Web aplikacija za rad ambulante opće prakse

Anišić, Tin

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

Stručni studij

WEB APLIKACIJA ZA RAD AMBULANTE OPĆE PRAKSE

Završni rad

Tin Anišić

Osijek, 2021.

SADRŽAJ

1. UVOD

Svrha ovog završnog rada je izraditi web aplikaciju za rad ambulante opće prakse. Danas se u zdravstvu koriste programi poput Softmed 2, za koji je potrebna instalacija na svako računalo medicinskog osoblja što je dodatan trošak vremena i novca. Dizajniranjem web aplikacije riješio bi se problem pristupačnosti i takav sustav ne bi bio ovisan o operacijskom sustavu ili hardveru. Jedini zahtjev je da medicinsko osoblje i pacijent imaju pristup internetu i internetski preglednik. Također, web aplikacija je korisna radi organizacije medicinskog pregleda na način da se pacijent javi sa opisom bolesti i datumom kada može doći na pregled, što doktoru olakšava praćenje slobodnih termina. Zbog naglog porasta popularnosti web aplikacija napravljeni su razni okviri (engl. *frameworks*) za brzi razvoj web aplikacija. Za kreiranje web aplikacije koristit će se Pythonov web okvir zvan Django kojeg karakterizira sigurnost, visoka skalabilnost i brz razvoj aplikacije. U ovom radu detaljnije ću objasniti zadatak završnog rada, alate koje sam koristio za izradu web aplikacije i konkretno izradu aplikacije.

1.1. Zadatak završnog rada

Zadatak završnog rada bio je izraditi web aplikaciju za rad ambulante opće prakse. Aplikacija ima 3 role:

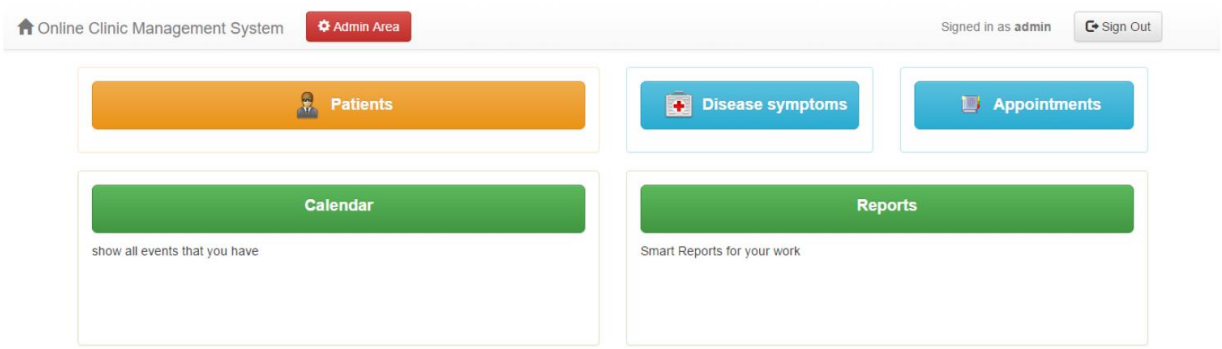
- Liječnik
- Medicinska sestra
- Pacijent

Liječnik ima najviše ovlasti, može unositi podatke vezane uz pregled pacijenta i ažurirati podatke o pacijentu. Medicinska sestra ima ovlasti unijeti novog pacijenta, uređivati osobne podatke pacijenata te printati nalaze, ali nema pravo unositi podatke vezane za pregled. Pacijent ima najmanje ovlasti i može se naručiti za pregled i pregledati svoje osobne podatke.

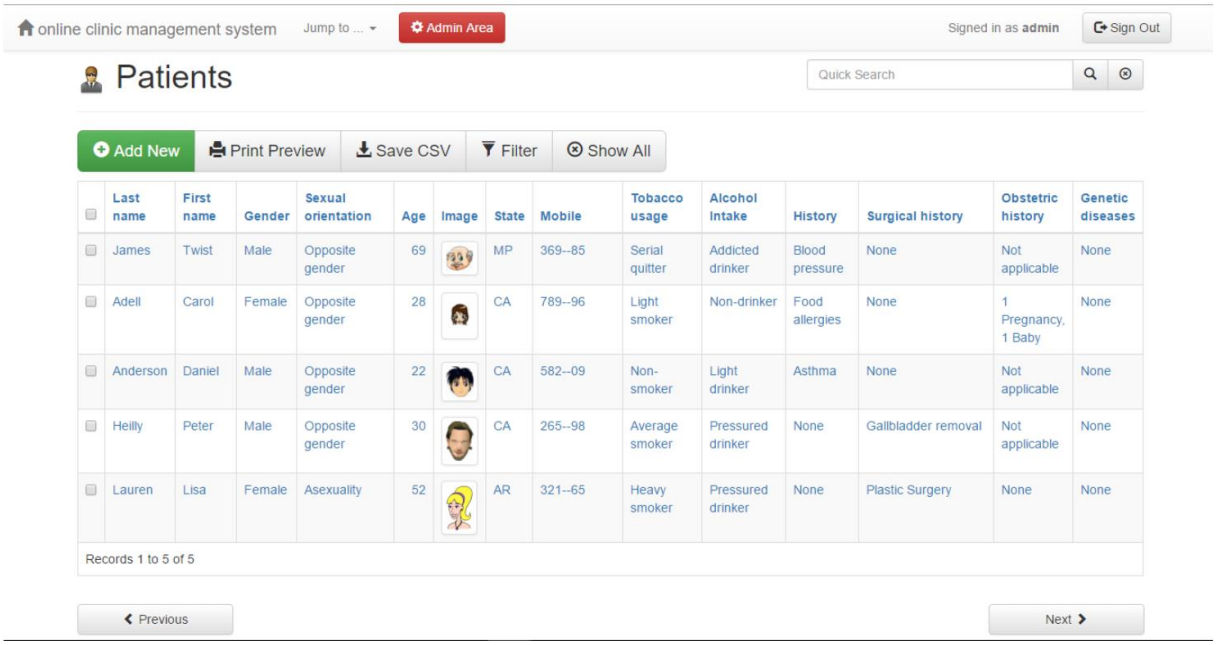
2. SLIČNE WEB APLIKACIJE

Na internetu postoji nekoliko sličnih programskih rješenja, među njima najsličnija je web aplikacija **Online Clinic Management System (OCMS)** [1]. Aplikacija je kreirana koristeći AppGini [2] alat. AppGini je alat koji je razvila tvrtka **BigProf Software** i svrha mu je brz razvoj web aplikacije. Korisnik ne mora poznavati programske jezike da bi koristio program, već je samo potrebno da korisnik definira svoju bazu podataka, sve ostalo se samostalno generira. Neke od prednosti OCMS web aplikacije su:

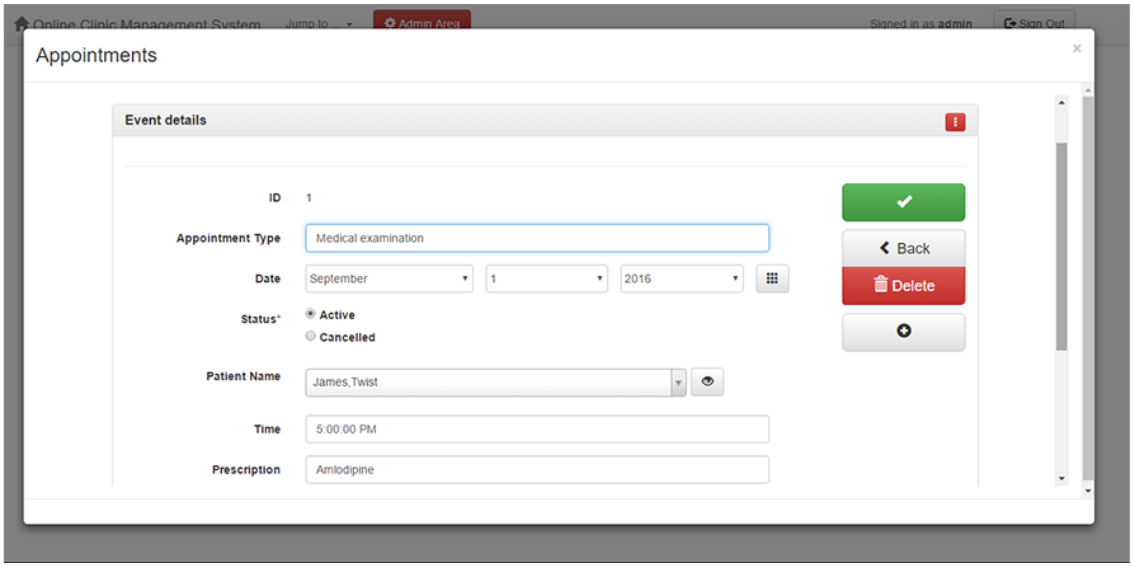
- Mogućnost uvida u genetske bolesti pacijenta
- Mogućnost uvida u pacijentove navike (pušenje, unos alkohola)
- Mogućnost praćenja opstetričke povijesti pacijenta



Sl. 2.1. Izgled kontrolne ploče OCMS aplikacije



Sl. 2.2. Izgled liste pacijenata OCMS aplikacije



Sl. 2.3. Izgled forme za dodavanje pregleda OCMS aplikacije

3. KORIŠTENI ALATI I TEHNOLOGIJE

Ovo poglavlje opisuje sve alate i tehnologije koje su korištene u izradi završnog rada.

3.1. Django web okvir

Django je web okvir otvorenog koda (engl. *open-source*) visoke razine (engl. *high-level*) koji je napravljen u Python programskom jeziku i baziran je na model-view-template (MVT) arhitekturi. Razvio ga je tim web programera koji je bio odgovoran za stvaranje i održavanje novinske web stranice između 2003. i 2005. godine [1]. Django karakterizira sigurnost, velika skalabilnost, jednostavno održavanje i brz razvoj aplikacije [2]. Sadrži gotove alate koji pomažu lakše i sigurnije razviti web aplikaciju. Neki od alata su:

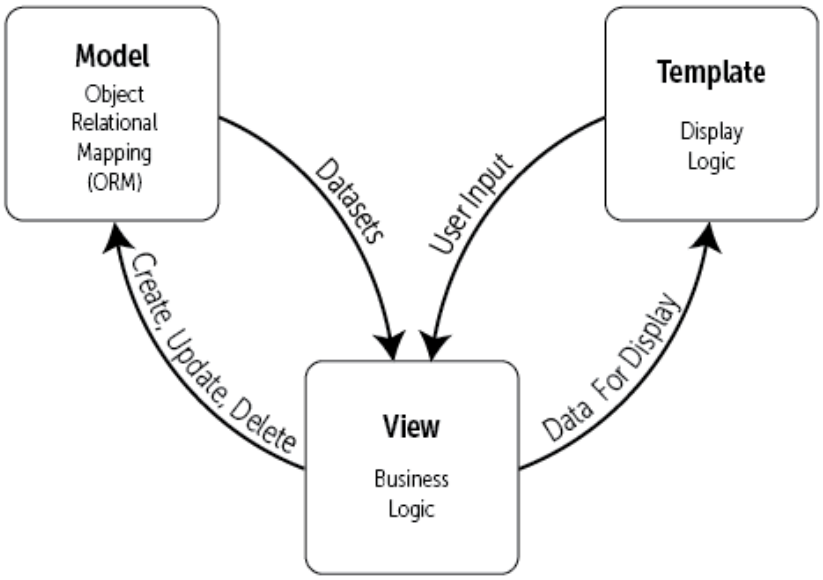
- Objektno-relacijsko mapiranje (ORM) (engl. *Object-relational mapping*)
- Administratorsko sučelje (engl. *Admin interface*)
- Predlošci za dinamičko generiranje HTML koda (engl. *Templates*)
- Zaštita od SQL ubrizgavanja (engl. *SQL injection*)
- SSL/HTTPS enkripcija
- Zaštita od krivotvorenja zahtjeva na više lokacija (CSRF) (engl. *Cross site request forgery*)
- Sučelje za rad s SQL i NoSQL bazama podataka.

Django se temelji na *„Don't Repeat Yourself“* (DRY) filozofiji koja je usmjerena prema smanjenju redundancije koda odnosno svaki različiti koncept ili dio koda treba postojati samo na jednom mjestu. Također jedan od bitnijih Django zahtjeva je labavo spajanje (engl. *loose coupling*) i čvrsta kohezija (engl. *tight cohesion*) kako različiti slojevi okvira ne bi trebali znati jedni o drugima osim ako je to prijeko potrebno.

3.1.1. MVT arhitektura

Model-view-template (MVT) arhitektura je slična model-view-controller (MVC) arhitekturi. Glavna razlika je u tome što je u MVC arhitekturi potrebno napisati kod za kontroler, ali u MVT arhitekturi okvir (engl. *framework*) brine o kodu kontrolera [3]. Model predstavlja programski kod koji opisuje entitete u bazi podataka. View predstavlja programski kod koji opisuje logiku i

odlučuje koji podaci će se moći prikazati u predlošku (engl. *Template*). Template se koristi kao predložak za definiranje izgleda web stranice i prosljeđuju mu se podaci za prikaz. Template je napisan kao HTML dokument koji koristi Django prezentacijski jezik (engl. *markup language*) za predloške. Na slici 3.1. je vidljiv prikaz MVT arhitekture koja se koristi u Django okviru.



Sl. 3.1. MVT arhitektura.

3.2. Bootstrap 4

Bootstrap 4 je jedan od najčešće korištenih okvira za kreiranje responzivnog dizajna web stranice. Zasniva se na već napravljenom Cascading Style Sheets (CSS) i JavaScript (JS) kodu. Bootstrap omogućuje jednostavan razvoj pomoću sustava mreže (engl. *grid system*) koja se zasniva na 12 stupaca koji se mogu spajati. Spajanjem stupaca dobiva se raspored koji služi kao okvir za web stranicu. Također Bootstrap nudi korištenje gotovih komponenti poput navbar-a, gumba, formi za unos podataka. Korištenjem gotovih klasa se definira dizajn stranice.

