Izrada aplikacije za zdravstveni sustav korištenjem okvira Django

Kučina, Sven

Sveučilište u Rijeci – Fakultet informatike i digitalnih tehnologija

Modul – Informacijski i komunikacijski sustavi

Sven Kučina

Izrada aplikacije za zdravstveni sustav korištenjem okvira Django

Diplomski rad

Mentor: doc. dr. sc. Martina Ašenbrener Katić

Rijeka, rujan 2023.

Sveučilište u Rijeci
Fakultet informatike i digitalnih tehnologija
www.inf.uniri.hr

Rijeka, 12. lipnja 2023.

Zadatak za diplomski rad

Pristupnik: Sven Kučina

Naziv diplomskog rada: Izrada aplikacije za zdravstveni sustav korištenjem okvira Django

Naziv diplomskog rada na eng. jeziku: Development of a healthcare system application using the Django framework

Sadržaj zadatka:

Zadatak diplomskog rada je opisati postupak planiranja i stvaranja aplikacije za zdravstveni sustav čiji bi korisnici bili medicinsko osoblje i pacijenti. Aplikacija će imati različite funkcionalnosti kao što su evidencije podataka (o medicinskom osoblju, pacijentima i slično), mogućnost naručivanja pacijenata za određeni termin za pregled, upis simptoma bolesti pacijenata, uvid doktora u pregled termina svih pacijenata i slično. Procesu izrade aplikacije prethodit će analiza sustava, definiranje korisničkih zahtjeva, oblikovanje modela podataka i dizajn sučelja. Potrebno je aplikaciju izraditi koristeći okvir Django te opisati postupak izrade aplikacije.

Mentor:

Doc. dr. sc. Martina Ašenbrener Katić

Martina Ašenbrener Katić

Voditeljica za diplomske radove:

Prof. dr. sc. Ana Meštrović

Ana Mr.

Zadatak preuzet: 15. lipnja 2023.

(Sven Kučina)

Sven Kučina

Adresa: Radmile Matejčić 2
51000 Rijeka, Hrvatska
Tel: **+385(0)51 584 700**
E-mail: **ured@inf.uniri.hr**
OIB: **64218323816**
IBAN: **HR1524020061400006966**

Sadržaj

Sažetak 1
Uvod 2
Django 3
MVC 3
MVT 4
Analiza sustava 7
Logika aplikacije 7
Okruženje aplikacije 7
Korisnički zahtjevi 10
Kako napisati specifikaciju korisničkih zahtjeva 11
URS za Medigital aplikaciju u obliku korisničkih priča 11
Oblikovanje modela podataka 13
Model podataka za Medigital aplikaciju 15
Dizajn sučelja 17
Dizajn sučelja za Medigital aplikaciju 17
Razvoj aplikacije 21
Stvaranje aplikacije 21
URL konfiguracije 22
Modeli 23
Predlošci 28
Admin.py i admin sučelje 30
Pogledi 30
Upotreba aplikacije 35
Pacijent 35
Doktor 37
Administrator 38
Zaključak 39
Popis slika 40
Literatura 41

Sažetak

Cilj i zadatak ovog rada je opisati postupak planiranja i stvaranja web aplikacije za fiktivni zdravstveni sustav Medigital pomoću Django okvira. Procesu izrade aplikacije prethodit će analiza sustava, definiranje korisničkih zahtjeva, oblikovanje modela podataka i dizajn sučelja. Za dizajn sučelja koristit ću alat Figma. Nakon ove početne faze planiranja, krenut ću s izgradnjom aplikacije te opisivati korake i značajke. Aplikacija će imati razne funkcionalnosti poput evidencije podataka o medicinskom osoblju, pacijentima i slično. Pacijenti će se moći naručiti za pregled kod doktora i upisati simptome bolesti. Doktor će imati imat uvid u pregled svih termina njegovih pacijenata i simptoma pacijenata koje su sami pacijenti postavili. Doktor će zatim moći pregledavati, brisati, mijenjati i dodavati termine pacijenata. Nakon što pacijent odabere određeni termin, doktor će mu morati potvrditi taj termin. Od alata koristit ću HTML, CSS i CSS okvir bootstrap 4, SCSS, Django, Python i SQLite3 bazu podataka.

Ključne riječi: Django, web aplikacija, python, bootstrap 4, rute, modeli, SQLite3

Uvod

Django je Python web okvir visoke razine koji omogućava brzi razvoj web aplikacija i čisti, pragmatični dizajn. Dizajniran je tako da olakša razvoj web-aplikacija i pruža programerima mnoge ugrađene funkcionalnosti kako bi im olakšao posao. Django-va glavna karakteristika s kojom se iscrpno hvali na njegovoj web-stranici je njegova brzina što znači da pomoću Djnago web okvira programer može na vrlo brz i efikasan način izraditi web-aplikaciju od njenog konceptualnog početka do produkcijskog završetka. Osim toga, izuzetno je siguran i skalabilan.

Django koristi ORM (eng. *Object Relational Mapping*) koji pruža apstraktni sloj za komunikaciju s bazom podataka što omogućuje programerima rad sa bazom kroz objekte i metode umjesto da pišu SQL upite. Django ima unaprijed izrađeno administrativno sučelje koje omogućuje jednostavno upravljanje podacima u aplikaciji kao što su stvaranje, uređivanje i brisanje objekata.

Motivacija za izradom ovakve aplikacije leži u potrebi za korisnom aplikacijom koja će omogućiti pacijentima jednostavnije naručivanje i doktorima jednostavnije upravljanje narudžbi pacijenata.

Najprije ću započeti sa opisom Django okvira. Zatim ću nastaviti sa analizom sustava prilikom koje ću proučiti, razumjeti i opisati web aplikaciju koju ću kasnije razviti. Zatim ću definirati i opisati korisničke zahtjeve i podijeliti ih na funkcionalne (što sustav treba raditi) i nefunkcionalne (kako sustav treba raditi). Oblikovat ću model podataka i dizajnirati sučelje u alatu Figma. Naposlijetku, nakon navedenih koraka, razvit ću i opisat razvoj web aplikacije za zdravstveni sustav pomoću Django okvira.

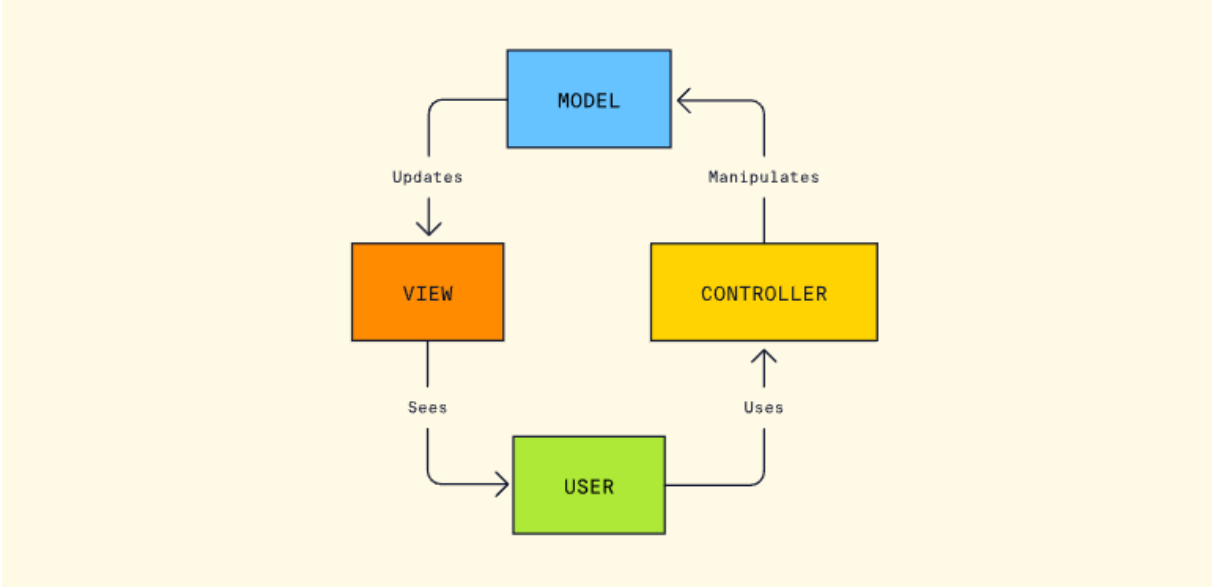
Django

Django je Python web okvir otvorenog izvora koji omogućava brz razvoj skalabilnih i sigurnih web aplikacija. Napravljen je tako da olakša razvoj web aplikacija i pruža programerima mnoge ugrađene funkcionalnosti kako bi im olakšao posao [1].

Koristi Model-View-Controller (MVC) arhitekturu, ali s malo drugačijim pristupom koji se naziva Model-View-Template (MVT). Objasnimo najprije kako MVC općenito funkcionira te zatim objasnimo Djangovu malo drugačiju paradigmu MVT [1].

MVC

MVC je skraćeno od *Model, View and Controller*. MVC je popularan način organiziranja koda. Ideja iza MVC paradigme je da svaka sekcija koda ima određenu svrhu koja je različita od drugih dijelova koda. Dakle, MVC uredno organizira kod u svoje prikladne kutije [1]. Slika 1 prikazuje MVC model: korisnik koristi upravljač koji manipulira modelom, a model ažurira pogled.



Slika 1 - MVC model

Opišimo dijelove MVC paradigme te što svaki od njih radi.

Model: Kod modela generalno predstavlja objekte u stvarnom svijetu. Praktično, to znači da možemo definirati model unutar koda koji će na primjer predstavljati doktore i model koji će predstavljati pacijente. Ovaj kod definira bitne i neophodne komponente aplikacije. Model predstavlja podatke i poslovnu logiku aplikacije. Odgovoran je za obradu podataka, pristupanje bazi podataka te dohvaćanje i spremanje podataka. Opisuje strukturu i ponašanje podataka koji se koriste u aplikaciji, a to uključuje definiranje klasa i metoda koje omogućuju manipulaciju podacima, provođenje poslovne logike te validaciju unosa i ostale operacije povezane s podacima. Model sadrži samo čiste aplikacijske podatke te ne sadrži logiku koja predstavlja te podatke korisniku. Model zatim pruža podatke koje **pogled (eng. *view*)** koristi za prikaz korisničkom sučelju [1].

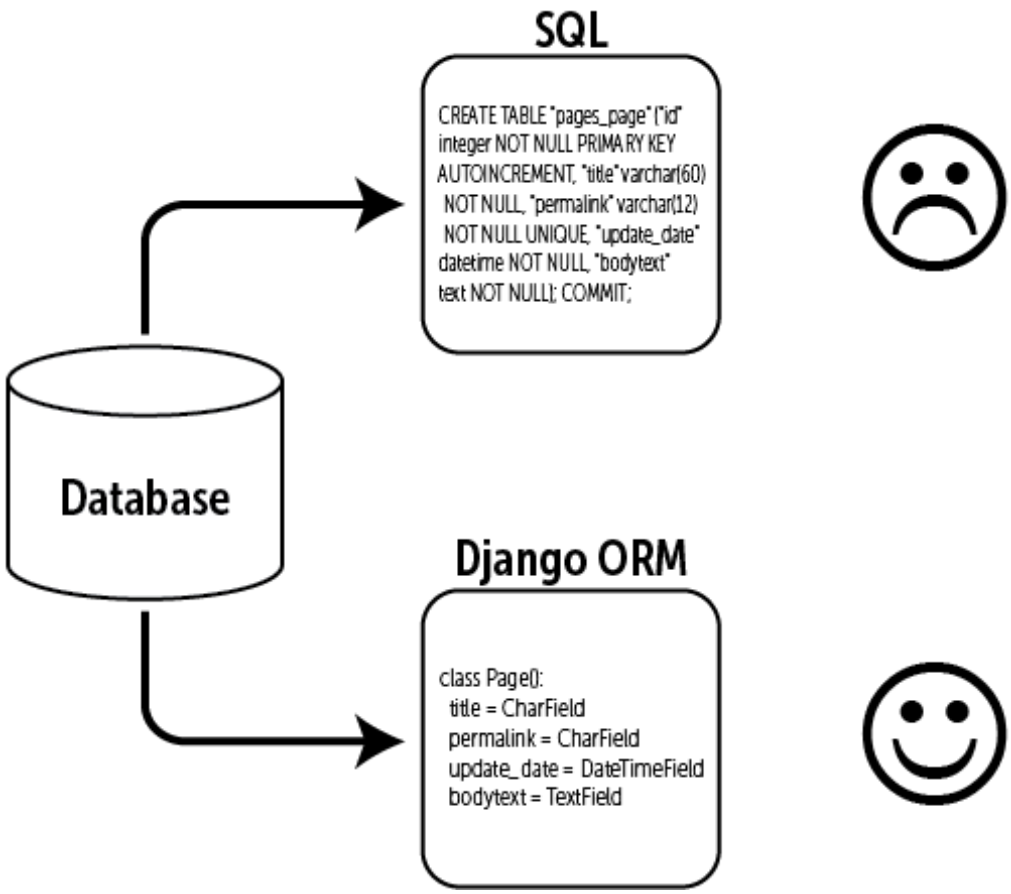
Pogled (eng. *view*) prezentira podatke modela korisniku. Pogled zna kako pristupiti podacima modela, ali ne zna što ti podaci znače, predstavljaju i što korisnik može učiniti kako bi upravljao tim podacima. Datoteke pogleda su obično tipa .html formata [1].

