static/images/tablete.png" alt="Tablete">
        </div>
    </div>
</div>
```

Slika 39 HTML kod za prikazivanje terapija

```javascript
fetch('/api/user/therapy/', {
    method: 'GET',
    headers: {
        'Authorization': `Token ${token}`,
        'Content-Type': 'application/json'
    }
})
.then(response => response.json())
.then(data => {
    const tbody = document.querySelector('.therapies-table tbody');
    tbody.innerHTML = '';
    data.forEach(therapy => {
        const tr = document.createElement('tr');
        tr.innerHTML = `
            <td>${therapy.medicine_name}</td>
            <td>${therapy.medicine_time}</td>
            <td>${therapy.medicine_dosage}</td>
            <td>
                <button class="edit-button" data-id="${therapy.id}">Uredi</button>
                <button class="delete-button" data-id="${therapy.id}">Obriši</button>
            </td>
        `;
        tbody.appendChild(tr);
    });
})
.catch(error => console.error('Error:', error));
```

Slika 40 Dohvaćanje terapija iz baze podataka

```
        const url = therapyId ? `/api/user/therapy/${therapyId}/` : '/api/user/therapy/';

        fetch(url, {
            method: method,
            headers: {
                'Authorization': `Token ${token}`,
                'Content-Type': 'application/json',
                'X-CSRFToken': document.querySelector('meta[name="csrf-token"]').getAttribute('content')
            },
            body: JSON.stringify({
                medicine_name: medicineName,
                medicine_time: medicineTime,
                medicine_dosage: medicineDosage,
                medicine_description: medicineDescription
            })
        })
        .then(response => {
            if (response.ok) {
                alert('Terapija je uspješno spremljena.');
                localStorage.removeItem('therapyId');
                window.location.href = '/profil/#therapies';
            } else {
                response.json().then(data => {
                    alert('Greška prilikom spremanja terapije: ' + data.detail);
                });
            }
        })
        .catch(error => console.error('Error:', error));
    });

    document.getElementById('cancel-button').addEventListener('click', function () {
        localStorage.removeItem('therapyId');
        window.location.href = '/profil/#therapies';
    });
});
```

Slika 41 Dodavanje nove terapije

```
    document.querySelector('.therapies-table tbody').addEventListener('click', function(event) {
        if (event.target.classList.contains('delete-button')) {
            const therapyId = event.target.getAttribute('data-id');
            fetch(`/api/user/therapy/${therapyId}/`, {
                method: 'DELETE',
                headers: {
                    'Authorization': `Token ${token}`,
                    'Content-Type': 'application/json',
                    'X-CSRFToken': csrftoken
                }
            })
            .then(response => {
                if (response.ok) {
                    alert('Terapija je obrisana.');
                    event.target.closest('tr').remove();
                } else {
                    alert('Greška prilikom brisanja terapije.');
                }
            })
            .catch(error => console.error('Error:', error));
        } else if (event.target.classList.contains('edit-button')) {
            const therapyId = event.target.getAttribute('data-id');
            localStorage.setItem('therapyId', therapyId);
            window.location.href = '/dodaj_terapiju/';
        }
    });
});
```

Slika 42 Brisanje terapija i uređivanje postojećih terapija

```python
class Therapy(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    medicine_name = models.CharField(max_length=100)
    medicine_time = models.TimeField()
    medicine_dosage = models.IntegerField()
    medicine_description = models.TextField(blank=True, null=True)

    def __str__(self):
        return self.medicine_name
```

Slika 43 Model za "moje terapije"

```python
class TherapySerializer(serializers.ModelSerializer):
    class Meta:
        model = Therapy
        fields = ['id', 'medicine_name', 'medicine_time', 'medicine_dosage', 'medicine_description']
```

Slika 44 Serijalizer za "moje terapije"

5.3.1.3. Moji nalazi

Sekcija "Moji nalazi" (Slika 45) omogućava korisnicima da pregledavaju (Slika 46), dodaju (Slika 47), ažuriraju i preuzimaju svoje medicinske nalaze (Slika 48), osiguravajući da su svi podaci pravilno pohranjeni i lako dostupni. Model „MedicalReport“ koristi se za pohranu ovih podataka (Slika 49), a „MedicalReportSerializer“ osigurava validaciju podataka (Slika 50).

```html
<div id="reports" class="profile-section">
    <div class="reports-container">
        <div class="search-container">
            <input type="text" placeholder="Pretraga nalaza" class="search-input">
            <button class="search-button"><i class="fa fa-search"></i></button>
        </div>
    </div>
    <div class="reports-table-wrapper">
        <div class="reports-table-container">
            <table class="reports-table">
                <thead>
                    <tr>
                        <th>Datum nalaza</th>
                        <th>Ime nalaza</th>
                        <th>Akcije</th>
                    </tr>
                </thead>
                <tbody>
                </tbody>
            </table>
            <button class="add-button" onclick="window.location.href='{% url 'nalaz' %}'">Dodaj</button>
        </div>
        <div class="report-image">
            <img src="/static/images/nalaziprofil.png" alt="Nalazi Profil">
        </div>
    </div>
</div>
```

Slika 45 HTML kod za prikaz nalaza

```
fetch('/api/user/reports/', {
    method: 'GET',
    headers: {
        'Authorization': `Token ${token}`,
        'Content-Type': 'application/json'
    }
})
.then(response => response.json())
.then(data => {
    const tbody = document.querySelector('.reports-table tbody');
    tbody.innerHTML = '';
    data.forEach(report => {
        const tr = document.createElement('tr');
        tr.innerHTML = `
            <td>${report.report_date || ''}</td>
            <td>${report.report_name || ''}</td>
            <td>
                <button class="edit-button" data-id="${report.id}">Uredi</button>
                <button class="download-button" data-id="${report.id}">Preuzmi</button>
                <button class="delete-button" data-id="${report.id}">Obriši</button>
            </td>
        `;
        tbody.appendChild(tr);
    });
})
.catch(error => console.error('Error:', error));
```

Slika 46 Dohvaćanje postojećih nalaza iz baze podataka

```
    const method = reportId ? 'PUT' : 'POST';
    const url = reportId ? `/api/user/reports/${reportId}/` : '/api/user/reports/';

    fetch(url, {
        method: method,
        headers: {
            'Authorization': `Token ${token}`,
            'X-CSRFToken': getCsrfToken()
        },
        body: formData
    })
    .then(response => {
        if (response.ok) {
            alert('Nalaz je spremljen.');
            localStorage.removeItem('reportId');
            window.location.href = '/profil/#reports';
        } else {
            alert('Greška prilikom spremanja nalaza.');
        }
    })
    .catch(error => console.error('Error:', error));
});
```

Slika 47 Dodavanje novog nalaza

```
document.querySelector('.reports-table tbody').addEventListener('click', function(event) {
    const reportId = event.target.getAttribute('data-id');

    if (event.target.classList.contains('delete-button')) {
        fetch(`/api/user/reports/${reportId}/`, {
            method: 'DELETE',
            headers: {
                'Authorization': `Token ${token}`,
                'Content-Type': 'application/json',
                'X-CSRFToken': csrftoken
            }
        })
        .then(response => {
            if (response.ok) {
                alert('Nalaz je obrisan.');
                event.target.closest('tr').remove();
            } else {
                alert('Greška prilikom brisanja nalaza.');
            }
        })
        .catch(error => console.error('Error:', error));
    } else if (event.target.classList.contains('edit-button')) {
        localStorage.setItem('reportId', reportId);
        window.location.href = `/dodaj_nalaz/`;
    } else if (event.target.classList.contains('download-button')) {
        fetch(`/api/user/reports/${reportId}/download/`, {
            method: 'GET',
            headers: {
                'Authorization': `Token ${token}`
            }
        })
```

Slika 48 Brisanje, uređivanje i preuzimanje nalaza

```
class MedicalReport(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    report_name = models.CharField(max_length=255)
    report_date = models.DateField()
    report_description = models.TextField()
    report_file = models.FileField(upload_to='reports/')

    def __str__(self):
        return self.report_name
```

Slika 49 Model za "moji nalazi"

```
class MedicalReportSerializer(serializers.ModelSerializer):
    class Meta:
        model = MedicalReport
        fields = ['id', 'report_name', 'report_date', 'report_description', 'report_file']
```

Slika 50 Serijalizer za "moji nalazi "

5.3.1.4. Moji pregledi

Sekcija "Moji pregledi" (Slika 51) omogućava korisnicima praćenje medicinskih pregleda (Slika 52). Korisnici mogu dodavati (Slika 53), brisati i uređivati detalje o pregledima kao što su naziv i datum pregleda (Slika 54). Ovi podaci pohranjuju se u modelu „Exam“ (Slika 55), a validaciju osigurava „ExamSerializer“ (Slika 56).