3.3. GitLab

GitLab je započeo kao projekt otvorenog koda za pomoć timovima koji surađuju u razvoju softvera. Ono što ističe GitLab od drugih servisa za verzioniranje koda je to što je GitLab DevOps platforma. DevOps je kombinacija kulturnih filozofija, praksi i alata koji povećavaju sposobnost organizacije da vrlo brzo isporuče aplikacije ili usluge. DevOps posjeduje različite alate kojima je cilj smanjiti životni ciklus razvoja softvera (engl. *development life cycle*). Koristiti će se Git kao sustav kontrole verzija (engl. *version control system*).

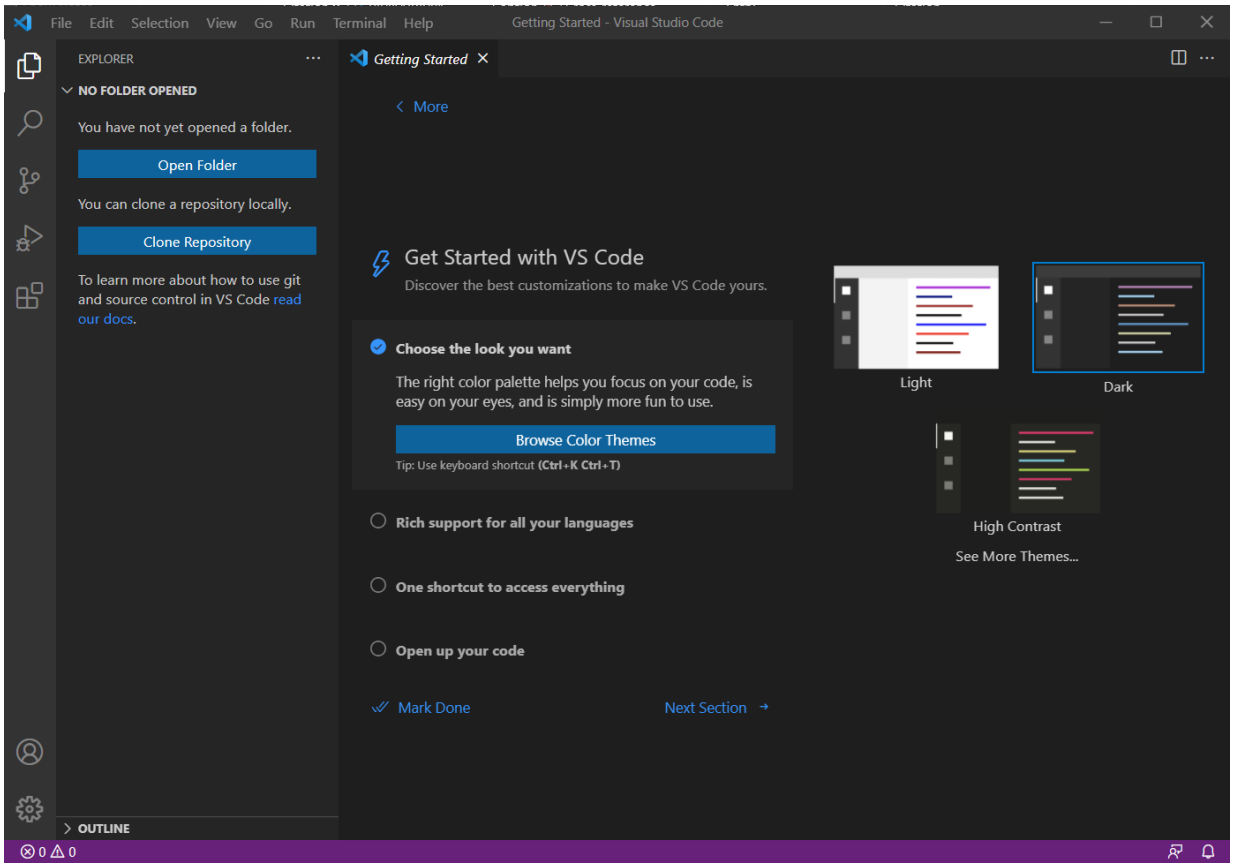
3.4. Visual Studio Code

Visual Studio Code je lagan (engl. *lightweight*), ali snažan uređivač teksta (engl. *text editor*) koji je kreiran od strane Microsofta. Dostupan je za Windows, Linux i macOS operacijske sustave.

Prednosti ovog alata su:

- Besplatan
- Program za provjeru i analizu sintakse (engl. *code linter*)
- Mogućnost detaljnog uređivanja postavki uređivača
- Velik broj raznih proširenja (engl. *extensions*)
- Automatsko dovršavanje koda (IntelliSense) (engl. *code completion*)
- Mogućnost sinkronizacije postavki putem GitHub računa
- Podrška za velik broj programskih jezika
- Program za ispravljanje pogrešaka (engl. *debugger*)
- Podržava verzioniranje softvera.

U istraživanju nad programerima Stack Overflow 2019 [4]. Visual Studio Code je sa 50.7% glasova rangiran kao najbolje razvojno okruženje za razvoj softvera.



Sl. 3.2. Izgled Visual Studio Code uređivača teksta.

3.5. Lunacy

Lunacy je besplatan alat za dizajn web i mobilnih aplikacija zasnovan na vektorskoj grafici. Razvio ga je Icons8. Dostupan je za Windows operacijske sustav. Koristi ***.sketch*** format koji je namijenjen za macOS operacijski sustav. Na tržištu postoji nekoliko aplikacija koje omogućuju otvaranje .sketch datoteka, ali niti jedna od aplikacija ne jamči točnost prikazivanja na ekranu od barem 80% [5]. Lunacy jamči 99% točnosti prikaza bez obzira na verziju formata. Lunacy podržava export .sketch datoteke u nekoliko formata:

- PNG
- JPEG
- BMP
- ICO
- WEBP
- SVG
- PDF

4. PRAKTIČNI DIO RADA

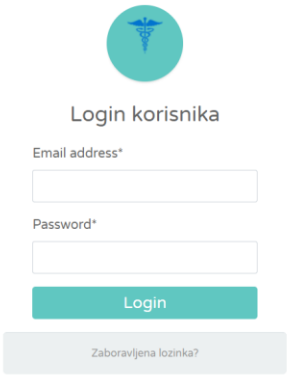
U ovom poglavlju opisivat će se praktični dio kreiranja web aplikacije. Prvo je bilo potrebno napraviti maketu web aplikacije, a zatim se kodirala aplikacija.

4.1. Izrada makete

Maketa (engl. *Mockup*) predstavlja statičnu sliku aplikacije srednje do visoke vjernosti. Maketa se uvijek radi prije kodiranja. Cilj makete je prikazati sheme boja, raspored sadržaja, fontove, ikone, navigaciju, slike i cjelokupni osjećaj budućeg dizajna softvera. Za potrebe završnog rada korišten je program Lunacy.