Upravljač (eng. *controller*) postoji između pogleda i modela. U upravljaču je zapisana poslovna logika. Upravljač sluša na događaje koji se pokreću o pogledu te izvršava prikladnu reakcije na takve događaje. U većini slučajeva, reakcija upravljača je pozivanje određene metode na modelu. Budući da su model i pogled povezani putem mehanizma obavješćivanja, pogled može odmah prikazati rezultat [2].

MVT

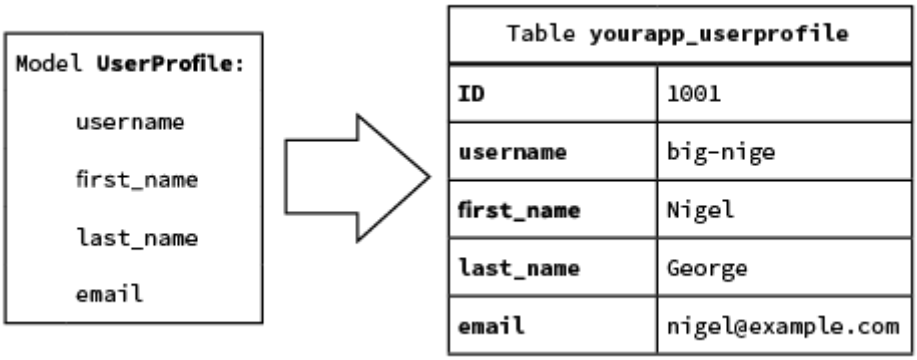
Django koristi paradigmu MVT (eng. *Model-View-Template*). Svaka stranica kreirana pomoću Django okvira koristi ove tri stvari za prikaz podataka. Djangov MVT radi u obrnutom slijedu od MVC. Korisnik će posjetiti URL te će zatim naš upravljač ukazati na određeni prikaz. Taj prikaz može, ali i ne mora biti povezan sa drugim modelima. Dakle prvo ide upravljač, zatim view i naposlijetku model što se razlikuje od MVC paradigme gdje prvo ide upravljač, zatim upravljač manipulira modelom i na kraju pogled prikazuje podatke koji vuče iz modela. Glavna razlika kod Djangove paradigme je ta da model nije potreban da bi se nešto prikazalo korisniku, već model dolazi na samom kraju [3].

Model: Sloj modela u Djangu se odnosi na bazu podataka. Modelira stvarnost te pohranjuje sve potrebne podatke unutar baze podataka koje su potrebne web aplikaciji. Djangovi modeli omogućavaju **Objektno-Relacijsko-Preslikavanje (eng. *Object-Relational-Mapping*)** ili kraće **ORM** nad bazom podataka. Kao što je rečeno, ORM je moćna programerska tehnika koja olakšava rad sa relacijskih bazama podataka te ne zahtjeva od programera da poznaje SQL (eng. *Structured Query Language*) upite. Dakle, ORM alat pruža preslikavanje između objekta i baze podataka što prikazuje slika 2 [3].



Slika 2 - Django ORM

U Djangu, model je objekt koji se preslikava u pripradajuću tabilcu u bazi podataka bez da programer mora utipkati ijednu liniju SQL koda. Django stavlja naziv Django aplikacije kao prefiks naziva tablice [4].



Slika 3 - Preslikavanje modela u tablicu

Template: Django template je datoteka namijenjena odvajanju aplikacijskih podataka od načina na koji su ti podaci prezentirani korisniku. Django template-i su HTML (eng. *Hypertext Markup Language*) formata, ali nisu ograničeni na HTML. Django template-i su bazirani na

temelju tri glavna principa:

1. Template sustav treba odvojiti logiku programa od dizajna
2. Template-i trebaju smanjiti redundanciju koda – DRY (eng. *Don't Repeat Yourself*)
3. Template sustav treba biti siguran – zabranjeno izvršenje koda u template-ima [4]

Web dizajn i web programiranje su dvije različite vještine. Uglavnom, u većim korporacijama, web dizajn i web programiranje su odvojeni te se dizajneri sučelja bave web dizajnom dok se programeri bave web programiranjem. Ponekad se čak različite korporacije bave web dizajnom dok se druge bave web programiranjem. Kada je Django bio napravljen, kreatori Djanga su imali na umu činjenicu da programeri i dizajneri moraju moći neovisno raditi. Rezultat toga je Django template jezik koji koristi **oznake (eng. *tags*)** kako bi pružio prezentacijsku logiku te znao koje podatke će prikazati unutar template-a [4].

Jedan od Djangovih glavnih principa je DRY (eng. *Don't Repeat Yourself*) koji teži da smanji redundanciju koda. Takav princip je realiziran upravo putem Django template sustava koji omogućava nasljeđivanje template-a (eng. *template inheritance*). Uzmimo za primjer neku općenitu web stranicu koja ima zaglavlje (eng. *header*), podnožje (eng. *footer*) i navigaciju (eng. *navigation*). Navedena tri elementa će se pojavljivati na svim kategorijama naše web stranice. Django template sustav omogućuje programeru da napravi roditelj-template koji sadrži kod koji je jednak svim kategorijama web stranice. Programer zatim može napraviti dijete-template koji će naslijediti takve jednake značajke bez nepotrebnog kopiranja redundantnog odnosno šablonskog koda (eng. *boilerplate code*) [4].

Pogled (eng. *view*): Središnji dio sustava koji sadrži logiku koja povezuje cjeline kako bi se pružio odgovor korisniku na traženi zahtjev. Upravlja zahtjevima i odgovorima na njih te uspostavlja vezu sa bazom podataka. Imajmo na umu da je Djangov pogled unutar MTV paradigme korespondentan sa upravljačem (eng. *controller*) unutar MVC paradigme. Django pogled dohvaća podatke iz baze podataka i dostavlja ih unutar template-a. Pogled odlučuje koji podaci će biti dostavljeni template-u na način da reagira na unose korisnika ili reagira na unutarnje procese. Svaki Django pogled obavlja specifičnu funkciju i ima odgovarajući template [4].

Django koristi tehniku zvanu Objektno Relacijsko Preslikavanje (eng. *Object Relational Mapping*) koja stvara „most“ između objektno orijentiranih programa i relacijskih baza podataka. ORM je zapravo sloj koji povezuje objektno orijentirano programiranje (OOP) sa relacijskom bazom. Prilikom interakcije sa bazom podataka, programer mora izvršiti niz različitih operacija kao što su CRUD operacije – stvori, čitaj, ažuriraj i izbriši (eng. *create, read, update, delete*). ORM alat olakšava programerima interakciju sa bazom podataka na način da ne moraju pisati SQL upite za komunikaciju s bazom podataka, već komuniciraju s njom putem objekta i modela [4].

Analiza sustava

Analiza sustava je proces razumijevanja, evaluacije i modeliranja složenih sustava kako bi se bolje razumjela njihova struktura, funkcionalnost, performanse i ponašanje [22].

Analiza sustava se bavi vanjskim ponašanjem softverskog sustava odnosno ono što korisnik vidi. Analiza se ne bavi dizajnom ni implementacijom [22].

Logika aplikacije

Objasnimo logiku i tok aplikacije. U ovo web aplikaciji za zdravstveni sustav postoje tri vrste korisnika: doktor, pacijent i administrator. Doktor ima svoj vlastiti profil te se prijavljuje pomoću korisničkog imena i lozinke u aplikaciju. Nakon prijavljivanja, doktoru se prikazuje dodatna kategorija koja je nevidljiva običnom korisniku (pacijentu) zvana 'Moji termini'. Unutar te kategorije, doktor može pregledati sve narudžbe pacijenata kod njega, opise tih narudžbi, datum pregleda i vrijeme pregleda te simptome koji su njegovi pacijenti naveli.

Pacijent može napraviti narudžbu i poslati je. Unutar narudžbe pacijent navodi svoje ime, prezime, telefon, email, datum i vrijeme pregleda, odabir željenog doktora za pregled i simptome bolesti.

Administrator je taj koji dodaje doktore, medicinske sestre i domove zdravlja u bazu podataka. On može mijenjati, brisati i dodavati sve entitete unutar baze. Ako na primjer jedan doktor da otkaz, administrator sustava ga vrlo lako može izbrisati. U odnosu o tome što administrator doda ili obriše, to će se dinamički prikazati na web sjedištu.

Okruženje aplikacije

Aplikaciju mogu koristiti tri vrste korisnika: pacijenti, doktori i administrator. Doktor ima mogućnost prijavljivanja i pregledavanja pacijenata i narudžbi dok pacijenti imaju mogućnost naručivanja na pregled u odabrano vrijeme.

Aplikacija je napravljena u Django web okviru koji koristi Python kao programski jezik. Osim toga, Django nudi svoj posebni sustav za izradu dinamičkih HTML-ova te ostalih tekstualnih datoteka zvan Django sustav predložaka ili Djangov jezik za predloške (eng. *Django template language*). Django sustav predložaka omogućuje programerima da kombiniraju HTML sa posebnim oznakama i kako bi se generirao dinamički sadržaj. Omogućuje nam da ubacimo varijablu, iteriramo kroz liste, koristimo uvjete i logiku i još mnogo toga, sve unutar HTML predložaka [17].

Aplikacija za zdravstveni sustav je napravljena kao jedan projekt u Django okviru unutar kojeg postoje dvije aplikacije: *users* i *zdravstveni_sustav_app*. Objasnimo detaljnije. Django razlikuje projekt i aplikaciju. Jedan projekt može imati više aplikacija, a jedna aplikacija se može koristiti u više projekata. Na ovaj način Django omogućuje modularnost te ako nam se na primjer svidi naša aplikacija *users* možemo je koristiti i u nekom drugom projektu. Svaki Django projekt sadrži datoteku *settings.py* unutar koje možemo dodavati aplikacije pod *INSTALLED_APPS* [23]:

```
INSTALLED_APPS = [
    'zdravstveni_sustav_app.apps.ZdravstveniSustavAppConfig',
    'users.apps.UsersConfig',
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
]
```

Dakle, kao što vidimo, u listu instaliranih aplikacija smo dodali aplikacije *users* i *zdravstveni_sustav_app*. Svrha aplikacije *users* je da omogući korisnicima da prijavi. Svrha aplikacije *zdravstveni_sustav_app* čini zapravo glavninu aplikacije. U njoj su definirani modeli, pogledi, predlošci i tako dalje [23].