```
<div id="exams" class="profile-section">
    <div class="exams-container">
        <div class="exams-table-container">
            <table class="exams-table">
                <thead>
                    <tr>
                        <th>Datum pregleda</th>
                        <th>Ime pregleda</th>
                        <th>Akcije</th>
                    </tr>
                </thead>
                <tbody>
                </tbody>
            </table>
            <button class="add-button" onclick="window.location.href='{% url 'pregled' %}'">Dodaj</button>
        </div>
        <div class="exam-image">
            <img src="/static/images/pregled.png" alt="Pregled">
        </div>
    </div>
</div>
```

Slika 51 HTML kod za prikazivanje pregleda

```
fetch('/api/user/exams/', {
    method: 'GET',
    headers: {
        'Authorization': `Token ${token}`,
        'Content-Type': 'application/json'
    }
})
.then(response => response.json())
.then(data => {
    const tbody = document.querySelector('.exams-table tbody');
    tbody.innerHTML = ''; // Očisti postojeće redove
    data.forEach(exam => {
        const tr = document.createElement('tr');
        tr.innerHTML = `
            <td>${exam.exam_date}</td>
            <td>${exam.exam_name}</td>
            <td>
                <button class="edit-button" data-id="${exam.id}">Uredi</button>
                <button class="delete-button" data-id="${exam.id}">Obriši</button>
            </td>
        `;
        tbody.appendChild(tr);
    });
})
.catch(error => console.error('Error:', error));
```

Slika 52 Dohvaćanje postojećih pregleda iz baze podataka

```
        fetch(url, {
            method: method,
            headers: {
                'Authorization': `Token ${token}`,
                'Content-Type': 'application/json',
                'X-CSRFToken': csrfToken
            },
            body: JSON.stringify(requestData)
        })
        .then(response => {
            if (response.ok) {
                alert('Pregled je uspješno spremljen.');
                localStorage.removeItem('examId');
                window.location.href = '/profil/#exams';
            } else {
                alert('Greška prilikom spremanja pregleda.');
            }
        })
        .catch(error => console.error('Error:', error));
    });

    cancelButton.addEventListener('click', function() {
        localStorage.removeItem('examId');
        window.location.href = '/profil/#exams';
    });
});
```

Slika 53 Dodavanje pregleda

```
document.querySelector('.exams-table tbody').addEventListener('click', function(event) {
    if (event.target.classList.contains('delete-button')) {
        const examId = event.target.getAttribute('data-id');
        fetch(`/api/user/exams/${examId}/`, {
            method: 'DELETE',
            headers: {
                'Authorization': `Token ${token}`,
                'Content-Type': 'application/json',
                'X-CSRFToken': csrfToken
            }
        })
        .then(response => {
            if (response.ok) {
                alert('Pregled je obrisan.');
                event.target.closest('tr').remove();
            } else {
                alert('Greška prilikom brisanja pregleda.');
            }
        })
        .catch(error => console.error('Error:', error));
    } else if (event.target.classList.contains('edit-button')) {
        const examId = event.target.getAttribute('data-id');
        localStorage.setItem('examId', examId);
        window.location.href = '/dodaj_pregled/';
    }
});
});
```

Slika 54 Brisanje i uređivanje pregleda

```python
class Exam(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    exam_name = models.CharField(max_length=100)
    exam_date = models.DateField()
    exam_description = models.TextField(blank=True, null=True)

    def __str__(self):
        return self.exam_name
```

Slika 55 Model za "moji pregledi"

```python
class ExamSerializer(serializers.ModelSerializer):
    class Meta:
        model = Exam
        fields = ['id', 'exam_name', 'exam_date', 'exam_description']
```

Slika 56 Serijalizer za "moji pregledi"

5.3.1.5. Vitalni znakovi

Sekcija "Vitalni znakovi" (Slika 57) omogućava korisnicima praćenje njihovih osnovnih zdravstvenih parametara kao što su krvni tlak, broj otkucaja srca, razina šećera u krvi i tjelesna temperatura (Slika 58). Također, svi se ti parametri mogu dodavati (Slika 59), brisati i uređivati (Slika 60). Podaci se pohranjuju u modelu „VitalSign“ (Slika 61), dok „VitalSignSerializer“ osigurava validaciju podataka (Slika 62).

```html
<div id="vital-signs" class="profile-section">
    <div class="vitalsigns-content">
        <div class="vitalsigns-table-container">
            <table class="vitalsigns-table">
                <thead>
                    <tr>
                        <th>Tlak</th>
                        <th>Broj otkucaja</th>
                        <th>Razina šećera u krvi</th>
                        <th>Tjelesna temperatura</th>
                        <th>Datum mjerenja</th>
                        <th>Akcije</th>
                    </tr>
                </thead>
                <tbody>
                </tbody>
            </table>
            <button class="add-button" onclick="window.location.href='{% url 'znak' %}'">Dodaj</button>
        </div>
        <div class="vitalsigns-image">
            <img src="/static/images/stetoskop.png" alt="Vitalni Znakovi">
        </div>
    </div>
    </div>
</section>
<button id="logout-button" class="logout-button">Odjavi se</button>
</div>
```

Slika 57 HTML kod za prikazivanje vitalnih znakova

```
fetch('/api/user/vital_signs/', {
    method: 'GET',
    headers: {
        'Authorization': `Token ${token}`,
        'Content-Type': 'application/json'
    }
})
.then(response => response.json())
.then(data => {
    const tbody = document.querySelector('.vitalsigns-table tbody');
    tbody.innerHTML = '';
    data.forEach(vitalSign => {
        const tr = document.createElement('tr');
        tr.innerHTML = `
            <td>${vitalSign.blood_pressure}</td>
            <td>${vitalSign.heart_rate}</td>
            <td>${vitalSign.blood_sugar}</td>
            <td>${vitalSign.body_temperature}</td>
            <td>${vitalSign.measurement_date}</td>
            <td>
                <button class="edit-button" data-id="${vitalSign.id}">Uredi</button>
                <button class="delete-button" data-id="${vitalSign.id}">Obriši</button>
            </td>
        `;
        tbody.appendChild(tr);
    });
})
.catch(error => console.error('Error:', error));
```

Slika 58 Dohvaćanje vitalnih znakova iz baze podataka

```
fetch(url, {
    method: method,
    headers: {
      'Authorization': `Token ${token}`,
      'Content-Type': 'application/json',
      'X-CSRFToken': csrftoken
    },
    body: JSON.stringify(data)
  })
  .then(response => {
    if (response.ok) {
      alert('Vitalni znak je spremljen.');
      localStorage.removeItem('vitalSignId');
      window.location.href = '/profil/#vital-signs';
    } else {
      alert('Greška prilikom spremanja vitalnog znaka.');
    }
  })
  .catch(error => console.error('Error:', error));
});
```

Slika 59 Dodavanje vitalnih znakova

```
    document.querySelector('.vitalsigns-table tbody').addEventListener('click', function(event) {
        if (event.target.classList.contains('delete-button')) {
            const vitalSignId = event.target.getAttribute('data-id');
            fetch(`/api/user/vital_signs/${vitalSignId}/`, {
                method: 'DELETE',
                headers: {
                    'Authorization': `Token ${token}`,
                    'Content-Type': 'application/json',
                    'X-CSRFToken': csrftoken
                }
            })
            .then(response => {
                if (response.ok) {
                    alert('Vitalni znak je obrisan.');
                    event.target.closest('tr').remove();
                } else {
                    alert('Greška prilikom brisanja vitalnog znaka.');
                }
            })
            .catch(error => console.error('Error:', error));
        } else if (event.target.classList.contains('edit-button')) {
            const vitalSignId = event.target.getAttribute('data-id');
            localStorage.setItem('vitalSignId', vitalSignId);
            window.location.href = '/dodaj_znak/';
        }
    });
});
```

Slika 60 Brisanje i uređivanje vitalnih znakova

```
class VitalSign(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    blood_pressure = models.CharField(max_length=10)
    heart_rate = models.CharField(max_length=10)
    blood_sugar = models.CharField(max_length=10)
    body_temperature = models.CharField(max_length=10)
    measurement_date = models.DateField(auto_now_add=True)

    def __str__(self):
        return f"VitalSign {self.measurement_date} - {self.user.username}"
```

Slika 61 Model za „vitalni znakovi“

```
class VitalSignSerializer(serializers.ModelSerializer):
    class Meta:
        model = VitalSign
        fields = ['id', 'blood_pressure', 'heart_rate', 'blood_sugar', 'body_temperature', 'measurement_date']
```

Slika 62 Serijalizer za „vitalni znakovi“

5.4. Stranica za praćenje statistike

Stranica "Statistika" pruža korisnicima mogućnost vizualizacije njihovih zdravstvenih parametara grafičkim prikazom. Korisnicima je omogućeno praćenje promjena u njihovim vitalnim znakovima poput krvnog tlaka, razine šećera u krvi, broja otkucaja srca i tjelesne temperature, pružajući im tako uvid u njihove osobne zdravstvene trendove i potencijalne probleme.

Kada korisnik otvori stranicu "Statistika", aplikacija mu prikazuje padajući izbornik s mogućnostima odabira različitih zdravstvenih parametara koje želi vizualizirati (Slika 63). Nakon što korisnik odabere željeni parametar, npr. krvni tlak, JavaScript funkcija na „frontend“ strani šalje „GET“ zahtjev na „backend“ server putem „API endpointa“ „/api/statistics/<measurement_type>/“, gdje „<measurement_type>“ predstavlja odabrani parametar (npr. „blood_pressure“).

Na „backend“ strani, zahtjev obrađuje klasa „StatisticsView“ (Slika 64) koja zatim nasljeđuje „APIView“ iz Django „REST framework-a“. „StatisticsView“ klasa odgovorna je za dohvaćanje podataka o odabranom parametru iz baze podataka. Unutar metode „GET“, podaci se filtriraju prema prijavljenom korisniku i vremenskom rasponu od proteklih sedam dana. Na primjer, ako je korisnik odabrao krvni tlak, podaci se dohvaćaju iz modela „VitalSign“ filtriranjem po korisniku i datumu mjerenja.

„Backend“ koristi ORM (Object-Relational Mapping) upite za dohvaćanje podataka, a rezultat se serijalizira u JSON format i vraća na „frontend“. JSON podaci uključuju datume mjerenja i odgovarajuće vrijednosti parametara.

Nakon što su podaci uspješno dohvaćeni na „frontend“ strani, JavaScript koristi Chart.js za prikaz podataka u obliku linijskog grafa. Graf prikazuje vremenski slijed odabranog parametra, omogućavajući korisnicima vizualno praćenje promjena kroz vremenski period od sedam dana. JavaScript funkcija također izračunava prosječnu vrijednost parametra (Slika 64) i prikazuje je korisniku pored grafa uz određeni emoji (sretni, tužni i normalni) (Slika 65) pomoću prosječne vrijednosti i definiranih vrijednosti, pružajući dodatni uvid u opće stanje njihovog zdravlja. Korisnik također može pogledati tablicu normalnih vrijednosti koja je informativnog karaktera (Slika 9).