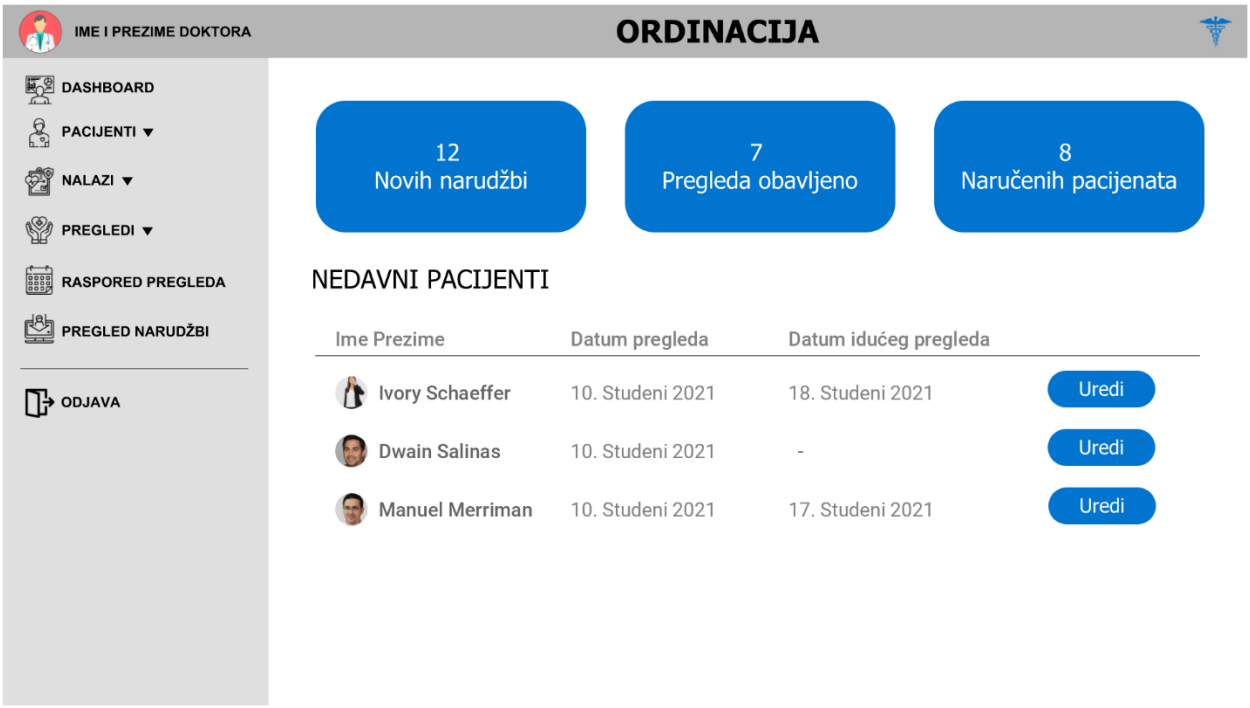
Maketu početne stranice moguće je vidjeti na slici 4.1. Na njoj su prikazane osnovne informacije o doktoru i ordinaciji. Prilikom klika na poveznicu „prijava“ koja se nalazi na navigaciji, otvara se stranica za ulogiravanje u sustav koja je vidljiva na slici 4.2.

Sl. 4.1. Maketa početne stranice web aplikacije.



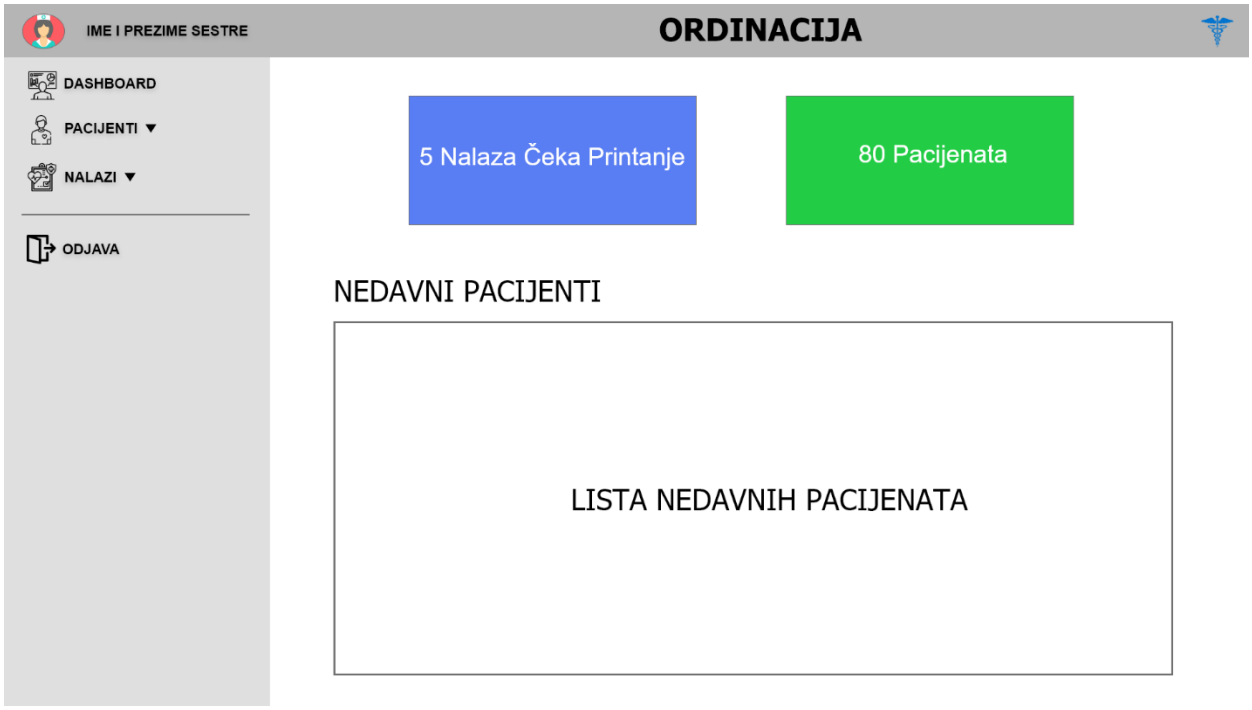
Sl. 4.2. Maketa login stranice.

Kada se korisnik logira, ovisno o svojoj roli prikaže mu se stranica kontrolne ploče. Primjer kontrolne ploče doktora vidljiv je na slici 4.3. Vidljivo je kako doktor ima najviše mogućnosti za odabir, jer on posjeduje najviše ovlasti.



Sl. 4.3. Maketa kontrolne ploče za doktora.

Medicinska sestra ima puno manje ovlasti od doktora i zbog toga ima manje mogućnosti za odabir u kontrolnoj ploči. Maketa kontrolne ploče za medicinsku sestru vidljiva je na slici 4.4.



Sl. 4.4. Maketa kontrolne ploče za medicinsku sestru.