Aplikacija je napravljena nad SQLite bazom podataka te je također definirano unutar *settings.py* datoteke:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
}
```

Naravno, baza podataka se može mijenjati ovisno o potrebama. Za naše potrebe SQLite baza podataka će biti sasvim dovoljna.

Django admin sučelje je jedna od najmoćnijih alata koju Django okvir pruža. Čita metapodatke iz modela kako bi brzo pružao sučelje gdje autorizirani korisnici mogu manipulirati podacima. Olakšava administraciju i upravljanje Django web aplikacijom i pruža administratorski pristup backendu putem web preglednika na taj način omogućavajući uređivanje, dodavanje, brisanje i izmjenjivanje podataka u aplikaciji na jednostavan način. Automatski generira administratorsko sučelje na osnovu definicija modela u aplikaciji. Django admin sučelje također omogućava kontroliranje pristupa korisnika i korisničkih grupa te određivanje privilegija korisnicima nad podacima. Dakle, može se omogućiti da samo određene osobe imaju prava upravljanjem sadržajem. Ako nismo zadovoljni izgledom sučelja, Django također nudi promjenu izgleda administratorskog sučelja te čak i dodavanje novih funkcionalnosti. Povrh toga, Django admin sučelje ima ugrađene mehanizme zaštite te se može dodatno pojačati sigurnost po potrebi [19].

Za uređivanje i stiliziranje web aplikacije se koristio Bootstrap, besplatan CSS okvir otvorenog koda koji olakšava razvoj web stranica i web aplikacija. Bootstrap pruža kolekciju predodređenih stilova i komponenata koje se mogu integrirati u web aplikaciju. Glavni cilj Bootstrapa je ubrzavanje procesa izrade web stranica i pružanje atraktivnog i dosljednog dizajna.

Korisnički zahtjevi

Kako bi se osiguralo da bilo koji proizvod zadovoljava stvarne potrebe korisnika, trebaju se odrediti i dokumentirati točni zahtjevi koje korisnik ima od tog proizvoda. Poznavanje onoga što korisnik treba presudna je za uspjeh svakog projekta. Za korisničke zahtjeve nije bitno kako će ih se ispuniti, već koji su oni. Specifikacija korisničkih zahtjeva ne smije biti previše komplicirana, nejasna ili previše tehnička. Rezultat specifikacije korisničkih zahtjeva je dokument specifikacije korisničkih zahtjeva (eng. *user requirements specification* - URS) koji opisuje što aplikacija odnosno sustav mora učiniti, ali ne i kako. Dobro izrađen dokument specifikacije korisničkih zahtjeva je jasan, čitak, dobro objašnjen i sažet [5].

Korisnički zahtjev je uvjet ili svojstvo koje korisnik treba kako bi riješio problem ili postigao cilj. Adekvatnim definiranjem korisničkih zahtjeva u ranijim fazama razvoja softvera se smanjuje rizik od neprihvaćanja finalnog proizvoda. U slučaju da korisnički zahtjevi nisu jasni, potpuni ili točni veća je vjerojatnost da projekt neće biti ostvaren unutar željenog proračuna [5].

Preduvjet za uspješan projekt i finalni proizvod koji ispunjava stvarne potrebe korisnika jesu dobro i jasno definirani zahtjevi korisnika. Kada su evidentirani predstavljaju ugovor između klijenta-naručitelja i pružatelja usluga te osiguravaju da obje strane rade na postizanju istog cilja. Sustav treba korisnicima pružati ono što oni zahtijevaju i traže. Tako precizno i jasno definirani zahtjevi umanjuju rizike od financijskih i vremenskih troškova na projektu [5].

Ovisno o tipu, zahtjevima i vrsti projekta, specifikacija korisničkih zahtjeva će uključivati različite informacije. Složeniji projekti će naravno imati više zahtjeva nego jednostavniji. Postoje ipak temeljni principi i važne značajke koje čine dobru praksu za većinu projekta bez obzira na veličinu projekta [5].

Prvi dio svake specifikacije korisničkih zahtjeva trebao bi biti nabrajanje autora odgovornih za izradu specifikacije. To bi trebalo uključivati ime, naziv radnog mjesta, datum i potpis svih koji su kokreirali specifikaciju korisničkih zahtjeva [5].

Specifikacija korisničkih zahtjeva bi nadalje trebala uključivati uvod koji vrlo ukratko opisuje pozadinu tvrtke koja razvija softverski sustav i razlog zašto tvrtka želi razviti određeni softver [5].

Povrh toga, nakon uvoda, potrebno je jasno i ukratko u ne-tehničkom žargonu opisati cilj projekta. Ovaj dio potrebno je napisati u narativno i deskriptivnom stilu te dati ukratko sliku cijelog projekta koristeći prirodni jezik. Za provedbu ovog dijela specifikacije korisničkih zahtjeva potrebno je postaviti sljedeća pitanja:

- Što želimo postići?
- Koje probleme će proizvod riješiti?
- Kako ćemo napraviti trenutni sustav efikasnijim?

Najvažniji dio URS su korisnički zahtjevi. Korisnički zahtjevi mogu biti opisani na različite načine. Svaki zahtjev se treba odraditi sam za sebe te precizno opisati. Dakle kao što smo rekli,

bitno je da se korisnički zahtjevi dokumentiraju u narativno i deskriptivnom stilu, bez tehničkog žargona, listi i sličnih obilježja. Također je bitno opisati što će proizvod raditi, a ne kako će to proizvesti. Dobra praksa je i za svaki zahtjev navesti ako je visoki, srednji ili niski prioritet te jedinstveno identificirati svaki zahtjev [5]. Pitanja koja si možemo postaviti su

- Što očekujemo?
- Što želimo da se dogodi u određenom trenutku?
- Što različiti korisnici trebaju vidjeti. Na primjer, *'Kao administrator, očekujem ...'*

Kako napisati specifikaciju korisničkih zahtjeva

Za najbolji način opisivanja korisničkih zahtjeva koristimo SMART obilježja. SMART je kratica od *Specific, Measurable, Achievable, Realistic* i *Time-bound* što bi u direktom prijevodu značilo Specifičnost, Mjerljivost, Izvedivost, Realističnost i Vremenska osviještenost. SMART obilježja pružaju odličan način da si osiguramo da su naše specifikacije korisničkih zahtjeva dobro određene i provjerljive [6].

Specifičnost: Specifikacija korisničkih zahtjeva mora biti jasna i specifična. Na primjer nećemo nejasno reći kao zahtjev da je potrebno 'poboljšati latenciju reklama', već ćemo ovako jasnije opisati 'poboljšati latenciju reklama za 50%' [6].

Mjerljivost: Da bi određeni zahtjev bio mjerljiv mora ga se moći potvrditi testiranje ili demonstracije. Potrebno je izbjegavati subjektivne pridjeve poput 'brže' ili 'jednostavnije'. Ako zahtjev nije mjerljiv ne postoji način da se utvrdi postignuće određenog zahtjeva budući da će on ovisiti o subjektivnoj interpretaciji. Dakle, jasnost i objektivnost je bitna [6].

Izvedivost: Cilja mora biti tehnički izvediv. U slučaju nesigurnosti oko tehničke izvedivosti zahtjeva potrebno je nadalje istraživanje i proučavanje tog zahtjeva, funkcionalnosti i problema. Ako i dalje nismo sigurni u izvedivost određenog zahtjeva, a potrebno je izraditi URS, koristimo riječi poput 'cilj' umjesto 'zahtjev' [6].

Realističnost: Čak i u slučaju da je zahtjev tehnički izvediv ne znači da je realističan (na primjer zbog proračunskih ograničenja, vremenskih ograničenja ili ostalih ograničenja). Potrebno je biti realističan prilikom specifikacije korisničkih zahtjeva [6].

Vremenska osviještenost: Opisuje koji je vremenski okvir odnosno vremenski rok za projekt [6].

URS za Medigital aplikaciju u obliku korisničkih priča

U nastavku su nabrojane specifikacije korisničkih zahtjeva u obliku korisničkih priča za aplikaciju za zdravstveni sustav Medigital:

„Kao administrator sustava, želim moći dodavati i brisati doktore iz sustava kako bi aplikacija bila ažurirana te odgovarala stvarnoj poslovnoj situaciji u tvrtki Medigital.“

„Kao administrator sustava, želim moći mijenjati podatke o doktorima, medicinskim sestrama, domovima zdravlja i narudžbama kako bih osigurao da su u skladu sa stvarnom situacijom tvrtke Medigital.“

„Kao pacijent, želim se moći naručiti kod doktora kako bih mogao doći na pregled.“

„Kao pacijent, želim odabrati datum pregleda kako bih znao doći na pregled na dan kada nemam ostalih obaveza.“

„Kao pacijent, želim odabrati vrijeme pregleda kako bih mogao doći na pregled kada mi najviše odgovara.“

„Kao pacijent, želim odabrati doktora koji će me pregledati kako bih si osigurao najbolje iskustvo.“

„Kao pacijent, želim moći upisati simptome i razloge svojeg naručivanja kako bih što bolje opisao svoju situaciju medicinskom stručnjaku.“

„Kao doktor, želim imati vlastiti korisnički profil kako bih se mogao prijaviti u sustav i vidjeti podatke koji su za mene relevantni.“

„Kao doktor, želim moći pregledati sve narudžbe i termine pacijenata koji su se naručili kod mene kako bih dobio što više informacija o narudžbi.“

„Kao doktor, želim moći vidjeti simptome koje su pacijenti naručeni kod mene naveli, kako bih što bolje mogao izvršiti medicinsku dijagnozu.“

Oblikovanje modela podataka

Model podataka je skup entiteta i operacija nad njima. Modeliranje podataka je proces stvaranje pojednostavljenog dijagrama softverskog sustava i podatke koje taj sustav sadrži koristeći tekst i simbole za reprezentaciju podataka i njihovih veza. Modeli podataka pružaju šablonu za izradu nove baze podataka ili preinaku postojeće. Modeliranje podataka pomaže organizaciji u korištenju svojih podataka na efikasan način kako bi ispunila poslovne potrebe [8].

Danas postoje različite metode za modeliranje podataka. U ovom radu, za modeliranje podataka koristit ćemo se metodom modeliranja entieti-veze koju je razvio Peter Chen 1976. godine. Ukratko ćemo opisati najvažnije definicije i simbole koji se koriste u Chenovim dijagramima kako bi bolje razumjeli naš model podataka za Medigital aplikaciju. Objasnit ćemo samo one simbole koji su korišteni za izradu aplikacije [7].

Tip entiteta predstavlja skup objekata s istim atributima unutar modela podataka. Tip entiteta predstavlja objekt u stvarnome svijetu ili koncept, poput osobe, automobila, koncepta i tako dalje [7]. Simboliziran je pravokutnik sa nazivom tipa entiteta unutar njega što možemo vidjeti na slici 4.

Slika 4 - Tip entiteta

Atribut tipa entiteta sadrži informaciju koja opisuje entitet. Pruža više detalja o entitetu i izražava njegove karakteristike. Slika 5 prikazuje atribut koji je prikazan kao elipsa. Postoji više oblika atributa, ali za naše potrebe su važna dva: obični atribut i atribut-ključ [7][8].

Slika 5 – Atribut

Ključ tipa entiteta je skup atributa koji zadovoljava uvjet jedinstvenosti (ne postoje dva ista takva atributa) i uvjet neredundantnosti (ako se iz ključa izostavi jedan atribut u skupu atributa onda ključ gubi svoju osobinu ključa). Služi kao primarni ključ za tip entiteta. Prikazan je unutar dijagrama kao oval sa podcrtanim naziv unutar njega [9]. Slika 6 prikazuje ključni atribut.

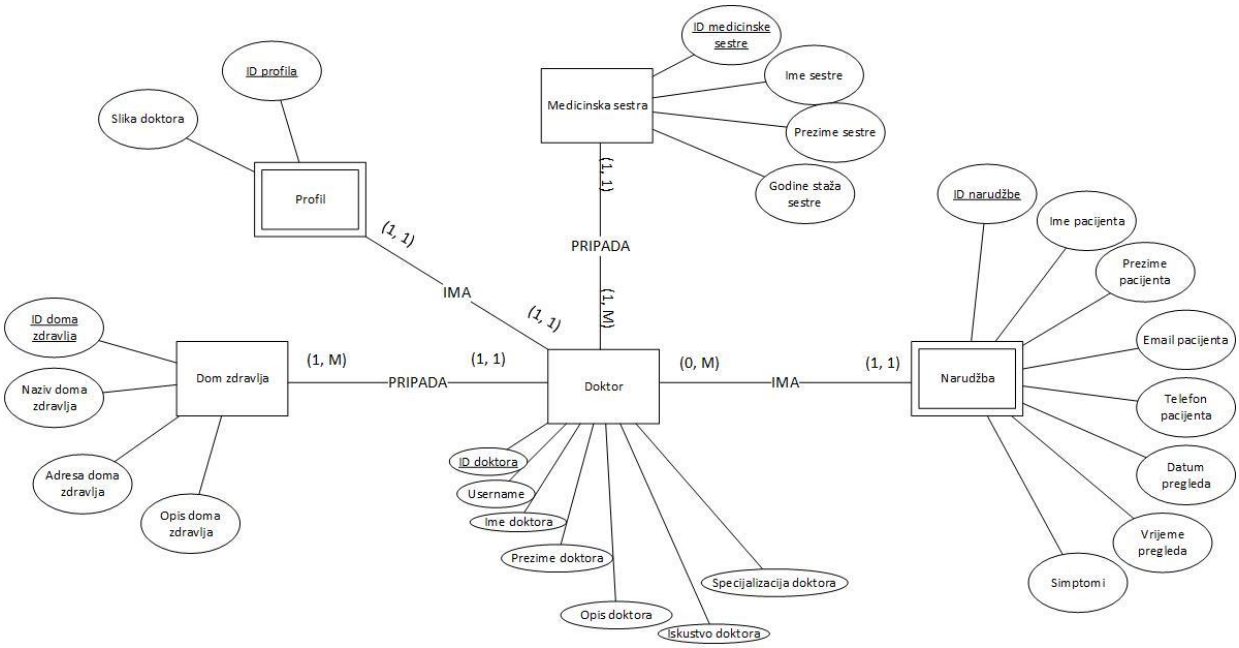
Slika 6 - Ključni atribut

Veza određuje međupovezanost između dva tipa entiteta te je prikazana kao romb ili pravac [9]. Slika 7 prikazuje vezu.

Slika 7 – Veza

Model podataka za Medigital aplikaciju

U nastavku je prikazan model podataka napravljen po modelu entiteti-veze (EV) od Chena.



Slika 8 - Model podataka

Kao što vidimo, u modelu podataka imamo pet tipa entiteta, dva od kojih su slabi tipovi entiteta. Slabi tip entitet ima egzistencijalnu ovisnost o odgovarajućem jakom tipu entiteta što znači da ako se izbriše jaki tip entiteta iz baze podataka, izbrisat će se i slabi tip entiteta. Opišimo u nastavku svaki od navedenih tipova entiteta.

Doktor je centralni, jaki tip entiteta koji ima sljedeće atribute: ID doktora, Username, Ime doktora, Prezime doktora, Opis doktora, Iskustvo doktora, Specijalizacija doktora. Doktor može imati nula ili više narudžbi. Ako se iz baze podataka izbriše entitet tipa doktor, briše se i entitet tipa narudžba. Doktor može imati jednu ili više medicinskih sestara koje su mu opredijeljene. Doktor ima jedan i samo jedan profil. Ako se iz baze podataka izbriše tip entiteta doktor izbrisat će se i tip entiteta profil.

Jedna narudžba može pripadati samo jednom doktoru (ne može biti jedna narudžba kod dva različita doktora).

Dom zdravlja može imati jednog ili više doktora koji rade u njemu. Doktor nije slab tip entiteta u odnosu na dom zdravlja zato jer ako se dom zdravlja na primjer ruši ne znači da će određeni doktori prestati biti zaposleni, već će jednostavno promijeniti dom zdravlja [10].

Profil pripada jednom i samo jednom doktoru. Svaki doktor ima dodijeljen svoj profil prilikom registracije. Ako se obriše doktor briše se i profil jer nema smisla da profil postoji bez doktora.

Medicinska sestra može pripadati samo jednom doktoru u trenutku vremena. Medicinska sestra nije slabi tip entiteta jer ako doktor na primjer da otkaz ne znači da će medicinska sestra prestati raditi, već će promijeniti doktora.

Dizajn sučelja

Pri izradi programskog proizvoda kreće se od ideje koju najčešće predstavljaju klijenti. Klijent navodi svoje zahtjeve, potrebe i funkcionalnosti koje želi da ih programerski tim implementira. Naime, osim funkcionalnosti koje će zadovoljiti klijenta i osigurati dobar proizvod, treba postojati i atraktivan dizajn koji će poboljšati iskustvo korisnika [5].

Korisničko sučelje je ono koje se vidi i čuje, dok su ostali dijelovi sustava skriveni. Korisnik ostvaruje interakciju s računalom upravo preko korisničkog sučelja. Korisničko sučelje treba planirati prije razvoja samog programskog proizvoda kako bismo smanjili rizik da finalni proizvod ne odgovara korisniku [5].

Prilikom dizajna korisničkog sučelja, možemo se koristiti raznim metodama. Najpopularnije su: *wireframe, mockup* i *prototip.*

Wireframe daje grubi prikaz web stranice ili aplikacije. Ne vodi se briga o atrakciji dizajna, već se ugrubo daje prikaz potrebnih elemenata na ekranu. Cilj je predstaviti glavni sadržaj stranice, način na koji su elementi raspoređeni i strukturu stranice. Ovakav način je brz i jeftin. Može se koristiti čak običan papira i olovka iako postoje i razni računalni programi koji bi olakšali izradu wireframe-a [5].

Mockup je nacrt koji prikazuje više vizualnih detalja poput boja, tipografije i slično. Mockup je vrlo zanimljiv korisniku budući da mu pokazuje izgled njegove buduće aplikacije. Prikazuje vizualne elemente, boje, gumbove, grafiku, tekstove, razmak među elementima i slično. Ovakva skica aplikacije se uglavnom izrađuje nakon prikupljanja i analize korisničkih zahtjeva [12].

Prototip donosi osnovne funkcionalnosti te korisnici mogu osjetiti kako će koristiti buduću aplikaciju. Realistično prikazuje budući proizvod, ponaša se jednako kako će se ponašati i buduća web stranica ili aplikacija i zato se na njemu može provesti testiranje uporabljivosti [5].

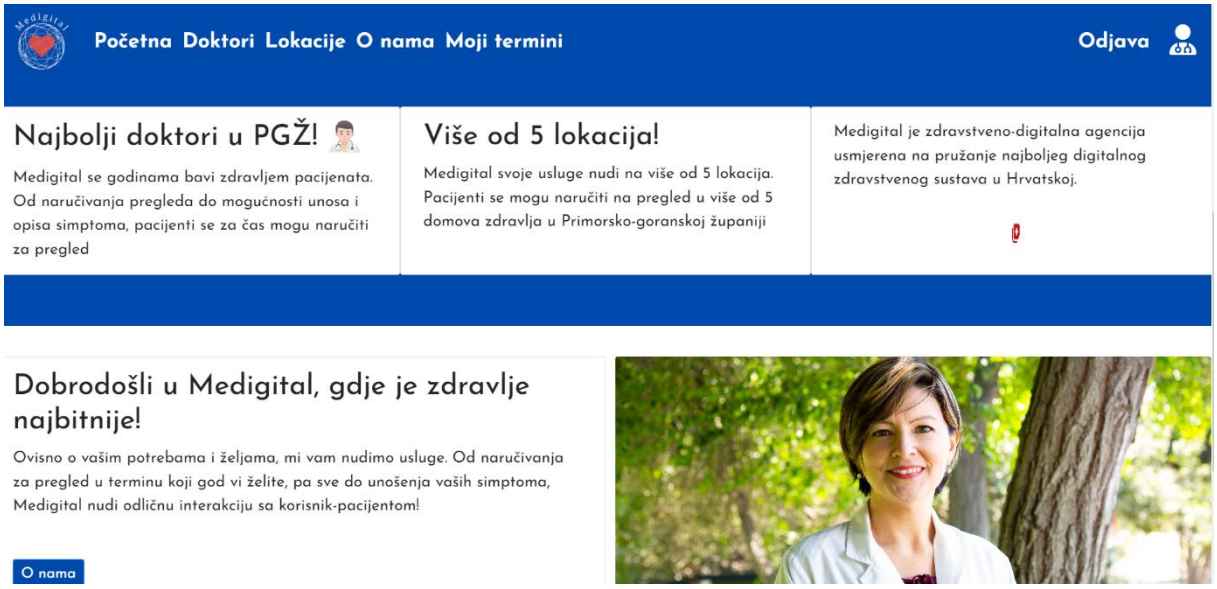
Dizajn sučelja za Medigital aplikaciju

Za izradu dizajna sučelja koristio sam alat **Figma**. Figma je alat za dizajn koristiv na oblaku. Figma funkcionira na bilo kojoj platformu. Može se koristiti na bilo kojem operativno sustavu koji ima instaliran web preglednik. To su dakle MacOS, Windows, i Linux računala [11].

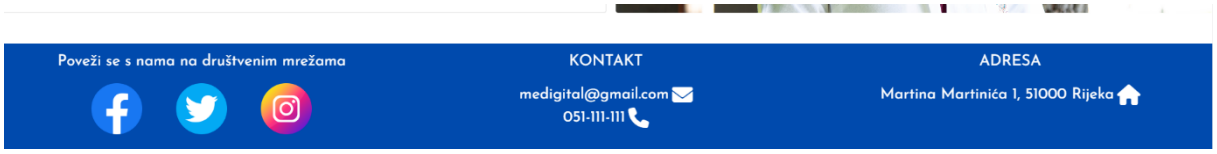
Za samo kodiranje i implementaciju web dizajna i izradu web stranica koristio sam HTML, CSS, CSS okvir Bootstrap 4 i nešto malo SCSS-a (engl. *Sassy Cascading Style Sheets*).

Pogledajmo sada našu **početnu** stranicu. Slika 9 prikazuje prvi dio početne stranice, slika 10 prikazuje drugi dio početne stranice a slika 11 prikazuje podnožje.

Slika 9 - Početna 1



Slika 10 - Početna 2



Slika 11 – Footer

Trudio sam se da boje i izgled budu u skladu sa dobrim web dizajnom. Logo je napravljen pomoću moćnog i besplatnog alata za brzu i jednostavnu izradu logotipa zvan Canva. Sa lijeve

strane nalazi se navigacija i logo a sa desne strane gumb za prijavu ili odjavu doktora. Ostatak stranice je napravljen pomoću Bootstrap 4 CSS okvira za razvoj web stranica. Koristio sam Bootstrap kartice i Bootstrap grid sustav koji su vrlo moći Bootstrap alati za izradu web stranica. Bootstrap ima CSS klasu *container-fluid* koja omogućuje da se određeni element responzivno rastegne preko cijelog ekrana. Cijelo web sjedište je napravljeno responzivno te se adekvatno prikazuje na svim veličinama ekrana. Pogledajmo primjer za mobilni prikaz na slici 12 i 13.

Slika 12 - Početna 1 mobile

Slika 13 - Početna 2 mobile

Vidimo da se stranice ponašaju responzivno. Bootstrap kartice zauzimaju širinu cijelog prozora na manjim prikazima poput mobilnog te se preklapaju jedna preko druge, zauzimajući svih 12 stupaca Bootstrap mrežnog sustava.

Razvoj aplikacije

Stvaranje aplikacije

Krenimo od početka. Najprije je potrebno instalirati Django i provjeriti ako je instaliran sa sljedećim naredbama [13]:

```
python -m pip install Django
python -m django --version
```

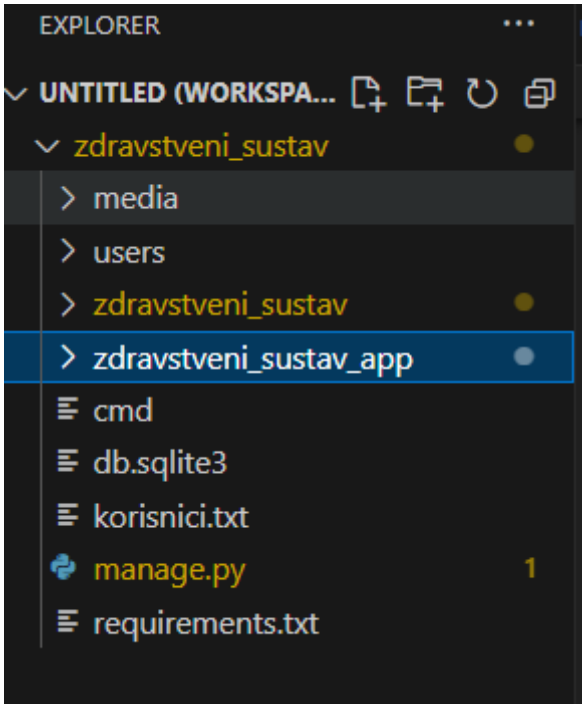
Zatim pokrećemo naš projekt:

```
django-admin startproject zdravstveni_sustav
```

Naposlijetku našu aplikaciju kreiramo i pokrećemo lokalni poslužitelj:

```
python manage.py startapp zdravstveni_sustav_app
python manage.py runserver
```

Slika 14 prikazuje našu finalnu strukturu projekta:



Slika 14 - Struktura projekta

U Django okviru, postoji razlika između projekta i aplikacije. Projekt predstavlja osnovnu strukturu odnosno okvir aplikacije. Sadrži postavke, URL konfiguracije i ostale resurse koji su zajednički za cijelu web aplikaciju. Jedan Django projekt može sadržavati više aplikacija koje se zajedno koriste kako bi se stvorila cjelovita web aplikacija. Sa druge strane, aplikacija predstavlja samostalnu komponentu koja obavlja specifične funkcionalnosti unutar web

aplikacije. Django podržava modularnu strukturu, gdje se različite komponente odnosno aplikacije mogu razvijati neovisno jedna o drugoj te se zatim kombiniraju kako bi se stvorila cjelovita web aplikacija [13].

URL konfiguracije

Nakon stvaranja aplikacije, potrebno ju je dodati u datoteci „settings.py“ kako bi naš projekt znao koju aplikaciju koristiti:

```
INSTALLED_APPS = [
    'zdravstveni_sustav_app.apps.ZdravstveniSustavAppConfig',
    'users.apps.UsersConfig',
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'schedule'
]
```

Pri izradi svake Django aplikacije potrebno je konfigurirati URL-ove. URL konfiguracije mapiraju URL adrese na odgovarajuće poglede (eng. *views*) ili funkcije kada korisnik posjeti određeni URL. URL-ovi su središnji dio mehanizma za usmjeravanje zahtjeva na odgovarajuće dijelove naše web aplikacije. Pri kreaciji Django projekta *zdravstveni_sustav* stvorena je datoteka nazvana „urls.py“ koja sadrži potrebne konfiguracije. Povrh toga, bilo je potrebno napraviti datoteku istog naziva unutar same aplikacije *zdravstveni_sustav_app* [14].

Pogledajmo sadržaj „urls.py“ unutar glavnog projekta *zdravstveni_sustav*:

```
urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('zdravstveni_sustav_app.urls')),
    path('login/', views.login_view, name='login'),
    # path('login/',
auth_views.LoginView.as_view(template_name='users/login.html'), name='login'),
    path('logout/',
auth_views.LogoutView.as_view(template_name='users/logout.html'),
name='logout'),
]
```

Obrazložimo kod iznad. Ovo je lista ruta koja definira kako Django obrađuje različite URL adrese te svaki element liste *urlpatterns* predstavlja jednu rutu.

Ruta *admin/* usmjerava sve URL-ove koji započinju sa *admin/* na administratorsko sučelje u Django okviru.

Ruta *login/* usmjerava na funkciju *login_view* koja je definirana unutar naše „views.py“ datoteke koju ćemo kasnije pregledati. Također dodjeljujemo jedinstveno ime *login* koje se kasnije može koristiti u Django šablonama (engl. *template*) za generiranje URL-ova [14].

Ruta *logout/* usmjerava na ugrađeni Django pogled za odjavu *LoginView* te također ima jedinstveno ime *logout* za generiranje URL-ova [14].

Sljedeća ruta usmjerava sve prazne URL-ove na drugu URL konfiguraciju koja se nalazi unutar *zdravstveni_sustav_app.urls*. To se koristi za bolju organizaciju ruta unutar projekta i aplikacija [14].

```
path('', include('zdravstveni_sustav_app.urls')),
```

Modeli

Pogledajmo sada naše modele. Dakle, modeli se direktno mapiraju u tablice unutar baze podataka. U nastavku je objašnjena klasa *CustomUserManager* koji nasljeđuje *BaseUserManager*. Ova klasa je odgovorna za stvaranje i upravljanje korisnicima, a prilagođeni manager omogućava prilagodbu načina na koji se korisnici stvaraju te kako se njima upravlja u sustavu korisnika [15].

```
class CustomUserManager(BaseUserManager):
    def create_user(self, username, password=None, **extra_fields):
        if not username:
            raise ValueError('Unesite username')
        #email = self.normalize_email(email)
        user = self.model(username=username, **extra_fields)
        user.set_password(password)
        user.save(using=self._db)
        return user

    #kada pokrenemo naredbu manage.py createsuper user ovo se pokreće
    def create_superuser(self, username, password=None, **extra_fields):
        extra_fields.setdefault('is_staff', True)
        extra_fields.setdefault('is_superuser', True)

        if extra_fields.get('is_staff') is not True:
            raise ValueError('Superuser must have is_staff=True.')
        if extra_fields.get('is_superuser') is not True:
            raise ValueError('Superuser must have is_superuser=True.')

        return self.create_user(username, password, **extra_fields)
```

Metoda *create_user()* stvara i sprema običnog korisnika u sustav te prima argumente *username*, *password* i ***extra_fields*. *Username* je obvezan, *a password* opcionalan. Ako je korisnik

prazan podiže se iznimka koja sprječava stvaranje korisnika bez korisničkog imena. Zatim se stvara novi korisnik koristeći *self.model*.

Metoda *create_superuser()* stvara superkorisnika u sustavu (engl. *superuser*). Superkorisnik ima posebne privilegije i pristup svim dijelovima aplikacije, označen je kao osoblje (engl. *staff*) i njegove postavke *is_staff* i *is_superuser* su postavljenje na *True*. Metoda koristi *create_user()* metodu za stvaranje korisnika s dodatnim postavkama za superkorisnika (to je parametar *extra_fields*) [15].

```
class Doktor(AbstractBaseUser, PermissionsMixin):
    username = models.CharField(max_length=30, unique=True)
    email = models.EmailField(blank=True)
    first_name = models.CharField(max_length=30, blank=True)
    last_name = models.CharField(max_length=30, blank=True)
    is_active = models.BooleanField(default=True)
    is_staff = models.BooleanField(default=False)
    date_joined = models.DateTimeField(default=timezone.now)
    #ako se obriše dom_zdravlja, npr ruši se zgrada, postavi doktoru
dom_zdravlja na null (nema smisla brisat doktora)
    dom_zdravlja = models.ForeignKey(DomZdravlja, default=1,
on_delete=models.SET_NULL, null=True)

    # Add custom fields here
    opis_doktora = models.TextField()
    iskustvo_doktora = models.TextField()


    specijalizacija_doktora_izbori = [
        ("Anesteziolog", "Anesteziolog"),
        ("Infektolog", "Infektolog"),
        ("Kardiolog", "Kardiolog"),
        ("Pedijatar", "Pedijatar"),
        ("Reumatolog", "Reumatolog"),
        ("Pulmolog", "Pulmolog"),
        ("Urolog", "Urolog"),
        ("Onkolog", "Onkolog")
    ]
    #vazno da je default inace dobijem error da id ne postoji
    specijalizacija_doktora = models.CharField(max_length=30,
choices=specijalizacija_doktora_izbori, default="AN")

    objects = CustomUserManager()

    USERNAME_FIELD = 'username'
    REQUIRED_FIELDS = []

    def __str__(self):
        return f"{self.first_name} {self.last_name}"
```

Iznad je prikazan kod modela klase Doktor. Model klase Doktor nasljeđuje dvije klase: *AbstractBaseUser* i *PermissionsMixin*. Ove klase pružaju gotove funkcionalnosti i atribute koji su potrebni za korisničke modele.

AbstractBaseUser je apstraktna baza korisnika koja pruža osnovnu funkcionalnost za model korisnika. Nasljeđivanje ove klase omogućava nam prilagodbu modela korisnika prema potrebama naše aplikacije. Značajke te klase uključuju mogućnost definiranja vlastitog polja za

korisničko ime (najčešće e-mail ili username), ugrađene metode za autentifikaciju korisnika i opcionalna podrška za rad sa e-mail adresama kao sa korisničkim imenima.

PermissionMixin klasa pruža gotove funkcionalnosti vezane za ovlasti i dozvole korisnika. Nasljeđivanje ove klase omogućava korisnicima da budu povezani sa različitim dozvolama (engl. *perimission*) i grupama koje se koriste za kontrolu pristupa dijelovima aplikacije. Značajke uključuju dodjelu korisnika različitim dozvolama (npr. čitanje, pisanje, brisanje) te pripadnost korisnika različitim grupama [15].

Username predstavlja korisničko ime doktora. Sa tim imenom, doktor se može prijaviti u aplikaciju. *Email* označava e-mail adresu doktora koje je opcionalno (*blank=True*). *Is_active* i *is_staff* označavaju ako je račun doktora aktivan i ako je doktor osoblje. *First_name* i *last_name* označavaju ime i prezime doktora. Još imamo polja *dom_zdravlja, opis_doktora, specijalizacija doktora* koja označava predefiniranu listu opcija specijalizacije. Metoda *__str__(self)* definira kako će se korisnikov objekt prikazivati kad se pozove funkcija str(doktor). U našem slučaju, korisnik će se prikazivati kao kombinacija imena i prezimena doktora [15].

Opišimo model DomZdravlja.

```
class DomZdravlja(models.Model):
    naziv_doma_zdravlja = models.CharField(max_length=40)
    adresa_doma_zdravlja = models.CharField(max_length=30)
    opis_doma_zdravlja = models.TextField()

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

Dom zdravlja sadrži polja *naziv_doma_zdravlja, adresa_doma_zdravlja i opis_doma_zdravlja* te vraća *naziv_doma_zdravlja* u prikazima.

Pogledajmo model Narudžbe.

```
class Narudzba(models.Model):
    odabir_vremena = (
        ('jutro', "Jutro"),
        ('večer', "Večer")
    )
    ime_pacijenta = models.CharField(max_length=30)
    prezime_pacijenta = models.CharField(max_length=30)
    telefon_pacijenta = models.CharField(max_length=20)
    email_pacijenta = models.EmailField()
    #ako se izbriše doktor izbriše se i narudžba
    doktor = models.ForeignKey(Doktor, on_delete=models.CASCADE)
    datum_pregleda = models.DateField(default=timezone.now)
    vrijeme_pregleda = models.CharField(choices=odabir_vremena, max_length=10)
    # vrijeme_pregelda = models.TimeField(
    #     default='08:00',
```

```
    #     blank=False,
    #     null=False
    # )
    simptomi = models.TextField(blank=True, null=True)

    def __str__(self):
        return f"Doktor {self.doktor.first_name} {self.doktor.last_name} -
Pacijent {self.ime_pacijenta} {self.prezime_pacijenta}"
```

Model narudžbe sadrži sve relevantne podatke i polja o pacijentu i doktoru. Pacijent dakle može unijeti svoje ime, prezime, vrijeme pregleda, datum pregleda pa čak i simptome.

Pogledajmo još model medicinske sestre.

```
class MedicinskaSestra(models.Model):
    ime_sestre = models.CharField(max_length=30)
    prezime_sestre = models.CharField(max_length=30)
    godine_staza_sestre = models.IntegerField()
    #ako se obriše doktor, npr da otkaz, postavi medicinskoj sestri doktor na
null (nema smisla brisat sestru)
    doktor = models.ForeignKey(Doktor, on_delete=models.SET_NULL, null=True)

    def __str__(self):
        return f"{self.ime_sestre} {self.prezime_sestre}"
```

Sve ove modele u Django okviru dodajemo i primjenjujemo sljedećim naredbama u terminalu.

```
python manage.py makemigrations
python manage.py migrate
```

Naredba *python manage.py makemigrations* u Django aplikaciji koristi se za generiranje migracija. Migracije su skripte koje opisuju kako se promjene u modelima aplikacije trebaju odraziti na strukturu baze podataka. Kada mijenjamo modele (npr. dodajemo novo polje ili mijenjamo postojeće), Django koristi migracije kako bi održavao konzistentnost između naših modela i baze podataka. Django stvara datoteke s migracijama u direktoriji *migrations* unutar naše aplikacije. Svaka migracija ima jedinstveno ime i sadrži informacije o promjenama u modelima, uključujući dodavanje, brisanje ili promjene polja, stvaranje i brisanje tablica i tako dalje. Nako generiranja migracija sljedeći korak je primjena migracija na bazu podataka što radimo putem naredbe *python manage.py migrate*. Ova naredba izvršava migracije i ažurira strukturu baze podataka kako bi održavala promjene u modelima [16].