```
30 ▼    <div class="content">
31 ▼        <section id="statistics">
32              <h2>Statistika</h2>
33 ▼            <div class="statistics-select">
34                  <label for="measurement">Mjerenje:</label>
35 ▼                <div class="select-wrapper">
36 ▼                    <select id="measurement">
37                          <option value="blood_pressure">Krvni tlak</option>
38                          <option value="sugar">Razina šećera u krvi</option>
39                          <option value="bpm">Broj otkucaja u minuti</option>
40                          <option value="temperature">Tjelesna temperatura</option>
41                      </select>
42                  </div>
43              </div>
44 ▼            <div class="statistics-content">
45 ▼                <div class="statistics-graph">
46                      <canvas id="statisticsGraph"></canvas>
47                  </div>
48 ▼                <div class="statistics-result">
49                      <img id="resultIcon" src="/static/images/neutral.png" alt="Result Icon">
50                      <p id="resultText">Prosječan rezultat: <span id="averageResult"></span></p>
51                      <p>Tablicu normalnih parametara možete pronaći <a
                        href="https://drive.google.com/file/d/1ZOPY4R912Uha_66LXEiiNeTSnFY83cED/view?usp=drive_link">ovdje</a>.
                        </p>
52                  </div>
53              </div>
54          </section>
55      </div>
```

Slika 63 HTML kod za prikaz grafa, prosječne vrijednosti i emoji-a

```python
class StatisticsView(APIView):
    permission_classes = [IsAuthenticated]

    def get(self, request, measurement_type):
        user = request.user
        today = datetime.today()
        week_ago = today - timedelta(days=7)

        if measurement_type == 'blood_pressure':
            data = VitalSign.objects.filter(user=user, measurement_date__range=[week_ago, today]).values('measurement_date', 'blood_pressure')
            field_name = 'blood_pressure'
        elif measurement_type == 'sugar':
            data = VitalSign.objects.filter(user=user, measurement_date__range=[week_ago, today]).values('measurement_date', 'blood_sugar')
            field_name = 'blood_sugar'
        elif measurement_type == 'bpm':
            data = VitalSign.objects.filter(user=user, measurement_date__range=[week_ago, today]).values('measurement_date', 'heart_rate')
            field_name = 'heart_rate'
        elif measurement_type == 'temperature':
            data = VitalSign.objects.filter(user=user, measurement_date__range=[week_ago, today]).values('measurement_date', 'body_temperature')
            field_name = 'body_temperature'
        else:
            return Response({'error': 'Invalid measurement type'}, status=400)

        response_data = [{'date': entry['measurement_date'].strftime('%Y-%m-%d'), 'value': entry[field_name]} for entry in data]

        return Response(response_data)
```

Slika 64 „View“ za prikaz određenog mjerenja u zadnjih 7 dana

```javascript
function updateResultIcon(average, measurementType) {
    let icon = '/static/images/neutral.png';
    let text = 'Prosječan rezultat:';

    switch (measurementType) {
        case 'blood_pressure':
            if (average <= 129) {
                icon = '/static/images/sretan.png';
            } else if (average >= 130 && average <= 139) {
                icon = '/static/images/normal.png';
            } else {
                icon = '/static/images/tuzan.png';
            }
            break;
        case 'sugar':
            if (average < 100) {
                icon = '/static/images/sretan.png';
            } else if (average >= 100 && average <= 125) {
                icon = '/static/images/normal.png';
            } else {
                icon = '/static/images/tuzan.png';
            }
            break;
```

Slika 65 Prikaz određenog emoji-a

```
16 ▼    measurementSelect.addEventListener('change', function() {
17          const measurementType = measurementSelect.value;
18          fetchStatisticsData(measurementType);
19      });
20
21 ▼    function fetchStatisticsData(measurementType) {
22 ▼        fetch(`/api/user/statistics/${measurementType}/`, {
23              method: 'GET',
24 ▼            headers: {
25                  'Authorization': `Token ${token}`,
26                  'Content-Type': 'application/json'
27              }
28          })
29          .then(response => response.json())
30 ▼        .then(data => {
31              const labels = data.map(entry => entry.date);
32              const values = data.map(entry => parseFloat(entry.value));
33              const average = (values.reduce((a, b) => a + b, 0) / values.length).toFixed(2);
34              averageResult.textContent = `${average} ${getUnit(measurementType)}`;
```

Slika 66 Odabir, dohvaćanje i izračun prosječne vrijednosti mjerenja

5.5. Kontakt stranica

5.5.1. Kontakt specijaliste

Stranica za kontakt specijaliste omogućava korisnicima laku komunikaciju s raznim medicinskim specijalistima putem aplikacije (Slika 67). Ova funkcionalnost je ključna za pružanje podrške korisnicima koji trebaju savjete ili odgovore na specifična pitanja koja zahtijevaju stručan medicinski odgovor.

Nakon što korisnik pristupi stranici za kontakt specijaliste, prikazuje mu se forma koja zahtijeva unos osobnih podataka (ime, prezime, email adresa) i odabir tipa specijaliste (neurolog, oftalmolog, kardiolog, dermatolog). Kada korisnik ispuni formu i klikne gumb za slanje iste, JavaScript funkcija na „frontend“ strani šalje „POST“ zahtjev na „backend“ server putem „API endpoint-a“ „/api/contact-specialist/“. Ovaj zahtjev uključuje podatke koje je korisnik unio u formu (Slika 68).

Na „backend“ strani, zahtjev obrađuje klasa „ContactSpecialistView“ koja nasljeđuje „APIView“ iz Django „REST framework-a“. Metoda „POST“ unutar ove klase dohvaća podatke iz zahtjeva, uključujući specijalnost, subjekt, poruku i email adresu pošiljatelja. Na temelju odabrane specijalnosti, metoda određuje email adresu primatelja koristeći unaprijed definirani „specialist_emails“ (Slika 23).

Nakon što su svi podaci prikupljeni, funkcija „send_mail“ iz Django modula „django.core.mail“ koristi se za slanje email-a željenom specijalisti. Email sadrži podatke o pošiljatelju i sadržaj poruke. Kod uspješnog slanja email-a, korisnik dobiva potvrdnu obavijest o slanju poruke.

Da bi slanje email-ova bilo uspješno i funkcionalno, u settings.py datoteci konfigurirane su postavke za email koje specificiraju da se koristi Gmail-ov SMTP (Simple Mail Transfer Protocol) server za slanje email-ova. Postavke uključuju detalje poput adrese SMTP servera i broja porta (Slika 69). Korištenjem ovih postavki osigurava se sigurna i pouzdana komunikacija između aplikacije i email servera, omogućavajući tako uspješno slanje email-ova specijalistima.

Nakon što su svi podaci prikupljeni, funkcija „send_mail“ iz Django modula „django.core.mail“ koristi se za slanje email-a specijalistu.

```html
<div class="content">
    <section id="contact">
        <h2>Kontaktirajte doktora</h2>
        <div class="contact-specialist">
            <label for="specialist">Specijalist:</label>
            <div class="select-wrapper">
                <select id="specialist">
                    <option value="neurolog">Neurolog</option>
                    <option value="oftamolog">Oftamolog</option>
                    <option value="kardiolog">Kardiolog</option>
                    <option value="dermatolog">Dermatolog</option>
                </select>
            </div>
        </div>
        <div class="contact-content">
            <div class="contact-info">
                <p>
                    <label for="name">Ime i prezime:</label>
                    <input type="text" id="name" placeholder="Ime i prezime...">
                </p>
                <p>
                    <label for="email">Email:</label>
                    <input type="email" id="email" placeholder="Vaš email...">
                </p>
                <p>
                    <label for="question">Vaše pitanje:</label>
                    <textarea id="question" placeholder="Napišite pitanje..."></textarea>
                </p>
                <button class="send-button" id="sendButton">Pošalji</button>
            </div>
```

Slika 67 HTML kod za odabir specijaliste i forma za pitanje

```javascript
        fetch('/api/contact-specialist/', {
            method: 'POST',
            headers: {
                'Authorization': `Token ${token}`,
                'Content-Type': 'application/json',
                'X-CSRFToken': getCsrfToken()
            },
            body: JSON.stringify({
                specialist: specialist,
                subject: `Pitanje od ${name}`,
                message: question,
                email: email
            })
        })
        .then(response => {
            if (response.ok) {
                alert('Email je uspješno poslan.');
            } else {
                alert('Došlo je do greške prilikom slanja emaila.');
            }
        })
        .catch(error => console.error('Error:', error));
    });
```

Slika 68 Slanje upita specijalisti

```
116  EMAIL_BACKEND = 'django.core.mail.backends.smtp.EmailBackend'
117  EMAIL_HOST = 'smtp.gmail.com'
118  EMAIL_PORT = 587
```

Slika 69 Postavke za slanje email-a

5.5.2. Kontakt korisničke podrške stranica

Stranica za kontakt korisničke podrške omogućava korisnicima postavljanje pitanja u vezi same aplikacije ili prijavu problema vezanu uz korištenje iste. Ovom funkcionalnošću osigurano je da korisnici uvijek mogu dobiti pomoć ukoliko je zatrebaju, a da je vezana uz aplikaciju i njene funkcionalnosti.

Nakon što korisnik pristupi stranici za kontakt korisničke podrške, prikazuje mu se forma s poljima za unos osobnih podataka (ime, prezime, email adresa) i polje za unos pitanja ili problema (Slika 70). Kad korisnik ispuni formu i klikne gumb za slanje, JavaScript funkcija na „frontend“ strani šalje „POST“ zahtjev na „backend“ server putem „API endpoint-a“ „/api/contact-support/“ (Slika 71).

„Backend“ zahtjev obrađuje klasa „ContactSupportView“, koja nasljeđuje „APIView“ iz Django „REST frameworka“. Metoda „POST“ unutar ove klase dohvaća podatke iz zahtjeva, uključujući ime, email adresu pošiljatelja i sadržaj poruke (Slika 24).

Nakon što su svi podaci prikupljeni, funkcija „send_mail“ koristi se za slanje email-a timu za korisničku podršku. Email sadrži podatke o pošiljatelju i sadržaj poruke. Nakon uspješnog slanja emaila, korisnik dobiva potvrdu o uspjehu. Za ispravno funkcioniranje primanja email-a na zadanu adresu bilo je potrebno napraviti aplikacijsku lozinku unutar postavki Gmail-a jer inače Gmail ne dopušta primanje email-ova kroz aplikacije koje ne podržavaju dvofaktorsku autentifikaciju (2FA). Aplikacijska lozinka se koristi također kako bi se povećala sigurnost.