Rola pacijenta je zamišljena da ima najniže ovlasti. Na slici 4.5. prikazana je maketa kontrolne ploče za pacijenta. Pritiskom na poveznicu „naruči se za pregled“ otvara se forma koju popunjava pacijent u svrhu prijave bolesti.

Sl. 4.5. Maketa kontrolne ploče za pacijenta.

4.2. Virtualna okolina

Virtualna okolina (engl. *Virtual environment*) je alat koji pomaže pakete ovisne (engl. *dependencies*) o projektu držati odvojene od drugih projekata na način da kreira izoliranu okolinu za izvođenje. Korišten je ***Ubuntu Linux*** ugrađen u Windows 10 operacijski sustav (WSL) [6] (engl. *Windows Subsystem for Linux*), ***Python*** verzije 3.8.5 i ***python3-venv*** paket koji omogućuje korištenje Python virtualne okoline.

Prvo je potrebno napraviti direktorij u kojem će se nalaziti projekt i virtualna okolina naredbom *mkdir zavrsni-rad*. Nakon što je direktorij kreiran potrebno je ući u direktorij naredbom *cd zavrsni-rad* i instalirati virtualnu okolinu naredbom *python -m venv venv*. Ova naredba kreira virtualnu okolinu naziva *venv*. Kada je kreirana virtualna okolina potrebno ju je aktivirati naredbom *source venv/bin/activate*. Iz slike 4.6. oznaka *(venv)* ispred nove linije simbolizira aktiviranu virtualnu okolinu naziva „venv“.

```
tinac@DESKTOP-42OKGR9:~/zavrsni-rad$ source venv/bin/activate
(venv) tinac@DESKTOP-42OKGR9:~/zavrsni-rad$
```

Sl. 4.6. Aktivacija virtualne okoline „venv“.

Ako korisnik želi napustiti virtualnu okolinu to može naredbom *deactivate* što prikazuje slika 4.7.

Sl. 4.7. Deaktivacija virtualne okoline.

4.3. Instalacija paketa

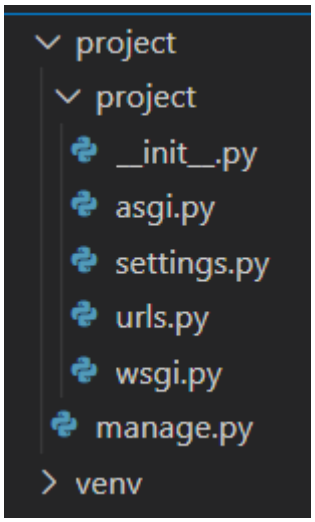
Za instalaciju Python paketa koristi se ***pip*** upravitelj paketima (engl. *Package manager*). Pip je alat koji dopušta instalaciju i upravljanje dodatnim bibliotekama (engl. *Libraries*) koje nisu distribuirane kao dio standardne Python biblioteke. Važno je napomenuti kako je pip potrebno koristiti unutar virtualne okoline. Za instalaciju Django paketa koristi se naredba *pip install django* koja će preuzeti (engl. *Download*) i instalirati paket unutar virtualnog okruženja.

4.4. Kreiranje projekta

Projekt predstavlja Django web aplikaciju u cjelini. Django projekt služi kao kontejner za Django aplikacije. Za kreiranje projekta potrebno je u terminalu pozicionirati se u glavni direktorij projekta („*zavrsni-rad*“) i provjeriti je li aktivirana virtualna okolina. Ako nije, tada je potrebno unijeti naredbu sa slike 4.6. Ako je virtualna okolina već aktivirana, Django projekt se kreira naredbom *django-admin startproject <ime_projekta>*. U završnom radu za ime projekta je korišten naziv ***project***. Preporučeno je ne koristiti ***„django“*** ili ***„test“*** kao ime projekta radi mogućeg konflikta s ugrađenim Python paketima.

4.4.1. Struktura Django projekta

Na slici 4.8. vidljiva je struktura praznog Django projekta.



Sl. 4.8. Struktura Django projekta bez kreiranih aplikacija.

Svaki novi projekt, kada se kreira, posjeduje datoteke:

- ***__init__.py*** je prazna Python datoteka koja predstavlja direktorij u kojem se nalazi kao Python modul.
- ***manage.py*** sadrži kod za pokretanje servera i rad sa migracijama. Ne smije se izmjenjivat njezin sadržaj.
- ***settings.py*** je datoteka u kojoj se nalaze sve opcije vezane za projekt. U nju se dodaju aplikacije, međuprogrami (engl. *middleware*), baza podataka i informacije o predlošcima.
- ***urls.py*** datoteka sadrži sve URL uzorke (engl. *pattern*) i veže poglede (engl. *View*) s određenim URL uzorkom.
- ***asgi.py*** je ASGI (engl. *Asynchronous Server Gateway Interface*). U ranijim verzijama Django okvira nije postojao. Smatra se kako je ASGI nasljednik WSGI sučelja.
- ***wsgi.py*** je WSGI (engl. *Web Server Gateway Interface*). Koristi se za postavljanje projekta na udaljeni server i ne smije se izmjenjivati.

4.5. Kreiranje aplikacija unutar projekta

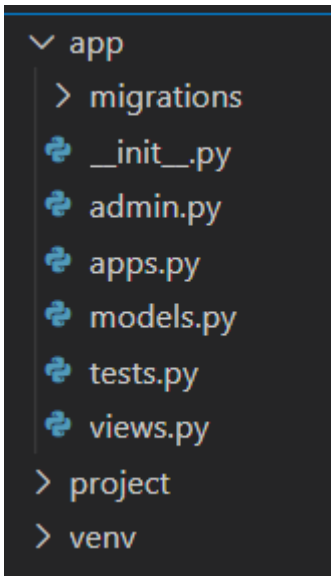
Kada je projekt kreiran potrebno je kreirati aplikacije koje će imati odvojene funkcionalnosti. Aplikacija predstavlja modul projekta koji je potpuno neovisan o drugim aplikacijama u projektu. U teoriji aplikaciju bi bilo moguće kopirati u neki drugi projekt i ona bi trebala raditi bez ikakvih dodatnih promjena u kodu.

Aplikacija se kreira naredbom *django-admin startapp <ime aplikacije>*. U svrhu završnog rada projekt je podijeljen na dvije aplikacije. Prva aplikacija se koristi u svrhu upravljanja korisnicima i nazvana je ***account***. U prvoj aplikaciji je definiran model za korisnika (engl. *User*) i predložak za logiranje korisnika aplikacije. Druga aplikacija naziva ***app*** ima svrhu upravljanja s medicinskim modelima, pogledima i predlošcima za medicinsko osoblje i pacijente.