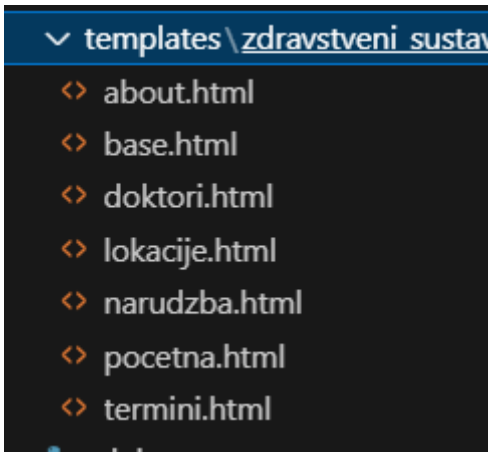
Predlošci

U Django web okviru, predlošci (engl. *templates*) su HTML datoteke koje definiraju izgled i strukturu web stranica. Predlošci omogućuju odvajanje dizajna i prezentacije od logike i podataka u web aplikaciji. Korištenje predložaka omogućuje nam da olakšamo održavanje i skaliranje naše aplikacije, jer možemo promijeniti izgled stranica bez mijenjanja samih pogleda (engl. *views*) ili logike naše aplikacije [17].

Predlošci su HTML datoteke u kojima možemo koristiti Django oznake i sintaksu za dinamičko generiranje HTML-a. Na primjer, možemo koristiti {% for %} petlje za iteriranje kroz podatke iz baze podataka i {% if %} uvjete za kontrolu prikaza određenih dijelova stranice na temelju logike [17].

U template-ima možemo inkorporirati dinamičke podatke iz našeg Django modela ili konteksta koji je poslan iz pogleda. Ovo nam omogućuje da prikažemo različite informacije korisnicima na temelju njihovih akcija ili stanja [17].

Pogledajmo strukturu našeg *templates* direktorija.



Slika 15 - Struktura predložaka

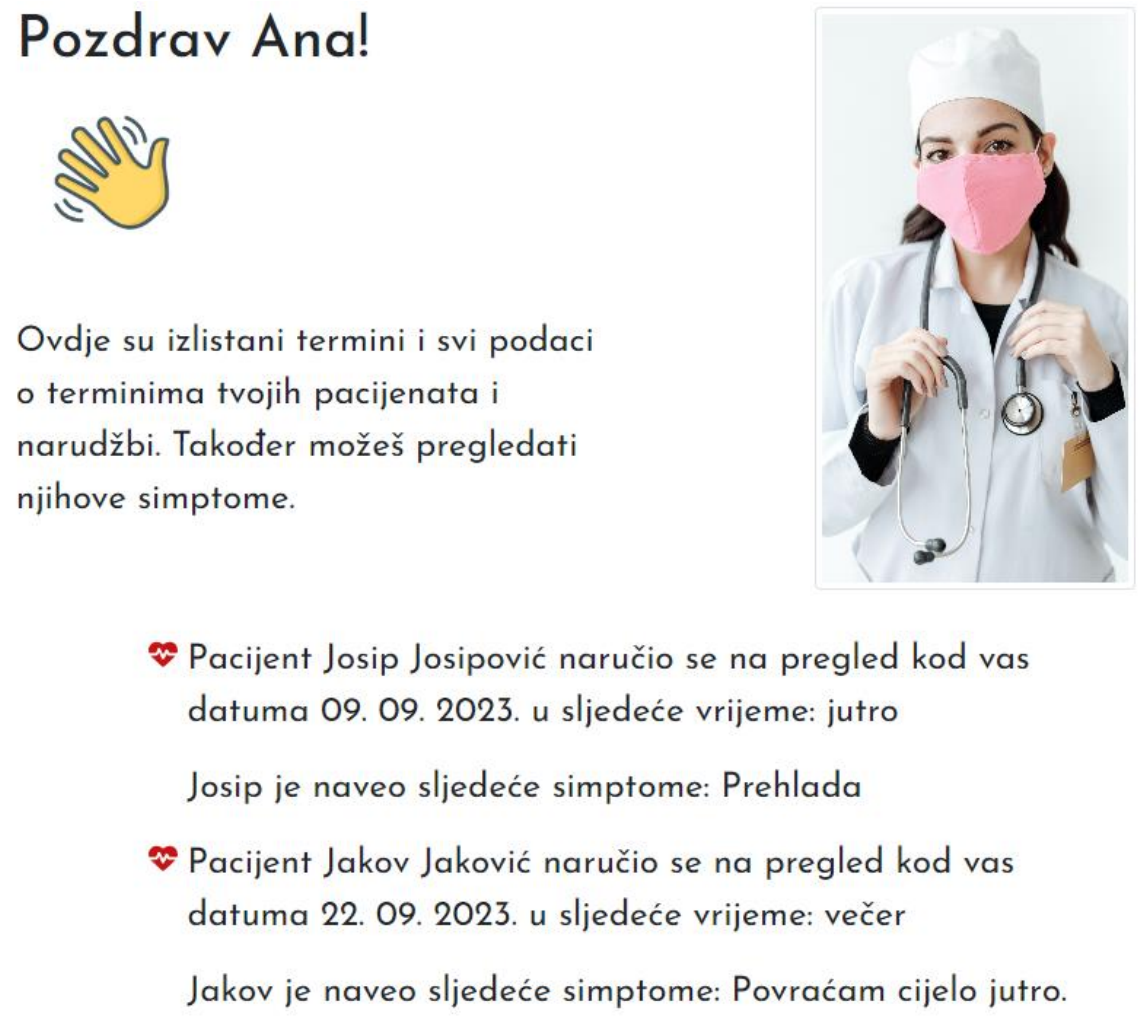
Pogledajmo te proučimo datoteku *termini.html*. Unutar ovog HTML template-a nalazi se HTML kod zajedno sa Django templating jezikom koji prikazuje prijavljenom doktoru termine i detalje o terminima na kategoriji 'Moji termini' [17].

```
{% for narudzba in narudzba_set %}
    <li>
        <p>Pacijent {{ narudzba.ime_pacijenta }} {{ narudzba.prezime_pacijenta
}}
            naručio se na pregled kod vas datuma {{
narudzba.datum_pregleda|date:"d. m. Y." }}
            u sljedeće vrijeme: {{ narudzba.vrijeme_pregleda }}</p>
        <p>{{ narudzba.ime_pacijenta }} je naveo sljedeće simptome: {{
narudzba.simptomi }}</p>
    </li>
{% endfor %}
```

Pogledajmo samo dio koda koji je relevantan za prikazivanje termina. Template *termini.html* dobiva proslijeđenu listu *narudzba_set* od njegovog odgovarajućeg pogleda.

```
return render(request, 'zdravstveni_sustav_app/termini.html', context)
```

Ovo dakle označava da na zahtjev za pristupanje resursu *termini.html* se pošalje *context* varijabla koja sadrži informacije o skupu narudžbi. U template kodu iznad iterira se kroz svaku narudžbu u listi narudžbi te se ispisuje ime pacijenta, prezime pacijenta , datum pregleda, vrijeme pregleda i simptomi. Ovo je dakle sve dinamički generirano umjesto hardcode-ano u HTML datoteku. Pogledajmo sada prikaz narudžbi za jednog od doktora. [18]



Slika 16 - Prikaz doktorovih termina

Ana dakle vidi termine i opise tih termina od svakog pacijenta koji se naručio kod nje.

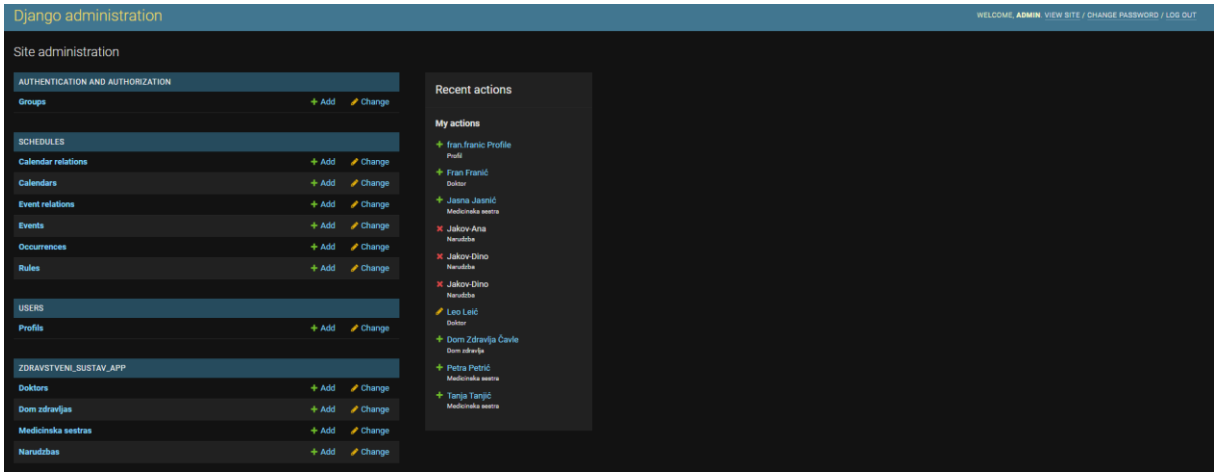
Admin.py i admin sučelje

Da bi se modeli koje smo prethodno bili izradili mogli vidjeti u administratorskom sučelju, potrebno ih je registrirati u *admin.py datoteci* na sljedeći način [19].

```
admin.site.register(Doktor, CustomUserAdmin)
admin.site.register(Narudzba)
admin.site.register(DomZdravlja)
admin.site.register(MedicinskaSestra)
```

Registracija modela za administraciju omogućava nam da pristupimo i upravljamo podacima tog modela putem web sučelja bez potrebe za pisanjem posebnog administrativnog sučelja. Django automatski generira osnovne funkcionalnosti za pregledavanje i uređivanje podataka, a dodatne prilagodbe mogu se napraviti unutar CustomUserAdmin klase kako bi se prilagodio način na koji se podaci prikazuju i upravljaju u admin panelu [19].

Pogledajmo kako izgleda naše administratorsko sučelje na slici 17.



Slika 17 - Admin sučelje

Django administratorsko sučelje nudi moćnu mogućnost dodavanja instanci objekata za naše modele. Ovdje vidimo da su svi modeli registrirani: Doktor, Dom zdravlja, Medicinska sestra, Narudžba i Profil te možemo dodavati sve te objekte [19].

Pogledi

U Django web okviru, pogledi (engl. *views*) predstavljaju Python funkcije ili klase koje obavljaju obradu HTTP zahtjeva i generiranje HTTP odgovora. Pogledi određuju kako će aplikacija reagirati na različite URL adrese i korisničke zahtjeve te što će se prikazati korisnicima [20].

Pogledi definiraju kako će se različiti tipovi HTTP zahtjeva (GET, POST, itd.) obraditi. Na primjer, možemo imati pogled koji prikazuje popis objekata kada se koristi GET zahtjev i dodaje novi objekt u bazu podataka kada se koristi POST zahtjev [20].

Pogledi sadrže poslovnu logiku aplikacije. To može uključivati dohvaćanje podataka iz baze podataka, izračune, validaciju podataka i druge operacije koje su potrebne za obradu zahtjeva. Pogledi su odgovorni za generiranje HTTP odgovora koji se šalju korisnicima. To može uključivati renderiranje HTML template-a, slanje JSON podataka ili generiranje drugih vrsta odgovora. Pogledi se povezuju sa određenim URL adresama putem URL konfiguracije. To znači da svaka URL adresa može biti povezana sa određenim pogledom koji će se izvršiti kada korisnik posjeti tu adresu. Pogledi mogu sadržavati logiku za provjeru autentifikacije korisnika i provjeru ovlasti kako bi se kontrolirao pristup određenim dijelovima aplikacije. Pogledi čine logiku web aplikacije dok template-i čine dizajn [20].