```html
<div class="content">
    <section id="support">
        <h2>Korisnička podrška</h2>
        <div class="support-content">
            <div class="support-info">
                <p>
                    <label for="name">Ime i prezime:</label>
                    <input type="text" id="name" placeholder="Ime i prezime...">
                </p>
                <p>
                    <label for="email">Email:</label>
                    <input type="email" id="email" placeholder="Vaš email...">
                </p>
                <p>
                    <label for="question">Vaše pitanje:</label>
                    <textarea id="question" placeholder="Napišite pitanje..."></textarea>
                </p>
                <button class="send-button">Pošalji</button>
            </div>
            <div class="support-details">
                <div class="support-picture">
                    <img id="support-image" src="/static/images/Podrska.png" alt="Podrška">
                </div>
            </div>
        </div>
    </section>
</div>
```

Slika 70 HTML kod za ispunjavanje forme korisničke podrške

```javascript
        if (name && email && message) {
            fetch('/api/contact-support/', {
                method: 'POST',
                headers: {
                    'Content-Type': 'application/json',
                    'Authorization': `Token ${token}`,
                    'X-CSRFToken': csrftoken
                },
                body: JSON.stringify({
                    name: name,
                    email: email,
                    message: message,
                })
            })
            .then(response => response.json())
            .then(data => {
                if (data.message) {
                    alert('Vaš upit je uspješno poslan.');
                } else {
                    alert('Došlo je do greške prilikom slanja upita.');
                }
            })
            .catch(error => console.error('Error:', error));
        } else {
            alert('Molimo ispunite sva polja.');
        }
    });
```

Slika 71 Slanje upita korisničkoj podršci

6. Testiranje aplikacije

Testiranje aplikacije za vođenje osobnog zdravstvenog dnevnika završni je dio razvoja od iznimne važnosti, osiguravajući da sve funkcionalnosti rade ispravno i da su podaci korisnika sigurni i točni. Kroz cijeli proces razvoja, svaka nova funkcionalnost je testirana odmah nakon implementacije. Takvim iterativnim i linearnim pristupom omogućena je momentalna identifikaciju problema i ispravljanje grešaka, osiguravajući tako da aplikacija radi ispravno u svakom trenutku.

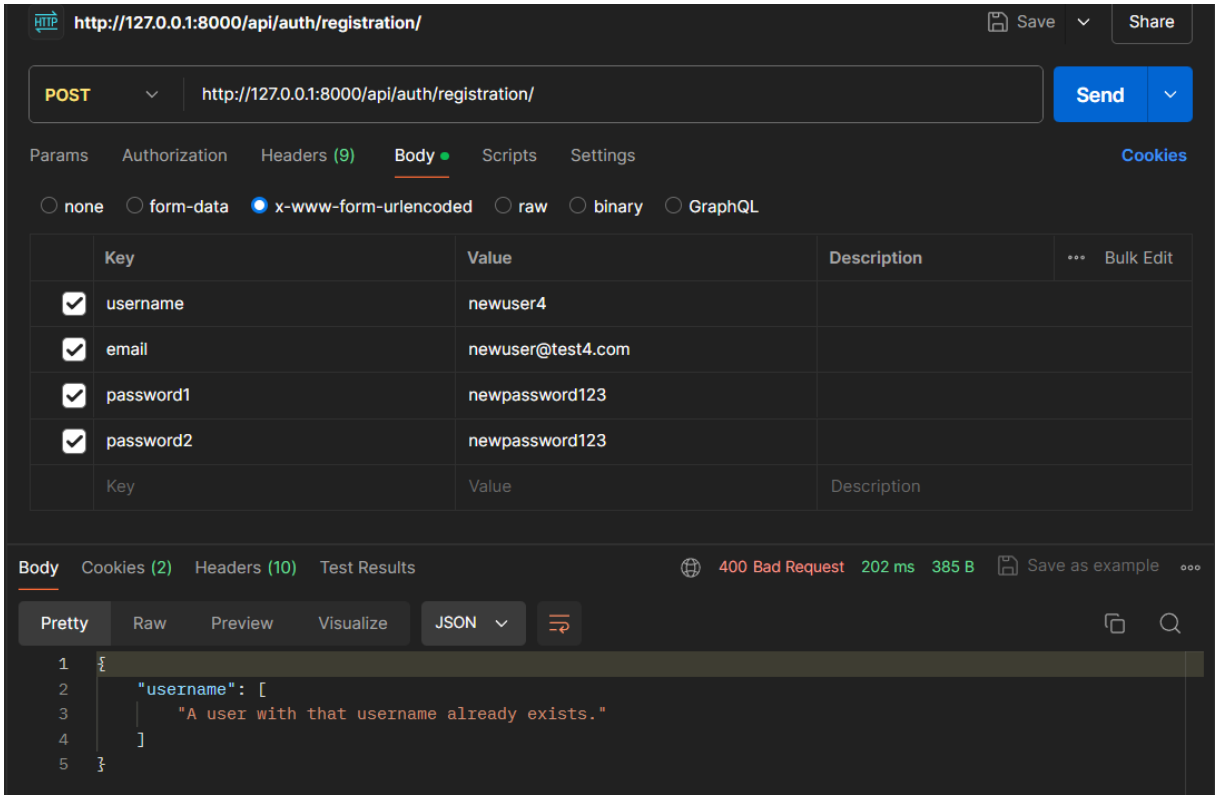
6.1. Način izvedbe

Korišteni su različiti alati i tehnike za testiranje aplikacije, uključujući Postman za „API“ testiranje, ispisi iz naredbenog retka (Command prompt) te provjere unutar koda. Testiranje je uključivalo slanje ispravnih i neispravnih podataka kako bi se provjerila validacija i autentifikacija korisnika.

Postman je popularan alat za razvoj i testiranje „API-ja“ koji omogućuje slanje HTTP zahtjeva i pregledavanje odgovora na jednostavan i intuitivan način. Korištenje Postman-a je postalo standardna praksa u razvoju softvera zbog njegove fleksibilnosti, bogatih značajki i jednostavnosti korištenja. Postman omogućava slanje različitih vrsta HTTP zahtjeva, uključujući „GET“, „POST“, „PUT“, „DELETE“, „PATCH“ i druge, što je korisno za testiranje „RESTful API-ja“. Programeri mogu definirati sve aspekte HTTP zahtjeva, uključujući URL, zaglavlja, tijelo zahtjeva, autentifikaciju i parametre upita. Nakon slanja zahtjeva, Postman prikazuje detaljan odgovor koji uključuje statusni kod, tijelo odgovora, zaglavlja i vrijeme odziva [17].

Za testiranje ove aplikacije, Postman je konkretno korišten za slanje različitih HTTP zahtjeva („GET“, „POST“, „PUT“ i „DELETE“) kako bi se provjerila ispravnost registracije (Slika 72), prijave korisnika, dodavanja terapija, nalaza, pregleda, vitalnih znakova, slanje email-ova specijalistima i korisničkoj podršci. Postman je omogućio brzo i jednostavno slanje zahtjeva te provjeru odgovora servera, osiguravajući da svi „endpoint-ovi“ vraćaju očekivane rezultate. Ukoliko to nije bio slučaj, pomagao je u otkrivanju i rješavanju problema.

Za stranicu „Statistika“ provjereno je ukoliko je graf ispravan pomoću pregleda x i y osi koje ispisuju datum mjerenja i mjerivi raspon za traženo mjerenje. Također, svaki stupac prelaskom miša prikazuje koliko točno iznosi njegovo mjerenje. Za provjeru ispravnosti prikaza emoji-a na stranici „Statistika“, upisalo se sedam testnih podataka unutar sekcije „vitalni znakovi“. Nakon upisivanja podataka, na izborniku na stranici „Statistika“ se odabrao svaki parametar i provjerio odgovara li dobiveni emoji prosječnoj vrijednosti, npr. ukoliko je prosječna vrijednost za temperaturu bila 40, trebao se pokazati tužni emoji.



Slika 72 Primjer provjere kreiranja novog korisnika

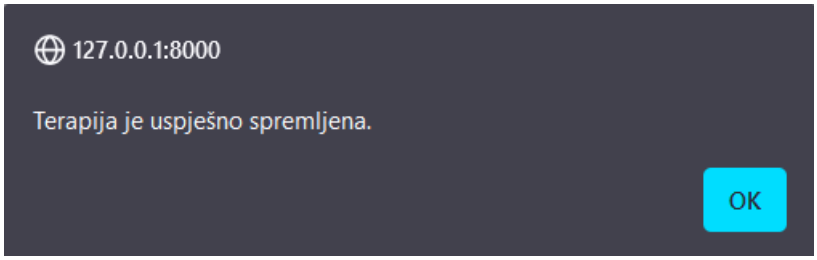
Tijekom razvoja, Django server je pokrenut u razvojnom okruženju, što je omogućilo praćenje svih ispisa u naredbenom retku. Ovi ispisi su uključivali informacije o uspješnosti HTTP zahtjeva, greškama i upozorenjima (Slika 73). Pomoću ovih zapisa je bilo lako ustanoviti jesu li sve datoteke uspješno učitane tokom otvaranje stranice, te ukoliko nije, bilo je lako ispraviti grešku jer je ispisivao o kakvoj grešci je riječ.

```
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
July 08, 2024 - 20:42:56
Django version 5.0.6, using settings 'myproject.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.