4.5.1. Struktura Django aplikacije

Na slici 4.9. vidljiva je struktura Django aplikacije unutar projekta. Svaka aplikacija posjeduje sljedeće datoteke:

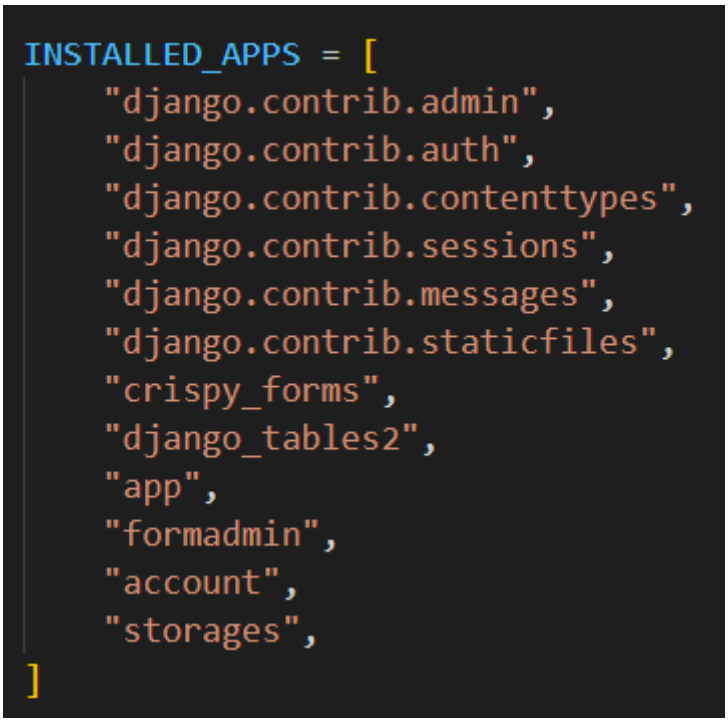
- ***migrations*** predstavlja direktorij u kojem se nalaze migracije, odnosno tamo se spremaju sve izmjene na modelima.
- ***__init__.py***
- ***admin.py*** je datoteka u kojoj se registriraju modeli za uređivanje putem admin sučelja.
- ***apps.py*** je datoteka u kojoj se definiraju različiti parametri o aplikaciji kao na primjer ime aplikacije.
- ***forms.py*** je datoteka u kojoj se definiraju forme za unos podataka u ovisnosti o modelu.
- ***models.py*** je datoteka u kojoj se definiraju modeli (entiteti) od kojih će biti napravljena baza podataka.
- ***tests.py*** je datoteka u kojoj se pišu unit testovi u svrhu provjere ispravnosti napisanog koda.
- ***urls.py***
- ***views.py*** je datoteka u kojoj se pišu pogledi, odnosno Python kod koji prima Web zahtjev (engl. *Request*) i vraća Web odgovor (engl. *Response*).



Sl. 4.9. Struktura Django aplikacije nazvane „app“.

4.5.2. Spajanje projekta i aplikacije

Kada su kreirani i projekt i aplikacije potrebno ih je spojiti. Spajanje se radi u ***settings.py*** datoteci koja se nalazi u korijenskom direktoriju projekta. Svaka aplikacija unutar projekta posjeduje datoteku naziva ***apps.py*** u kojoj je definirano ime aplikacije. Unutar ***settings.py*** datoteke nalazi se lista naziva ***INSTALLED_APPS***. Svrha te liste je uključiti aplikacije čiji je naziv naveden u listi. U Python programskom jeziku lista je definirana uglatim zagradama i svaki element liste se odvaja zarezom. Na slici 4.10. vidljiva je lista aplikacija koja se koristila u svrhu završnog rada.



Sl. 4.10. Prikaz instaliranih aplikacija unutar settings.py datoteke.

4.6. Postavljanje projekta na GitLab

Prilikom postavljanja projekta na GitLab potrebno je odvojiti virtualno okruženje jer se ono nikad ne dodaje u repozitorij zbog svoje veličine. To se radi tako da se u *.gitignore* datoteku doda *venv/* čime je osigurano da se virtualno okruženje ne postavi na GitLab. Datoteka *.gitignore* je obična tekstualna datoteka u kojoj svaki redak sadrži obrazac (engl. *pattern*) za zanemarivanje datoteka ili direktorija. U *.gitignore* datoteku je poželjno dodati Python keš (engl. *Cache*) datoteke **.pyc* i *.vscode* direktorij koji sadrži postavke Visual Studio Code uređivača teksta. To je poželjno zato što na aplikaciji radi više različitih programera koji koriste različite postavke uređivača teksta.

Kako bi se osiguralo da projekt ima sa sobom sve pakete, ali da se ne postavlja virtualno okruženje na GitLab kreira se datoteka *requirements.txt* u kojoj će se nalaziti imena i verzije paketa koje projekt zahtijeva. To je jednostavno moguće naredbom *pip freeze > requirements.txt*. Na slici 4.11. je vidljivo da *pip freeze* naredba ispisuje sve pakete instalirane u virtualnom okruženju i njihove verzije. Dodavanjem *> requirements.txt* sadržaj rezultata naredbe *pip freeze* se sprema u tekstualnu datoteku *requirements.txt*. Datoteku *requirements.txt* je važno dodati na GitLab jer ona sadrži sve informacije o paketima koje će biti potrebno instalirati prilikom postavljanja projekta na server. Također je vrlo važno ažurirati datoteku *requirements.txt* ako se obriše ili doda novi paket.

```
(venv) tinac@DESKTOP-42OKGR9:~/zavrsni-rad$ pip freeze
appdirs==1.4.4
asgiref==3.3.4
black==21.6b0
click==8.0.1
Django==3.2
django-crispy-forms==1.11.2
django-phonenumber-field==5.0.0
django-tables2==2.4.0
mypy-extensions==0.4.3
pathspec==0.8.1
phonenumberslite==8.12.21
pytz==2021.1
regex==2021.4.4
sqlparse==0.4.1
toml==0.10.2
```

Sl. 4.11. Paketi instalirani unutar virtualnog okruženja „venv“.

Ako se radi na drugom Django projektu, potrebno je aktivirati virtualnu okolinu tog projekta i instalirati programske pakete naredbom *pip install -r requirements.txt* gdje *-r* označava da se radi o datoteci *requirements.txt*, a ne o paketu.

4.7. Modeli

Model općenito definira strukturu pohranjenih podataka uključujući vrste polja (atributa), maksimalnu veličinu polja, zadane vrijednosti (engl. *default values*) i slično. Model zapravo predstavlja tablicu u bazi podataka. U Django okviru modeli se definiraju unutar ***models.py*** datoteke i svaki model nasljeđuje Python klasu ***django.db.models.Model***. S tako definiranim modelom Django omogućuje API (engl. *Application Programming Interface*) za pristup bazi podataka.

Django koristi ***SQLite3*** kao zadanu bazu podataka. U *settings.py* datoteci je moguće promijeniti bazu podataka. Na slici 4.12. vidljiv je Python kod za postavke baze podataka. Također je vidljivo da Django posjeduje pozadinsku aplikaciju (engl. *backend*) za upravljanje ***SQLite3*** bazom podataka. U svrhu završnog rada korištena je ***SQLite3*** baza podataka.

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

Sl. 4.12. Prikaz postavki baze podataka unutar settings.py.