Pogledajmo jednostavan pogled koji prikazuje našu početnu stranicu *pocetna.html*.

```
def pocetna(request):
    return render(request, 'zdravstveni_sustav_app/pocetna.html')
```

Funkcija *pocetna()* dobiva objekt *request* kao argument. *Request* sadrži informacije o HTTP zahtjevu koji je poslan na server, npr. informacije o korisniku, URL adresi, podacima iz zahtjeva i slično. Kada se pozove *pocetna()*, funkcija *render()* generira HTTP odgovor sa HTTP sadržajem [20].

Pogledajmo sada *doktori()* pogled.

```
def doktori(request):
    doktori = Doktor.objects.all()
    return render(request, 'zdravstveni_sustav_app/doktori.html',
{'doktori':doktori})
```

Ovdje pristupamo podacima u našoj bazi podataka na način da pristupimo klasi *Doktor* i izlistavamo sve njezine objekte. To zatim spremamo u varijablu i šaljemo kao varijablu konteksta u Django template *doktori.html* gdje će pomoću te varijable moći pristupati svakom *Doktor* objektu. Na taj način možemo vrlo jednostavno dinamički ispisati informacije o doktorima u našem template-u [20].

Pogledajmo sada kako prijava korisnika funkcionira.

```
def login_view(request):
    if request.method == 'POST':
        form = LoginForm(request.POST)  # Bind the form with POST data

        if form.is_valid():
            username = form.cleaned_data['username']
            password = form.cleaned_data['password']

            user = authenticate(request, username=username, password=password)

            if user is not None:
                login(request, user)
```

```
                # Redirect to a success page or perform other actions
                return redirect('/')
            else:
                messages.error(request, 'Pogrešno korisničko ime ili lozinka.
Pokušajte ponovno.')
        else:
            messages.error(request, 'Form validation failed. Please check your
inputs.')
    else:
        form = LoginForm()  # Create an instance of the login form

    return render(request, 'users/login.html', {'form': form})
```

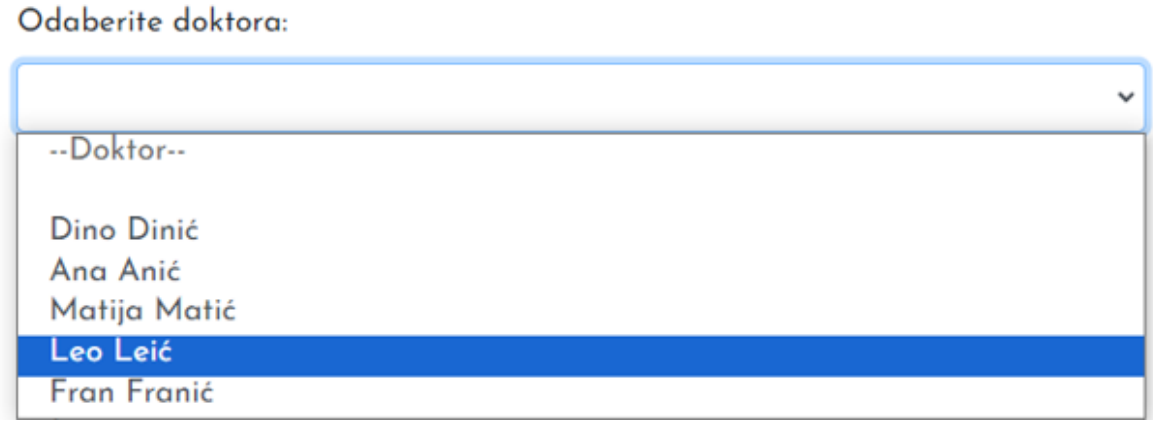
U početku koda provjeravamo ako je HTTP zahtjev tipa POST. To znači da će se blok koda izvršiti samo kada korisnik pošalje obrazac za prijavu (unese korisničko ime i lozinku). Zatim stvaramo instancu Django forme LoginForm i veže se sa POST podacima koji su poslani u zahtjevu. Ovim se korakom prikupljaju podaci koje je korisnik unio u obrazac. Nakon toga provjeravamo ako forma ispravna putem *is_valid()* funkcije. Ako su podaci ispravni odnosno ispravno uneseni korisničko ime i lozinka, onda se nastavlja s prijavom. Zatim dohvaćamo pročišćene podatke te koristimo ugrađenu funkciju *authenticate()* za provjeru korisničkog imena i lozinke unutar baze podataka. Ako postoje korisnički podaci koji se podudaraju s unesenim podacima, vraća se objekt korisnika *user*. Sljedeći uvjet provjerava ako je funkcija *authenticate()* pronašla korisnika u bazi podataka. Ako je to znači da su korisničko ime i lozinka ispravni. Ako je korisnik uspješno prijavljen koristimo funkciju *login()* za stvaranje sesije tog korisnika što označava da je korisnik prijavljen. U slučaju da je prijava uspješna, korisnika se usmjeruje na početnu stranicu aplikacije pomoću *return redirect('/').* Ako postoji neka greška ispisat će se poruka o grešci. Kada se prvi put otvori stranica, zahtjev nije tipa POST te se stoga stvara instanca forme *LoginForm()* kako bi se prikazala prazna forma za unos podataka. Na kraju, poziva se funkcija *render* kako bi se prikazala HTML stranica *login.html* koja sadrži obrazac za prijavu. Forma se šalje kao kontekst kako bi se mogla prikazati na stranici [20].

Pogledajmo sada pogled koji upravlja narudžbama pacijenata.

```
class NarudzbaView(View):
    def get(self, request, *args, **kwargs):
        context = {
            'doktori': Doktor.objects.all()
        }
        return render(request, "zdravstveni_sustav_app/narudzba.html",
context)

    def post(self, request, *args, **kwargs):
        ime_pacijenta = request.POST.get('ime_pacijenta')
        prezime_pacijenta = request.POST.get('prezime_pacijenta')
        telefon_pacijenta = request.POST.get('telefon_pacijenta')
```

```
        email_pacijenta = request.POST.get('email_pacijenta')
        doktor_id = request.POST.get('doktor')
        datum_pregleda = request.POST.get('datum_pregleda')
        vrijeme_pregleda = request.POST.get('vrijeme_pregleda')
        simptomi = request.POST.get('simptomi')
        if doktor_id:
            doktor = get_object_or_404(Doktor, id=doktor_id)

        if(ime_pacijenta and prezime_pacijenta and email_pacijenta and doktor
and datum_pregleda and vrijeme_pregleda):
            Narudzba.objects.create(
                ime_pacijenta=ime_pacijenta,
                prezime_pacijenta=prezime_pacijenta,
                telefon_pacijenta=telefon_pacijenta,
                email_pacijenta=email_pacijenta,
                doktor=doktor,
                datum_pregleda=datum_pregleda,
                vrijeme_pregleda=vrijeme_pregleda,
                simptomi=simptomi
                )

            messages.success(request,'Uspješna prijava narudžbe!')
        return redirect('narudzba')
```

NarudzbaView je klasa koja nasljeđuje *View, osnovnu klasu za Django poglede. get()* metoda obrađuje HTTP GET zahtjeve. Kada korisnik pristupi putanji *'zdravstveni_sustav_app/narudzba.html'*, Django poziva ovu metodu. U toj metodi dohvaćaju se svi postojeći doktori iz baze podataka putem *Doktor.objects.all()* i ti se podaci spremaju u *context* rječnik. Zatim, HTML predložak *narudzba.html* se renderira ovim kontekstom i šalje korisniku kao odgovor. Kontekstna varijabla *doktori* se u predlošku koristi za prikaz liste svih doktora pacijentu [21]. Pogledajmo listu doktora na slici 18:



Slika 18 - Lista doktora

Post() metoda obrađuje HTTP POST zahtjev. Kada korisnik pošalje podatke s web obrasca, Django poziva ovu metodu. U ovoj metodi se prvo dohvaćaju različiti podaci iz POST zahtjeva,

kao što su ime pacijenta, prezime pacijenta, telefon, email, odabrani doktor, datum pregleda, vrijeme pregleda i simptomi. Zatim se provjerava jesu li svi potrebni podaci prisutni i jesu li ispravni. Ako je odabrani doktor postavljen, dohvaća se iz baze pomoću *get_object_or_404* koja ako postoji, vraća objekt, ako ne, vraća 404 [21].

Ako su svi podaci ispravni, nova instanca *Narudzba* modela se stvara putem *Narudzba.objects.create()* funkcije, s unosom svih relevantnih podataka. To znači da se stvara nova narudžba u bazi podataka. Nakon što je narudžba uspješno kreirana, korisniku se prikazuje poruka putem *message.success()* funkcije što koristi Djangov sustav poruka za prikazivanje obavijesti korisniku. Korisnik se na kraju preusmjerava na URL nazvan narudzba [21].

Upotreba aplikacije

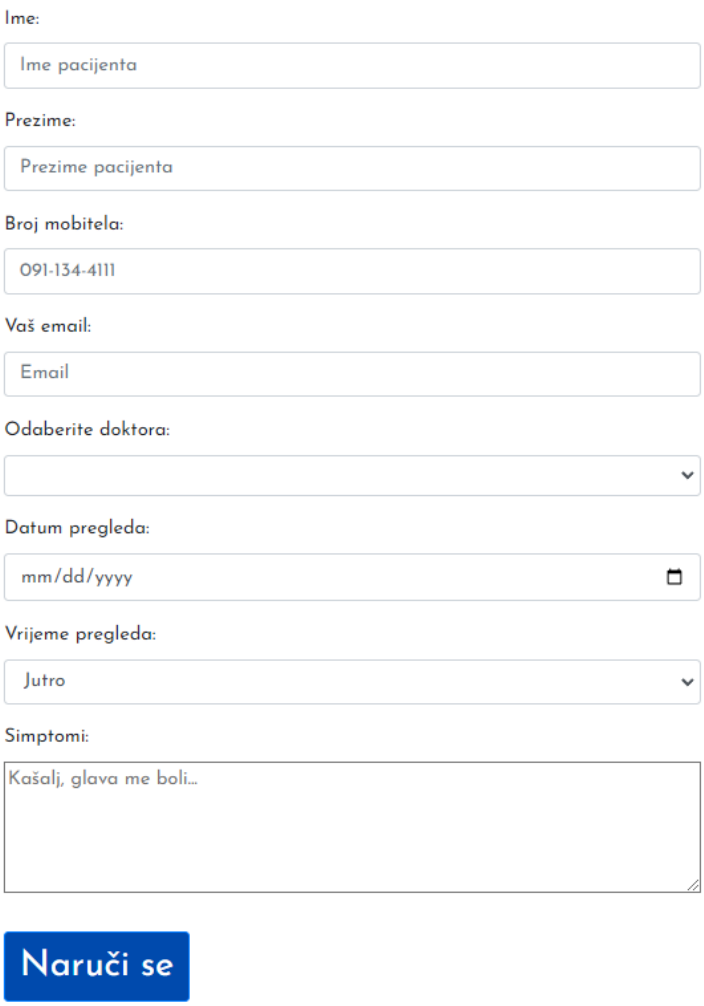
Aplikaciju mogu koristiti tri vrste korisnika: pacijent, doktor i administrator. Prođimo kroz načine uporabe aplikacije za sve od ove tri vrste korisnika.

Pacijent

Na početnoj stranici, pacijent može kliknuti gumb „Naruči se“ koji će ga odvesti na stranicu s obrascem za naručivanje. Slika 19 prikazuje dizajn početne stranice i gumb.

Slika 19 - Gumb "Naruči se"

Ukoliko se pacijent želi naručiti i klikne potrebni gumb, prikazat će mu se stranica „narudzba.html“ koja sadrži sljedeća polja za unos podataka o pacijentu: ime, prezime, broj mobitela, e-mail, datum pregleda, vrijeme pregleda, odabir željenog doktora i simptomi. Slika 20 prikazuje obrazac za unos podataka o pacijentu.



Slika 20 - Obrazac za narudžbu

Unosom svih potrebnih polja i klikom na gumb „Naruči se“ pacijent šalje informacije o narudžbi te se te informacije spremaju u bazu podataka u tablicu „Narudzba“.

Pacijent također može pristupiti kategoriji „O nama“ gdje može saznati razne informacije o tvrtki Medigital, kao što je na primjer njeno sjedište preko Google Karte te čime se tvrtka bavi.