[08/Jul/2024 20:43:03] "GET / HTTP/1.1" 200 5581
[08/Jul/2024 20:43:03] "GET /static/js/scripts.js HTTP/1.1" 304 0
[08/Jul/2024 20:43:03] "GET /static/css/styles.css HTTP/1.1" 200 36509
[08/Jul/2024 20:43:03] "GET /static/images/Graf.png HTTP/1.1" 200 42686
[08/Jul/2024 20:43:03] "GET /static/images/Nalazi.png HTTP/1.1" 200 139612
[08/Jul/2024 20:43:03] "GET /static/images/Terapije.png HTTP/1.1" 200 178070
```

Slika 73 Ispis ispravnosti učitavanja svih datoteka na početnoj stranici

Unutar koda su postavljene provjere koje su vraćale poruke o uspješnosti ili neuspješnosti određenih operacija. Na primjer, nakon dodavanje terapije, funkcija bi vratila poruku koja je pokazivala je li terapija uspješno spremljena ili nije (Slika 74). Ove provjere omogućile su brzu

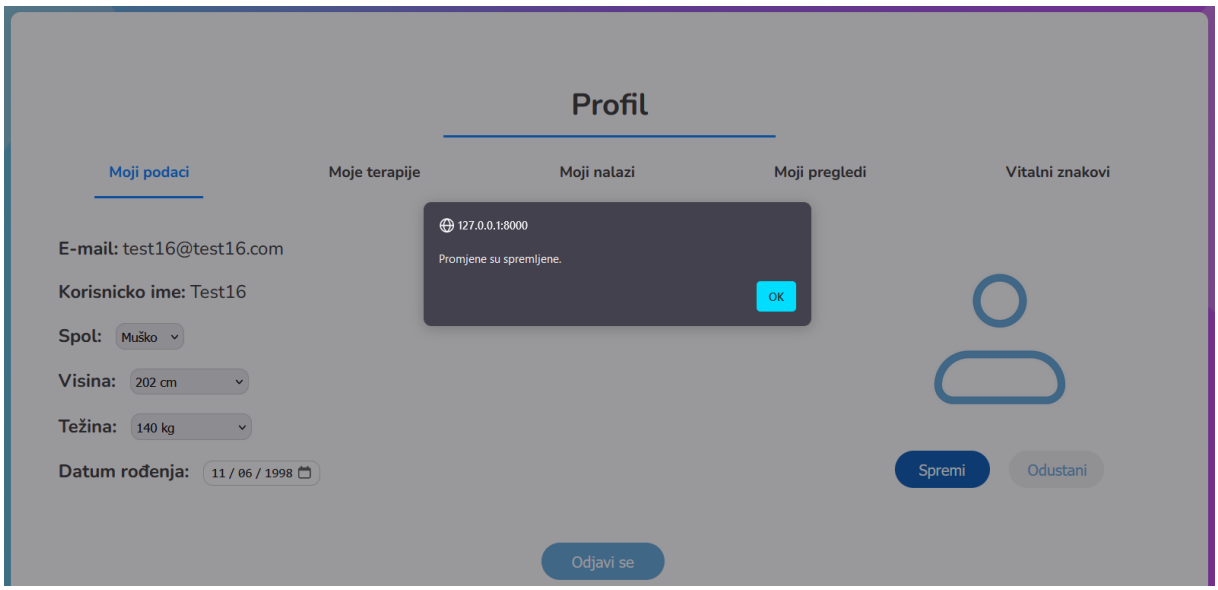
identifikaciju problema tijekom razvoja i testiranja jer se točno moglo znati koji dio koda ne radi.



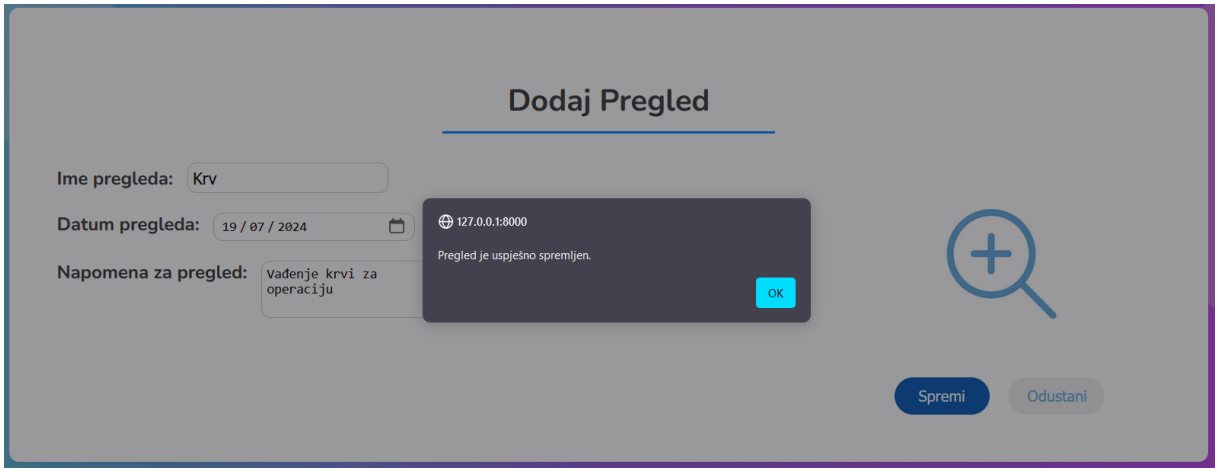
Slika 74 Poruka o uspješnosti spremanje terapije

6.2. Rezultati testiranja

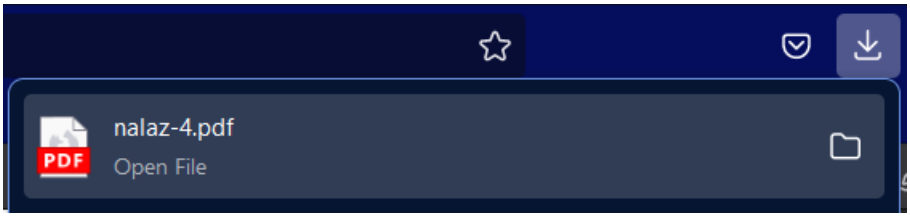
Krajnji rezultati testiranja pokazali su da su sve funkcionalnosti aplikacije rade ispravno, što uključuje registraciju i prijavu korisnika, upisivanje podataka za spol, visinu, težinu i datum rođenja (Slika 75), dodavanje, ažuriranje i brisanje terapija, nalaza, pregleda (Slika 76) i vitalnih znakova, te preuzimanje nalaza (Slika 77), prikazivanje odgovarajućeg emoji-a (Slika 78) (Slika 79) (Slika 80) i slanje email-ova korisničkoj podršci (Slika 81) i specijalistima (Slika 82) Svi testovi su uspješno prošli, potvrđujući da aplikacija ispravno radi. Testirani su svi CRUD „endpoint-ovi" za upravljanje terapijama, nalazima, pregledima i vitalnim znakovima. Testovi su pokazali da svi „endpoint-ovi" rade ispravno i da se podaci pravilno pohranjuju i prikazuju korisnicima. Testirano je slanje email-ova te su sve poruke ispravno poslane, a korisnici su dobili potvrdu o uspješnosti slanja.



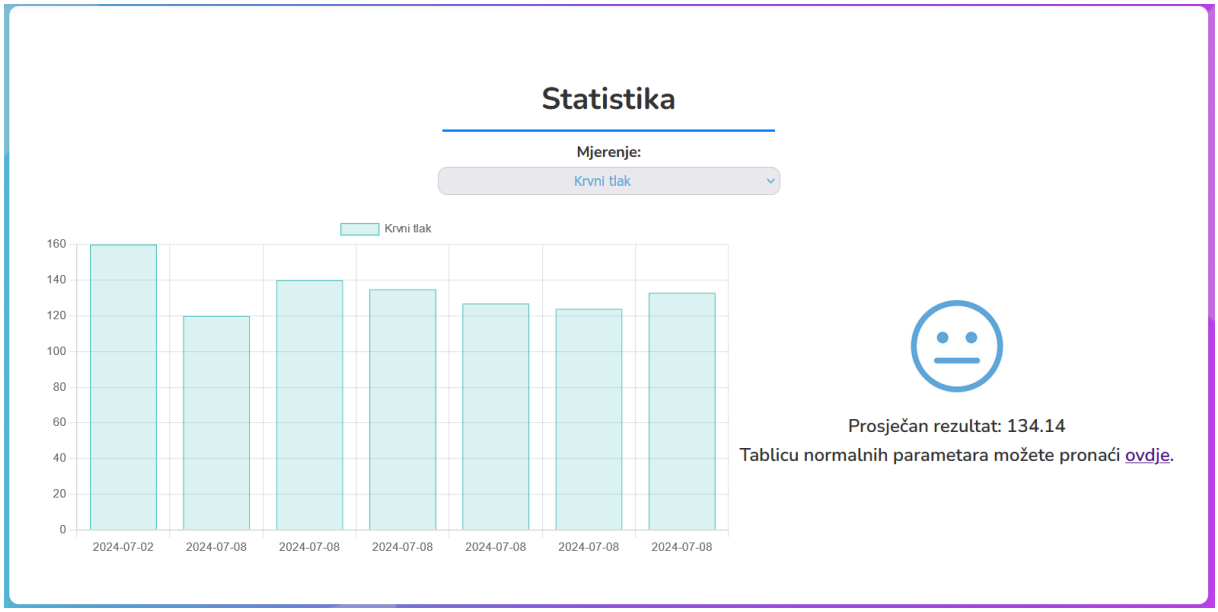
Slika 75 Spremanje mojih podataka



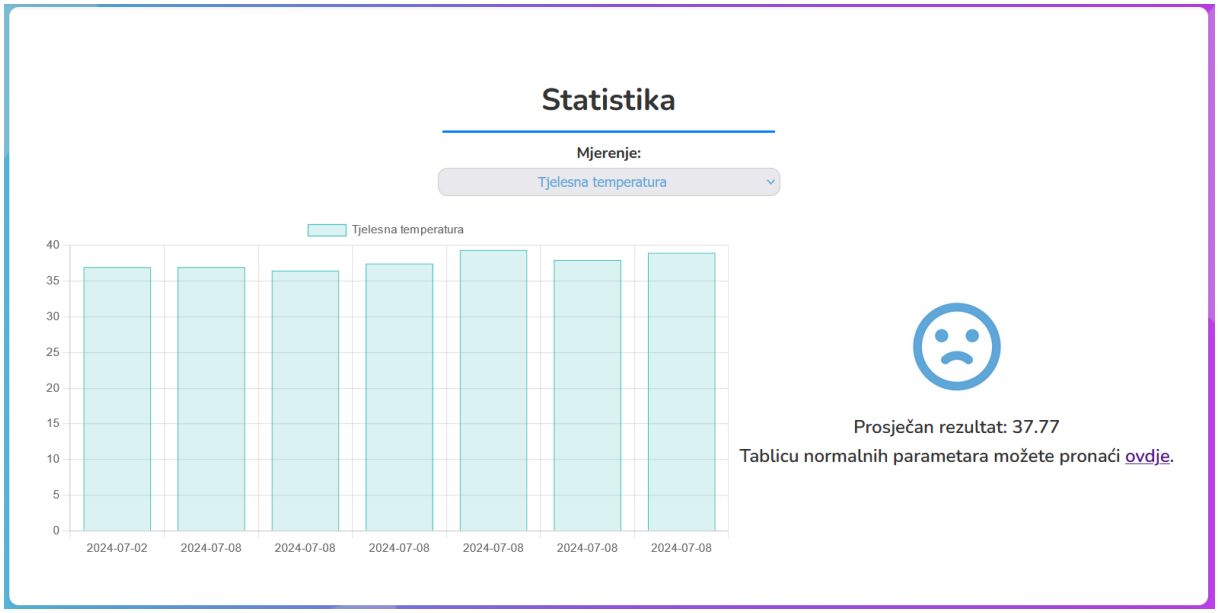
Slika 76 Dodavanje pregleda



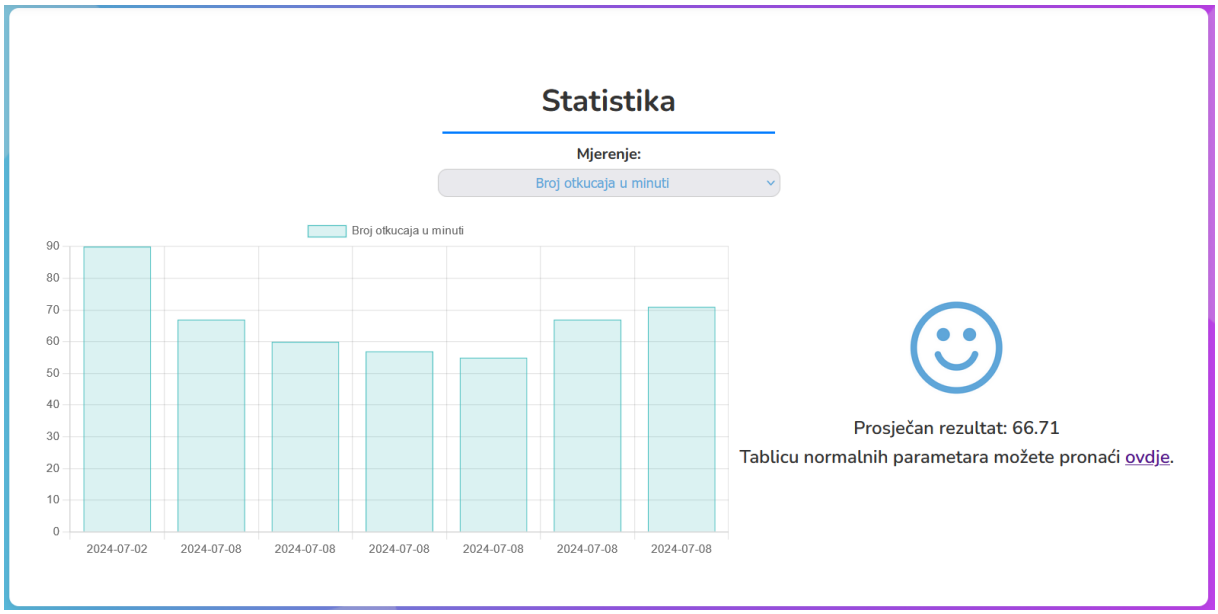
Slika 77 Preuzimanje nalaza



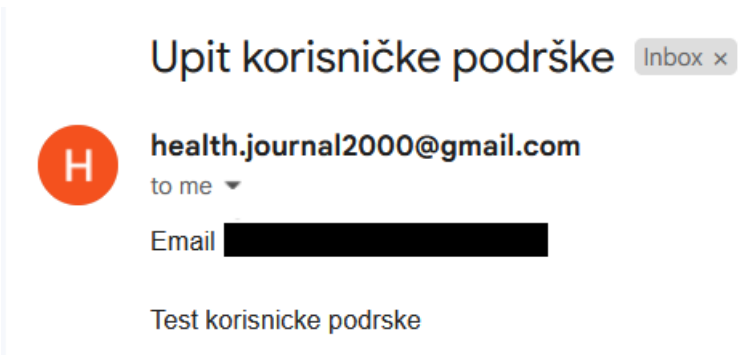
Slika 78 Prikaz grafa za krvni tlak i emoji



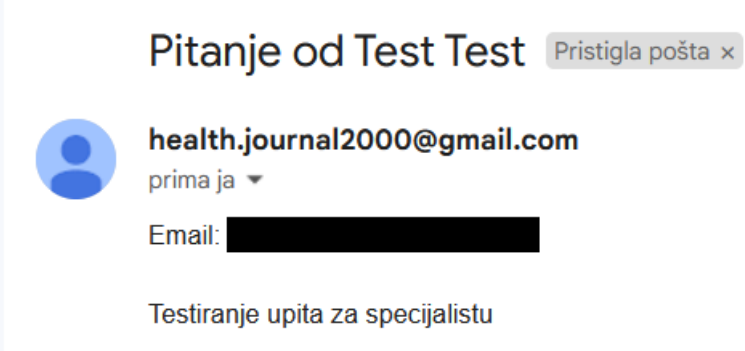
Slika 79 Prikaz grafa za tjelesnu temperaturu i emoji



Slika 80 Prikaz grafa za broj otkucaja u minuti i emoji



Slika 81 Testiranje primanje email-a za korisničku podršku



Slika 82 Testiranje primanje email-a za specijalistu

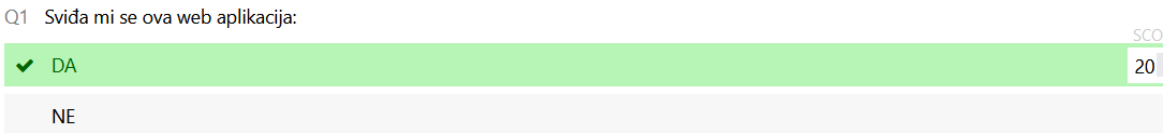
Rezultati testiranja pokazali su da je aplikacija za vođenje osobnog zdravstvenog dnevnika stabilna, sigurna i spremna za upotrebu. Sve funkcionalnosti su temeljito testirane i potvrđeno je da ispravno rade, pružajući korisnicima pouzdano i učinkovito sredstvo za upravljanje njihovim zdravstvenim podacima.

7. Korisnički dojmovi o aplikaciji

Za proučavanje korisničkih dojmova o web aplikaciji dnevnika za vođenje zdravlja, napravljena je anketa na dvadeset ljudi koji su najprije isprobali aplikaciju, a zatim odgovorili na pitanja u anketi.

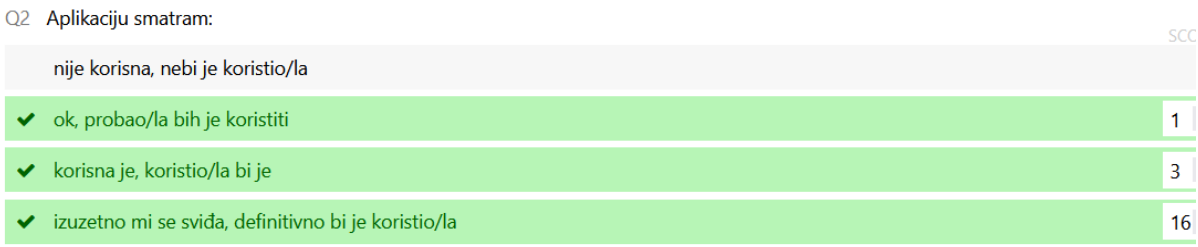
Gledajući rezultate ankete, može se zaključiti da aplikacija pruža visok stupanj zadovoljstva među korisnicima.

Iz prvog pitanja (Slika 83), gdje 20/20 ispitanika odgovara sa 'DA' na pitanje sviđa li im se aplikacija, jasno je vidljivo da su ispunjena sva očekivanja u vezi iste.



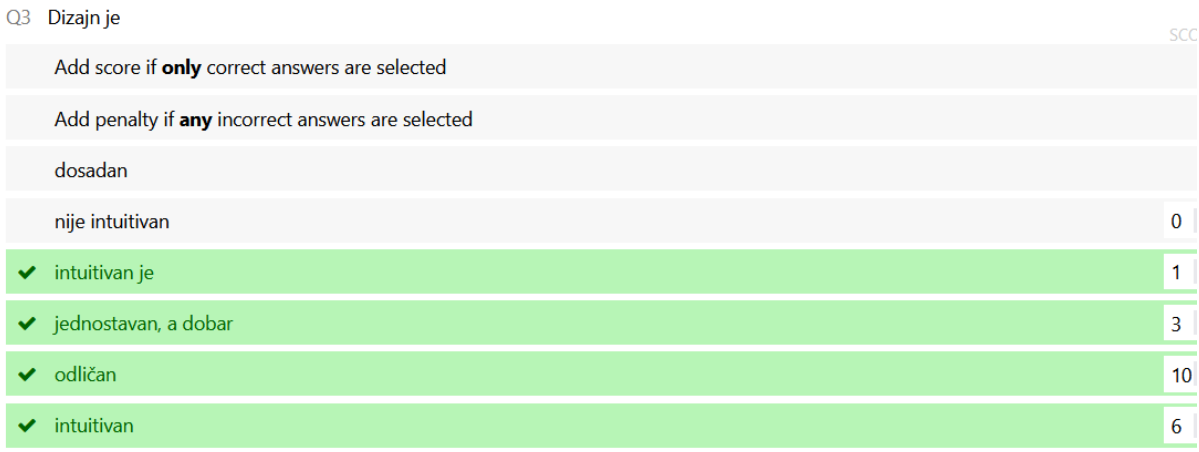