4.7.1. Korisnički model

Zadani model korisnika u Djangu koristi korisničko ime (engl. *username*) za jedinstvenu identifikaciju korisnika tijekom provjere autentičnosti. Kako bi se koristila e-mail adresa za identifikaciju potrebno je kreirati model za prilagođenog korisnika (engl. *Custom user model*). Uvijek se preporučuje postavljanje prilagođenog korisničkog modela prilikom kreiranja novog Django projekta. Kod za model korisnika piše se u ***models.py*** datoteci ***account*** aplikacije. Slika 4.13. prikazuje polja (atribute) korisničkog modela.

```
class CustomUser(AbstractUser, PermissionsMixin):
    username = None
    user_type = models.CharField(
        max_length=10, choices=Types.choices, default=Types.PATIENT
    )
    MBO = models.BigIntegerField(unique=True, verbose_name="MBO", validators=[MBO_validator])
    first_name = models.CharField(max_length=50, verbose_name="Ime")
    last_name = models.CharField(max_length=50, verbose_name="Prezime")
    picture = models.ImageField(
        upload_to="images/",
        default=DEFAULT_PROFILE_IMAGE_LOCATION,
        verbose_name="Slika profila",
    )
    address = models.CharField(max_length=150, verbose_name="Adresa stanovanja")
    telephone = PhoneNumberField(unique=True, blank=False, verbose_name="Broj telefona")
    date_of_birth = models.DateField(
        verbose_name="Datum rođenja",
        validators=[app.validators.validate_birthday_not_in_future],
    )
    email = models.EmailField(unique=True, verbose_name="E-mail adresa")
    is_staff = models.BooleanField(default=False)
    is_active = models.BooleanField(default=True)
    is_superuser = models.BooleanField(default=False)
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)
```

Sl. 4.13. Model korisnika.

CustomUser klasa predstavlja prilagođeni model (engl. *custom model*) za korisnika. Klasa nasljeđuje klasu ***AbstractUser*** za koju zahtjeva ažuriranje ***settings.py*** datoteke. Klasa ***AbstractUser*** se koristi ako se ne želi mijenjati postupak autentikacije i ako se u model žele dodati dodatne informacije o korisniku. Na slici 4.14. vidljiv je kod koji definira korisnički model unutar *settings.py* datoteke. Prilagođeni model korisnika bi se u idealnom slučaju trebao kreirati na početku razvoja projekta, jer će dramatično utjecati na shemu baze podataka.

```
AUTH_USER_MODEL = "account.CustomUser"
```

Sl. 4.14. Definiranje modela korisnika u settings.py.

4.7.2. Migracija baze podataka

Migracije su Djangov način širenja promjena koje unose u svoje modele. To mogu biti dodavanje polja, brisanje polja, izmjena naziva polja i sl. Migracije su dizajnirane da uglavnom budu automatske. Datoteke za migraciju se nalaze unutar svake aplikaciji u direktoriju „*migrations*“ i dizajnirane su da budu predane i distribuirane kao dio njene kodne osnove (engl. *codebase*).

Kada je napravljen model za korisnika potrebno je unesti naredbu *python manage.py makemigrations*. Ova naredba je odgovorna za stvaranje novih migracija na temelju promjena koje su napravljene u modelima. Slika 4.15. prikazuje primjer kreiranja migracije. Nakon što su migracije napravljene potrebno ih je primijeniti na aplikaciju naredbom *python manage.py migrate*. Slika 4.16. prikazuje primjenu migracije na aplikaciju.

```
(venv) tinac@DESKTOP-42OKGR9:~/zavrsni-rad/project$ python manage.py makemigrations
Migrations for 'account':
  account/migrations/0004_alter_customuser_user_type.py
    - Alter field user_type on customuser
```

Sl. 4.15. Primjer kreiranja migracije.

```
(venv) tinac@DESKTOP-42OKGR9:~/zavrsni-rad/project$ python manage.py migrate
Operations to perform:
  Apply all migrations: account, admin, app, auth, contenttypes, sessions
Running migrations:
  Applying account.0004_alter_customuser_user_type... OK
```

Sl. 4.16. Primjena migracije.

4.8. Pogledi

U Djangu pogled je Python funkcija koja prima web zahtjev (engl. *web request*), a za povratnu vrijednost ima web odgovor [7] (engl. *web response*). Pogled sadrži proizvoljnu logiku potrebnu za vraćanje web odgovora. Konvencija je poglede držati u datoteci ***views.py*** unutar aplikacije koja je odgovorna za određeni pogled. U svrhu završnog rada korišteni su klasni pogledi (engl. *class-based views*) zbog jednostavnosti izrade web aplikacije.

4.8.1. Klasni pogledi

Pogled također može biti i Python klasa koja nasljeđuje klasu *View*. U klasnom pogledu moguće je premostiti (engl. *override*) metode za HTTP akcije. Klasni pogledi se koriste kako bi se jednostavnije napisao pogled i kako bi se izbjeglo pisanje funkcija. Postoji više vrsta generičkih klasnih pogleda i koriste se ovisno o tome što se prikazuje na stranici. Svi generički klasni pogledi temelje se na osnovnoj klasi *View*. Neki od generičkih pogleda su:

- TemplateView
- RedirectView
- DetailView
- ListView
- UpdateView
- DeleteView

```
class CustomLoginView(LoginView):

    template_name = "registration/login.html"
    redirect_authenticated_user = True
```

Sl. 4.17. Klasni pogled za login.

Ako se koriste Django generički pogledi onda nije potrebno koristiti metode, ali je moguće. Na slici 4.18. vidljiv je jednostavan Python kod za prikazivanje predloška početne stranice.

```
class IndexView(TemplateView):
    template_name = "app/index.html"
```

Sl. 4.18. Klasni pogled za početnu stranicu aplikacije.

4.8.2. URL putanje

Kako bi se spojili pogledi s URL oznakom potrebno je u ***urls.py*** datoteci napisati URL putanje (engl. *path*). Putanje se pišu unutar *urlpatterns* liste i ako se koriste klasni pogledi potrebno je dodati i *as_view()* metodu za prikaz pogleda. U slučaju da se koriste pogledi kao funkcije onda nije potrebno dodati *as_view()* metodu jer je povratna vrijednost funkcije web odgovor. Primjer putanja za aplikaciju account vidljiv je na slici 4.19.

```
urlpatterns = [
    path("login/", views.CustomLoginView.as_view(), name="login"),
    path("logout/", auth_views.LogoutView.as_view(), name="logout"),
]
```

Sl. 4.19. URL putanje aplikacije account.

4.8.3. Mixini

Mixin je klasa koja sadrži metode koje koriste druge klase bez potrebe da bude nadređena (engl. *parent*) klasa tim klasama. Koriste se za kontrolirano dodavanje funkcionalnosti klasama, te kada

je potrebno koristiti određene redundantne značajke unutar više klasa. Svaki mixin bi trebao imati jedinstvenu odgovornost. Mixini nisu zamišljeni da budu produživani ili instancirani, već su zamišljeni da budu isključivo naslijeđeni. U završnome radu, mixini su korišteni kako bi se napravilo odvajanje permisija po rolama, što po zadanome nije omogućeno.