Pacijent može na kategoriji „Doktori“ pregledati informacije o doktorima kao što su: iskustvo doktora, opis doktora, trenutna lokacija gdje doktor radi, koja je specijalizacija doktora i tako dalje.

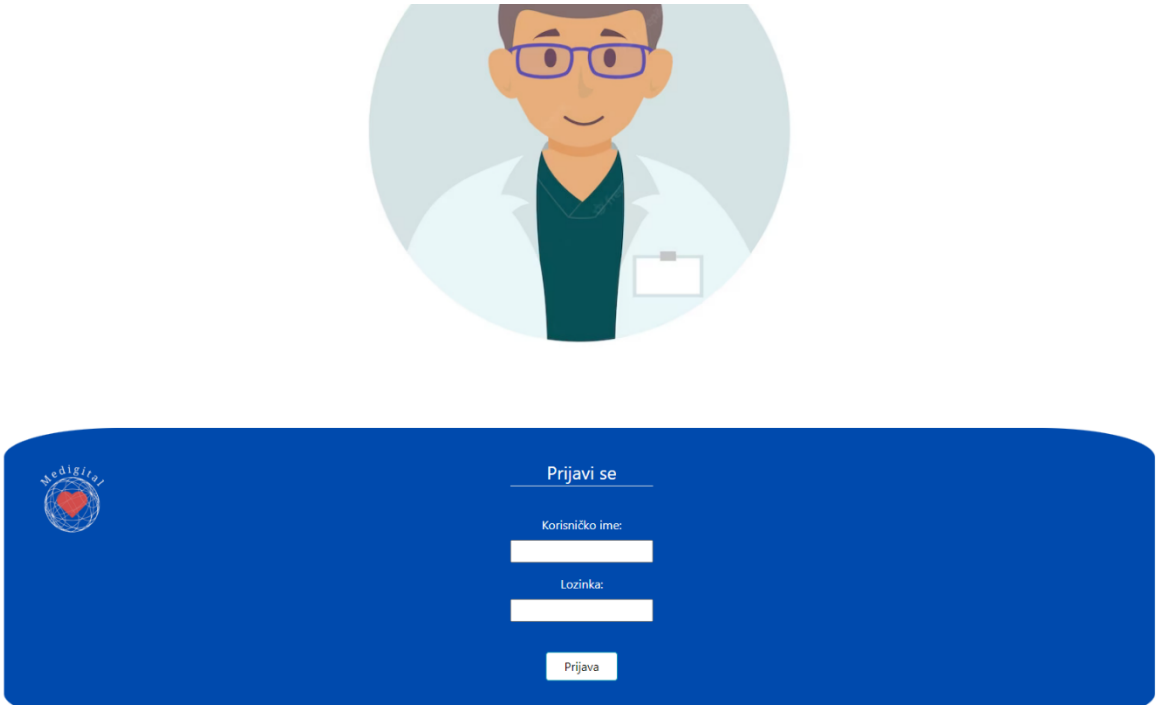
Na kategoriji „Lokacije“ može saznati informacije o domovima zdravlja kao što su opis i adresa doma zdravlja.

Doktor

Doktor se prijavljuje u sustav klikom na gumb „Prijava“ koji se nalazi na navigacijskoj traki. Slika 21 prikazuje gumb za prijavu.

Slika 21 - Gumb Prijava

Nakon što doktor klikne na gumb „Prijava“, preusmjerava ga se na stranicu „login.html“ gdje može unijeti svoje podatke za prijavu. Slika 22 prikazuje stranicu „login.html“.



Slika 22 - Login stranica

Nakon uspješne prijave, korisnika doktora preusmjerava se na početnu stranicu, gdje mu se pojavljuje nova kategorija „Moji termini“ koju prethodno nije mogao vidjeti. Također se gumb „Prijava“ promijenio u gumb „Odjava“ koji odjavljuje doktora ukoliko klikne na njega. Navigacijska traka nakon prijave je prikazana na slici 23.

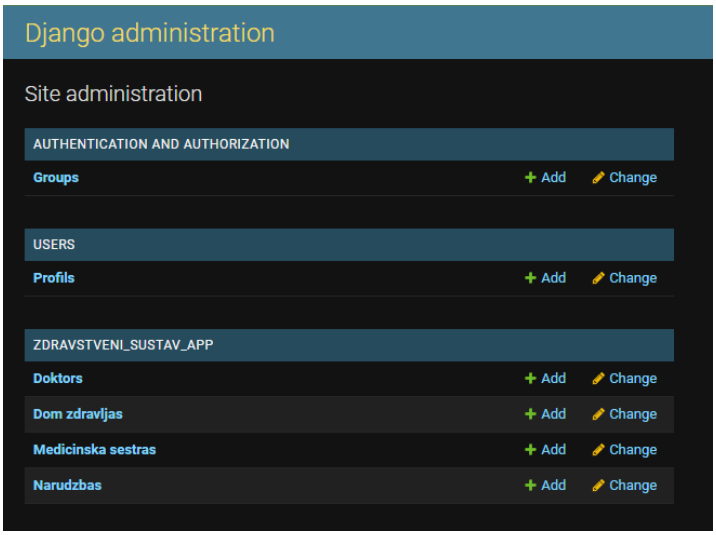
Slika 23 - Navigacija nakon prijave

Doktor na kategoriji „Moji termini“ može vidjeti sve svoje termine i podatke o njima. Može vidjeti ime i prezime pacijenta koji se naručio kod njega, njegove simptome te datum i vrijeme pregleda. Slika 24 prikazuje kategoriji „Moji termini“.

Slika 24 - Kategorija "Moji termini"

Administrator

Korisnik administrator može pristupiti administrativnom sučelju tako da u rutu početne stranice doda '/admin'. Administrator se zatim prijavljuje sa svojom lozinkom i korisničkim imenom te može dodavati, mijenjati i brisati objekte iz tablica „Doktors“, „Dom zdravljas“, „Medicinska sestras“, „Narudzbas“ i „Profils“.



Slika 25 - Stranica admin

Zaključak

U zaključku, prošli smo cijeli proces izrade web aplikacije, od analize sustava, gdje smo opisali značajke sustava, okruženje sustava i dali širu sliku sustava kojeg smo sagradili. Nakon toga slijedila je faza specifikacije korisničkih zahtjeva koja precizno i sažeto određuje što mi zapravo želimo od naše web aplikacije, ali ne i kako to želimo postići. Navodimo koji su korisnici naše aplikacije te u obliku korisničkih priča zapisujemo želje i potrebe svakog pojedinog korisnika. Zatim je uslijedila faza oblikovanja modela podataka gdje smo pomoću modela entiteti-veze (EV model) uspješno modelirali bazu podataka koja se sastoji od sljedećih tipova entiteta koji su u međusobnom odnosu: Doktor, Narudžba, Dom zdravlja, Medicinska sestra i Profil. Nakon toga, odredili smo kako će nam izgledati korisničko sučelje. Primarne boje su plava i bijela te smo slijedili pravila dobrog web dizajna što smo bolje mogli. Za izradu samog dizajna sučelja smo koristili besplatni mockup alat Figmu. Za kodiranje i implementaciju frontend dijela aplikacije koristili smo HTML, CSS, SCSS i popularni CSS okvir za razboj web stranica Bootstrap 4.

Popis slika

Slika 1 - MVC model 3
Slika 2 - Django ORM 5
Slika 3 - Preslikavanje modela u tablicu 5
Slika 4 - Tip entiteta 13
Slika 5 – Atribut 13
Slika 6 - Ključni atribut 14
Slika 7 – Veza 14
Slika 8 - Model podataka 15
Slika 9 - Početna 1 18
Slika 10 - Početna 2 18
Slika 11 – Footer 18
Slika 12 - Početna 1 mobile 19
Slika 13 - Početna 2 mobile 20
Slika 14 - Struktura projekta 21
Slika 15 - Struktura predložaka 28
Slika 16 - Prikaz doktorovih termina 29
Slika 17 - Admin sučelje 30
Slika 18 - Lista doktora 33
Slika 19 - Gumb "Naruči se" 35
Slika 20 - Obrazac za narudžbu 36
Slika 21 - Gumb Prijava 37
Slika 22 - Login stranica 37
Slika 23 - Navigacija nakon prijave 37
Slika 24 - Kategorija "Moji termini" 38
Slika 25 - Stranica admin 38

Literatura

[1] Big Nige. „How Django Works“. https://masteringdjango.com/django-tutorials/beginner-lesson-2-how-django-works/. Zadnje pristupljeno: 19.09.2023.

[2] Ihechikara Vincent Abba. „The Meaning of Object Relational Mapping Database Tools“. https://www.freecodecamp.org/news/what-is-an-orm-the-meaning-of-object-relational-mapping-database-tools/. Zadnje pristupljeno: 19.09.2023.

[3] Codecademy Team. „MVC: Model, View, Controller“. https://www.codecademy.com/article/mvc. Zadnje pristupljeno: 19.09.2023.

[4] adwitiyacode. „MVC Design Pattern“. https://www.geeksforgeeks.org/mvc-design-pattern/. Zadnje pristupljeno: 19.09.2023.

[5] Materijali iz kolegija Programsko inženjerstvo

[6] intersys. „This is How to Write a Foolproof User Requirements Specification“. https://intersys.co.uk/2021/03/24/this-is-how-to-write-a-foolproof-user-requirements-specification/. Zadnje pristupljeno: 19.09.2023.

[7] Pavlić, Mile, Oblikovanje baza podataka, Odjel za informatiku, Sveučilište u Rijeci, Rijeka, 2011.Datoteka

[8] Patrycja Dybka. „Chen Notation“. https://vertabelo.com/blog/chen-erd-notation/. Zadnje pristupljeno: 19.09.2023.

[9] Craig Stedman. „data modeling“. https://www.techtarget.com/searchdatamanagement/definition/data-modeling. Zadnje pristupljeno: 19.09.2023.

[10] Dusan Rodina. „Chen ER Diagram“. https://www.softwareideas.net/chen-er-diagram-erd. Zadnje pristupljeno: 19.09.2023.

[11] Ben Kopf. „The Power of Figma as a Design Tool“. https://www.toptal.com/designers/ui/figma-design-tool. Zadnje pristupljeno: 19.09.2023.

[12] airfocus.com. „What is a Mockup?“. https://airfocus.com/glossary/what-is-a-mockup/. Zadnje pristupljeno: 19.09.2023.

[13] Django dokumentacija. „Writing your first Django app, part 1“. https://docs.djangoproject.com/en/4.2/intro/tutorial01/. Zadnje pristupljeno: 19.09.2023.

[14] Django dokumentacija. „URL dispatcher“. https://docs.djangoproject.com/en/4.2/topics/http/urls/. Zadnje pristupljeno: 19.09.2023.

[15] Django dokumentacija. „Models“. https://docs.djangoproject.com/en/4.2/topics/db/models/. Zadnje pristupljeno: 19.09.2023.

[16] w3schools. „Django Models“. https://www.w3schools.com/django/django_models.php. Zadnje pristupljeno: 19.09.2023.

[17] Django dokumentacija . „Templates“. https://docs.djangoproject.com/en/4.2/topics/templates/. Zadnje pristupljeno: 19.09.2023.

[18] Django dokumentacija. „The Django template language“. https://docs.djangoproject.com/en/4.2/ref/templates/language/. Zadnje pristupljeno: 19.09.2023.

[19] Django dokumentacija. „The Django admin site“. https://docs.djangoproject.com/en/4.2/ref/contrib/admin/. Zadnje pristupljeno: 19.09.2023.

[20] Django dokumentacija. „Writing views“. https://docs.djangoproject.com/en/4.2/topics/http/views/. Zadnje pristupljeno: 19.09.2023.

[21] w3schools. „Django Views“. https://www.w3schools.com/django/django_views.php. Zadnje pristupljeno: 19.09.2023.

[22] Margaret Rouse. „Systems Analysis“. https://www.techopedia.com/definition/9611/systems-analysis. Zadnje pristupljeno: 19.09.2023.

[23] Django dokumentacija. „Applications“. https://docs.djangoproject.com/en/4.2/ref/applications/. Zadnje pristupljeno: 19.09.2023.