Slika 83 Pitanje ankete 1

Drugo pitanje (Slika 84), koje se odnosi na korisnost aplikacije, pokazuje da većina korisnika smatra aplikaciju korisnom, a neki su izrazili i izuzetno visoko zadovoljstvo, navodeći da bi je definitivno koristili. Ovaj rezultat upućuje na to da aplikacija ne samo da zadovoljava osnovne potrebe korisnika, već ih i nadmašuje, pružajući značajnu vrijednost u upravljanju zdravstvenim podacima.



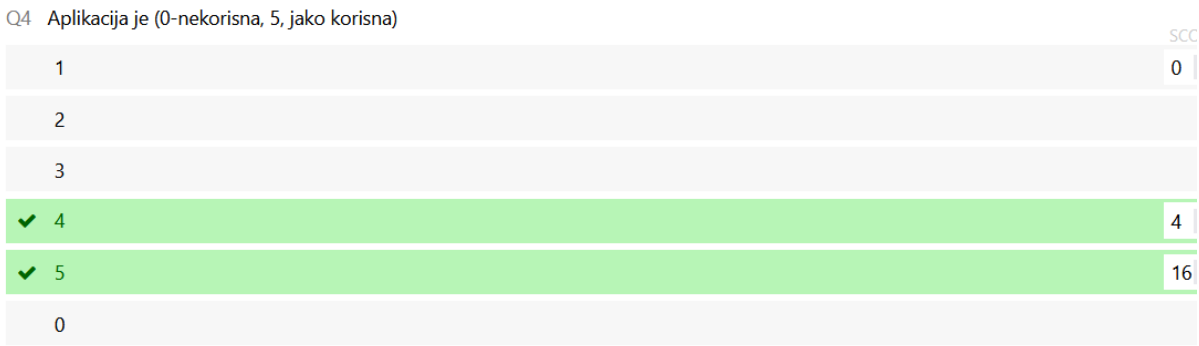
Slika 84 Pitanje ankete 2

Rezultati trećeg pitanja (Slika 85) naglašavaju intuitivnost i kvalitetu dizajna aplikacije. Većina ispitanika je ocijenila dizajn kao intuitivan, jednostavan i odličan, što ukazuje na uspješno implementiran korisnički interfejs koji olakšava navigaciju i korištenje aplikacije.



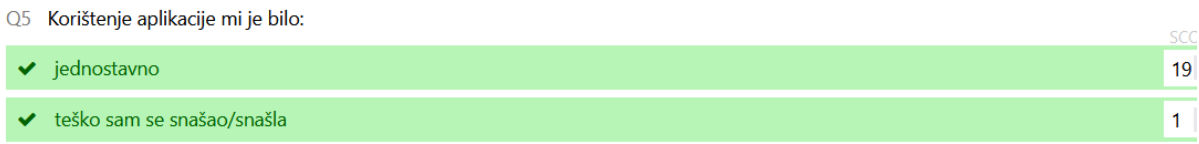
Slika 85 Pitanje ankete 3

U četvrtom pitanju (Slika 86), koje se odnosi na ukupnu korisnost aplikacije, ispitanici su većinom ocijenili aplikaciju visokom ocjenom, što dodatno potvrđuje njenu učinkovitost i praktičnost.



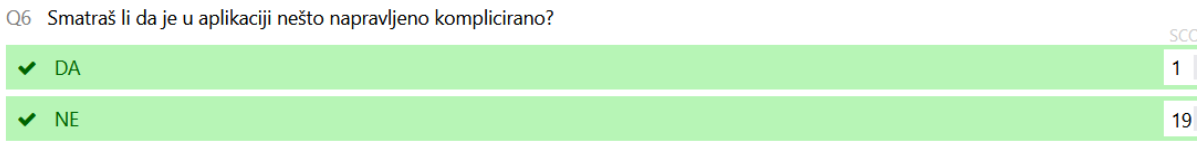
Slika 86 Pitanje ankete 4

Pitanje pet (Slika 87) se bavi jednostavnošću korištenja aplikacije. Većina korisnika navela je da im je korištenje aplikacije bilo jednostavno, što potvrđuje prethodne nalaze o intuitivnom dizajnu i lakoći navigacije kroz različite funkcionalnosti.



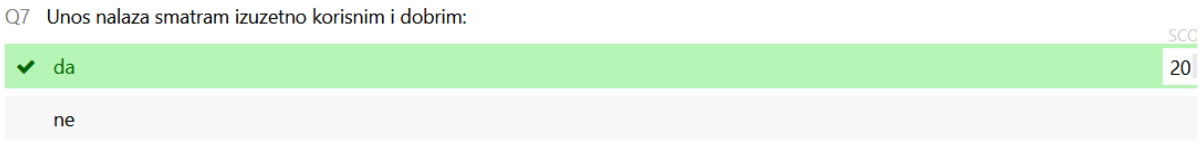
Slika 87 Pitanje ankete 5

U šestom pitanju (Slika 88) provjerava se je li nešto u aplikaciji napravljeno kompliciranо. Odgovori pokazuju da većina korisnika ne smatra aplikaciju kompliciranom, što dodatno potvrđuje da je dizajn usmjeren na korisnika i da uspješno zadovoljava potrebe bez stvaranja zbunjenosti.



Slika 88 Pitanje ankete 6

Sedmim pitanjem (Slika 89), fokus je na korisnost unosa nalaza, gdje su svi ispitanici istaknuli da smatraju unos nalaza izuzetno korisnim i dobrim. Ovaj rezultat ukazuje na ključnu vrijednost koju ova funkcionalnost pruža korisnicima, omogućujući im da lako prate i upravljaju svojim medicinskim nalazima.



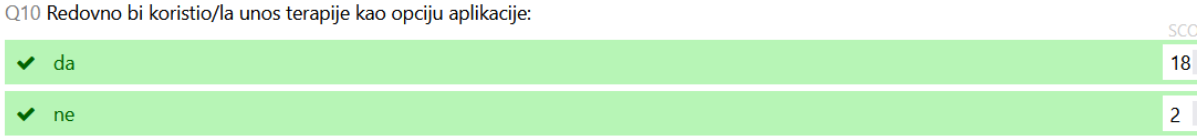
Slika 89 Pitanje ankete 7

Osmim pitanjem (Slika 90) istražena je učestalost unosa nalaza u aplikaciju. Svi ispitanici naveli su da bi uvijek unosili sve svoje nalaze u aplikaciju, što pokazuje visoku razinu povjerenja i integracije ove funkcionalnosti u njihove zdravstvene rutine.

Pitanje devet (Slika 90) se odnosi na unos terapija, gdje su svi ispitanici izrazili pozitivno mišljenje o korisnosti ove funkcionalnosti. To dodatno potvrđuje važnost ove funkcionalnosti za korisnike u praćenju i upravljanju njihovim terapijama.

Slika 90 Pitanje ankete 8 i 9

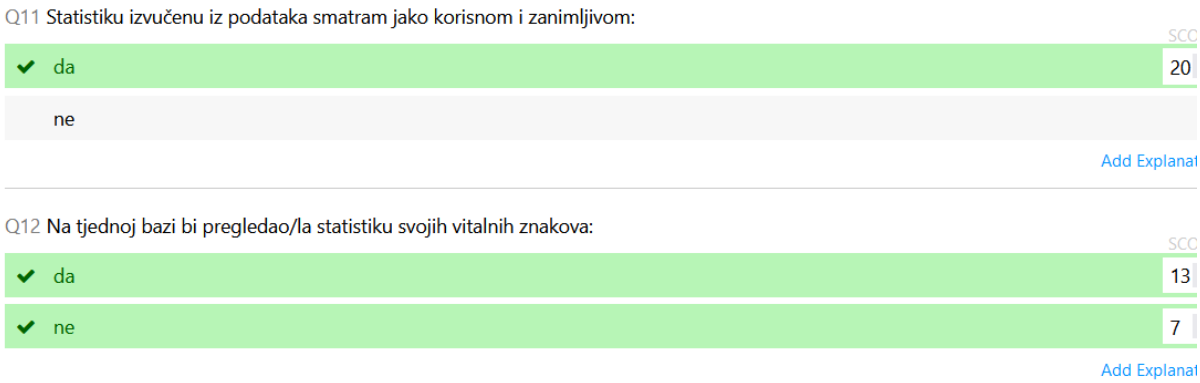
Desetim pitanjem (Slika 91) postavlja se pitanje o redovitom korištenju opcije unosa terapija, gdje većina ispitanika navodi da bi redovito koristili ovu opciju. Ovo sugerira da je funkcionalnost unosa terapija ne samo korisna, već i integralni dio korisničkog iskustva.



Slika 91 Pitanje ankete 10

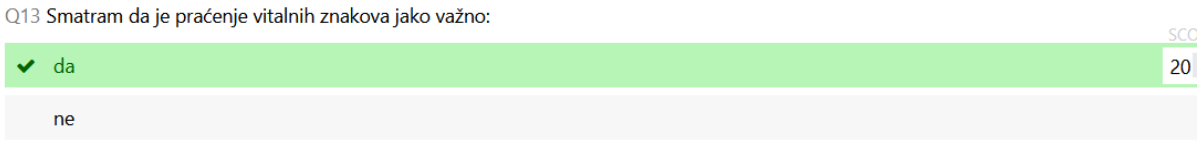
Jedanaesto pitanje (Slika 92) se bavi korisnošću statistike izvučene iz podataka, gdje su svi ispitanici izrazili pozitivno mišljenje. Ovo ukazuje na visoku vrijednost analitičkih alata koje aplikacija pruža, omogućavajući korisnicima da bolje razumiju svoje zdravstvene podatke.

Pitanje dvanaest (Slika 92) nadovezuje se na pitanje jedanaest te istražuje učestalost pregleda statistike vitalnih znakova, gdje većina ispitanika navodi da bi redovito pregledavali statistike na tjednoj bazi. Ovaj rezultat pokazuje koliko korisnici cijene mogućnost praćenja promjena u svojim vitalnim znakovima kroz vrijeme.



Slika 92 Pitanje ankete 11 i 12

Pitanje trinaest (Slika 93) se fokusira na važnost praćenja vitalnih znakova, gdje svi ispitanici smatraju da je praćenje ovih znakova jako važno. Ovo potvrđuje ključnu ulogu koju ova funkcionalnost igra u održavanju zdravlja korisnika.

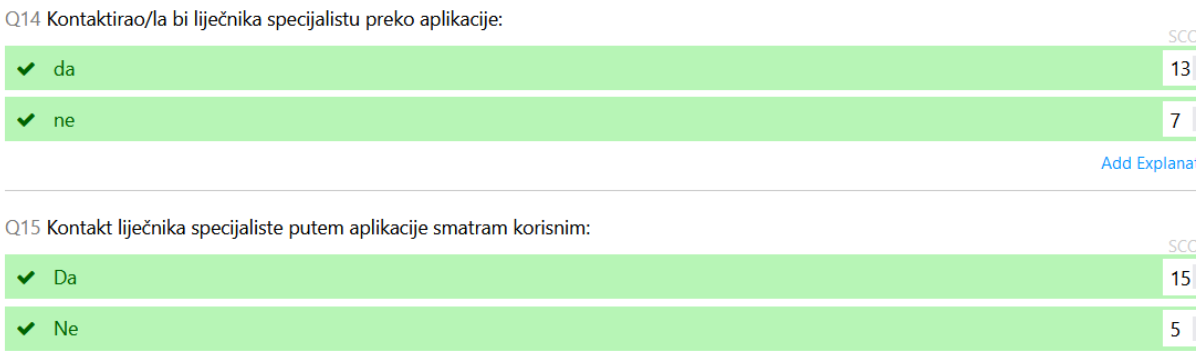


Slika 93 Pitanje ankete 13

Pitanje četrnaest (Slika 94) se odnosi na kontaktiranje liječnika specijalista putem aplikacije. Većina ispitanika navodi da bi kontaktirala specijalista preko aplikacije, što sugerira povjerenje u ovu funkcionalnost i potrebu za lako dostupnom medicinskom podrškom.

Pitanje petnaest (Slika 94) istražuje korisnost kontakta sa specijalistima putem aplikacije, gdje većina ispitanika smatra ovu funkcionalnost korisnom. Ovaj rezultat naglašava važnost integriranih komunikacijskih alata koji korisnicima omogućavaju pristup medicinskim savjetima i podršci.

Sveukupno, rezultati ankete pokazuju visoko zadovoljstvo korisnika s različitim aspektima aplikacije za vođenje osobnog zdravstvenog dnevnika, potvrđujući njenu korisnost, intuitivnost i vrijednost u podršci zdravlju korisnika.



Slika 94 Pitanje ankete 14 i 15

8. Zaključak

U suvremenom društvu, zdravlje je jedno od najvažnijih aspekata ljudskog života, a vođenje osobnog zdravstvenog dnevnika postaje sve značajnije za praćenje i održavanje zdravstvenog stanja. S razvojem tehnologije i povećanjem dostupnosti digitalnih alata, omogućeno je stvaranje aplikacija koje korisnicima olakšavaju upravljanje vlastitim zdravstvenim podacima. Ovaj rad detaljno prikazuje proces razvoja web aplikacije za vođenje osobnog zdravstvenog dnevnika.

Korištene su moderne tehnologije, uključujući JavaScript, HTML, CSS, Django i PostgreSQL kako bi se osigurala visoka funkcionalnost, sigurnost i kvalitetno korisničko iskustvo.

Implementacija dizajna u Figmi omogućila je detaljan i kolaborativan pristup, rezultirajući intuitivnim korisničkim sučeljem koje zadovoljava potrebe korisnika. Primjena smirujuće plave boje kao primarne boje aplikacije te različitih nijansi sive za tekst, osigurala je estetski ugodan i pregledan izgled. Sekundarne boje poput ljubičaste, zelene i crvene doprinose boljoj preglednosti i navigaciji kroz aplikaciju.

Korisničko iskustvo ove web aplikacije ocijenjeno je vrlo pozitivno prema rezultatima ankete. Većina korisnika smatra aplikaciju jednostavnom za korištenje i korisnom za praćenje njihovog zdravstvenog stanja. Posebno su pohvaljeni intuitivnost dizajna i funkcionalnosti kao što su unos i pregled medicinskih nalaza, praćenje vitalnih znakova, te evidencija lijekova i terapija. Vizualizacija zdravstvenih podataka kroz grafove dodatno je naglašena kao korisna značajka koja pomaže korisnicima u boljem razumijevanju i praćenju njihovog zdravlja.

Zaključno, rad potvrđuje da moderna web tehnologija može značajno doprinijeti razvoju funkcionalnih aplikacija koje pomažu korisnicima u upravljanju zdravstvenim podacima. Integracija analize i vizualizacije podataka u medicinske aplikacije pruža korisnicima vrijedne alate za prevenciju i praćenje kardiovaskularnih bolesti, istovremeno unapređujući njihovo cjelokupno zdravstveno iskustvo.

9. Pregled slika

10. Pregled literature

[1] Warrington L, Graetz I, i Snyder CF, „Remote symptom monitoring integrated into electronic health records: A systematic review. J Am Med Inform Assoc.“, 2016.

[2] Graetz I, Gordon N, i Fung V, „The association between utilization of the MyHealth record patient portal and health care utilization. J Am Med Inform Assoc. “, 2019.

[3] Austin L, Shetty S, i Zheng K, „Use of wearable devices for cardiovascular monitoring in a home-based setting. J Am Med Inform Assoc.“, 2016.

[4] Zylla D, Steele J, i Eklund J, „Medication adherence and interaction alerts in electronic health record systems. J Am Med Inform Assoc.“, 2016.

[5] Schougaard LMV, Thorsen T, i Brodersen J, „Use of personal health records for clinical communication and data sharing. J Am Med Inform Assoc.“, 2016.

[6] „JavaScript — Dynamic client-side scripting. MDN Web Docs.“, Dostupno na: https://developer.mozilla.org/en-US/docs/Learn/JavaScript (Pristupljeno 6.6.2024.).

[7] „HTML: HyperText Markup Language. MDN Web Docs.“, Dostupno na: https://developer.mozilla.org/en-US/docs/Web/HTML (Pristupljeno 6.6.2024.).

[8] „CSS media queries - CSS: Cascading Style Sheets | MDN.“, Dostupno na: https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_media_queries (Pristupljeno 6.6.2024.).

[9] „Figma: The collaborative interface design tool“, 2020, Dostupno na: https://www.figma.com/about/ (Pristupljeno 6.6.2024.).

[10] Taha Sufiyan, „What is Node.js? Simplilearn.“, Dostupno na: https://www.simplilearn.com/tutorials/nodejs-tutorial/what-is-nodejs (Pristupljeno 6.6.2024.).

[11] „Why Use MongoDB and When to Use It? MongoDB.“, Dostupno na: https://www.mongodb.com/resources/products/fundamentals/why-use-mongodb (Pristupljeno 6.6.2024.).

[12] Stephen J. Bigelow, „What is REST API (RESTful API)? TechTarget.“, 2024, Dostupno na: https://www.techtarget.com/searchapparchitecture/definition/RESTful-API (Pristupljeno 6.6.2024.).

[13] Rob Sobers, „What is OAuth? Definition and How it Works. Varonis.“, 2022, Dostupno na: https://www.varonis.com/blog/what-is-oauth (Pristupljeno 6.6.2024.).

[14] „Introducing asynchronous JavaScript“, Dostupno na: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Asynchronous/Introducing (Pristupljeno 10.6.2024.).

[15] „Meet Django“, Dostupno na: https://www.djangoproject.com/ (Pristupljeno 10.6.2024.).

[16] „What is PostgreSQL?“, Dostupno na: https://www.postgresql.org/about/ (Pristupljeno 10.6.2024.).

[17] Vivek Thuravupala, „Postman API Development Environment“, 2020, Dostupno na: https://www.postman.com/postman-galaxy/postman-api-development-platform-overview/ (Pristupljeno 11.6.2024.).

[18] Marijn Haverbeke, „Eloquent JavaScript, 3rd Edition: A Modern Introduction to Programming“, 2018.

[19] Aditya Yadav, „Hands-On Full-Stack Web Development with GraphQL and React“, 2020.

[20] „JavaScript Tutorial “, Dostupno na: https://www.w3schools.com/js/ (Pristupljeno 7.6.2024.).

[21] „CSS Tutorial “, Dostupno na: https://www.w3schools.com/css/ (Pristupljeno 7.6.2024.).

[22] „HTML Tutorial “, Dostupno na: https://www.w3schools.com/html/ (Pristupljeno 7.6.2024.).

[23] Ethan Brown, „Learning JavaScript: Add Sparkle and Life to Your Web Pages. “, 2019.

[24] Aidas Bendoraitis i Jake Kronika, „Django 3 Web Development Cookbook: Actionable solutions to common problems in Python web development, Fourth Edition “, 2020.

11. Prilozi

Prilog 1 aplikacija:
https://drive.google.com/file/d/1_WK7QrxBXSPMBn0j6kDxOPv5PCw29W5v/view?usp=sharing

Prilog 2 dizajn aplikacije:
https://drive.google.com/file/d/1h2oi9npwvH_ZejwCJ3FCtX5mbw9R5jVw/view?usp=sharing

Prilog 3 „sitemap“: https://drive.google.com/file/d/1BR7FiM0d-zUWMjv8_UqD56r6Pw57DPi8/view?usp=sharing