Na Sl. 4.20. napisan je mixin kojemu je svrha provjeriti rolu korisnika prilikom pristupa pogledu. Vidljivo je kako je nadjačana (engl. *override*) metoda ***dispatch*** koja je posrednik između web zahtjeva i web odgovora. Sl. 4.21. prikazuje pogled koji nasljeđuje mixin. U pogledu je potrebno definirati samo dopuštene role kao listu rola. U slučaju da pogledu pokuša pristupiti korisnik koji nije logiran ili nema odgovarajuću rolu, podići (engl. *raise*) će se ***PermissionDenied*** iznimka koja će prikazati stranicu HTTP pogreške 403, te pristup pogledu neće biti moguć.

```python
class HasRolePermissionMixin(object):
    allowed_roles = []

    def dispatch(self, request, *args, **kwargs):
        roles = self.allowed_roles
        user = request.user
        if user.is_authenticated:
            if user.get_user_role() in roles:
                return super().dispatch(request,*args,**kwargs)
        raise PermissionDenied
```

Sl. 4.20. Mixin za provjeru role korisnika.

```python
class PatientEditView(
    SuccessMessageMixin, UserContextMixin, HasRolePermissionMixin, UpdateView
):
    template_name = "app/update_patient.html"
    allowed_roles = [Types.DOCTOR, Types.NURSE]
    model = Patient
    form_class = EditPatientForm
    success_message = "Pacijent uspješno izmjenjen!"

    def get_success_url(self):
        return reverse_lazy("app:dashboard")
```

Sl. 4.21. Primjer pogleda koji koristi HasRolePermissionMixin.

4.9. Predlošci

Django predložak je tekstualna datoteka i može generirati bilo koji format zasnovan na tekstu (HTML, XML, CSV, itd.). Predložak sadrži varijable koje se zamjenjuju vrijednostima kada se predložak prikazuje. Django sustav predložaka nudi oznake koje funkcioniraju slično nekim programskim konstrukcijama kao što su *if* uvjet, *for* petlja i slično.

Predlošci se nalaze unutar direktorija ***templates***. U predlošku se također može koristiti statični sadržaj kao što su CSS (engl. *Cascading Style Sheets*) datoteke, JavaScript kod ili slike koje se koriste kao dodatak na HTML dokument. Statični sadržaj se nalazi u direktoriju ***static*** i najbolje ga je odvajati po aplikacijama radi organizacije projekta.

4.9.1. Produživanje predloška

Predložak se može produžiti na način da se u predlošku koji se želi ponašati kao osnovni predložak doda blok koji označava prostor za dodavanje drugog predloška unutar bloka. Na slici 4.22. je prikazan blok naziva content kojemu je namjena prikazat sadržaj predloška koji ga produžuje.

```
{% block content %}
{% endblock %}
```

Sl. 4.22. Blok za produživanje predloška.

Na slici 4.23. prikazan je predložak koji produžuje *base_dashboard.html* i unutar bloka naziva content umeće sadržaj iz bloka.

```
{% extends 'app/base_dashboard.html' %}
{% load static %}


{% block title %}
Svi pacijenti | ORDINACIJA
{% endblock title %}

{% block content %}
<div class="container">
    <div class="card shadow">
        <div class="card-header">
            <h4>SVI PACIJENTI</h4>
        </div>
        {% if  patients.data|length > 0 %}
        {% load render_table from django_tables2 %}
        <div class="card-body">
            {% render_table patients %}
        </div>
        {% else %}
        <br>
        <p class="text-secondary" style="margin-left: 30px; margin-bottom: 20px;">Nema upisanih pacijenata.</p>
        {% endif %}
    </div>
</div>

{% endblock content %}
```

Sl. 4.23. Predložak koji se produžuje blokom.

4.10. Forme

U HTML jeziku, forma je kolekcija elemenata unutar *<form>* elementa koja daje korisnicima stranice mogućnost unosa podataka i slanja tih podataka na server. Django pruža API za rad sa formama. Za rad sa formama koriste se ***GET*** i ***POST*** HTTP metode. ***GET*** metoda se koristi za prikaz web stranice, a ***POST*** metoda se koristi za slanje podataka iz forme na server. Za prikaz formi korišten je dodatak za Django zvan ***crispy forms*** koji koristi *Bootstrap4* dizajn za prikaz. Na slici 4.24. važno je za primjetiti kako se koristi *CSRF* token koji služi za zaštitu od krivotvorenja podataka.

```
<form method="post" id="login-form" novalidate>
  {% csrf_token %}
  {{ form|crispy }}
  <input type="submit" value="Login" class="btn btn-primary btn-lg btn-block">
</form>
```

Sl. 4.24. Primjer forme za login.

Django forme se uobičajeno pišu kao klase unutar ***forms.py*** datoteke. Svaka forma nasljeđuje osnovnu klasu *Form*, a postoje i generičke klase za forme koje olakšavaju rad s formama. Slika 4.25. prikazuje formu za uređivanje pacijenta.

```
class EditPatientForm(forms.ModelForm):
    class Meta:
        model = Patient
        fields = (
            "first_name",
            "last_name",
            "picture",
            "date_of_birth",
            "address",
            "MBO",
            "telephone",
            "email",
        )
```

Sl. 4.25. Forma za uređivanje pacijenta.

4.10.1. Formset

Skup obrazaca (engl. formset) je sloj apstrakcije koji omogućuje prikazivanje više formi na jednoj stranici. Za potrebe završnog rada korišten je inline formset. Inline formset je formset koji pojednostavljuje slučaj rada sa objektima povezanim putem stranog ključa. Kod kreiranja i uređivanja pregleda moguće je dodati uputnice, lijekove i nalaze kao datoteke. Svaki od tih medicinskih objekata su modeli koji su preko stranog ključa povezani sa pregledom. Unutar forms.py datoteke definirani su formsetovi, što je vidljivo na

Sl. 4.26.

```
ReferralFormSet = inlineformset_factory(
    MedicalExamination, MedicalReferral, fields=("referral_name",), extra=3
)
RecipeFormSet = inlineformset_factory(
    MedicalExamination, Drug, fields=("drug_name",), extra=3
)
ReportFormSet = inlineformset_factory(
    MedicalExamination, MedicalReport, fields=("report_file",), extra=3
)
```

Sl. 4.26. Formsetovi za medicinske objekte unutar forms.py datoteke.

Inline formset se kreira metodom ***inlineformset_factory*** [8] koja kao prvi parametar prima roditeljski model, kao drugi parametar prima model koji se želi nadopuniti kao forma,te je još potrebno odrediti koja će se polja iz modela koristiti za prikaz na formi (fields). Postoje još dodatne postavke kao što je inicijalni broj formi (extra), mogućnost brisanja forme i sl.

```
def form_valid(self, form):
    context = self.get_context_data()
    referral_formset = context["referral_formset"]
    recipe_formset = context["recipe_formset"]
    report_formset = context["report_formset"]

    if (
        form.is_valid()
        and referral_formset.is_valid()
        and recipe_formset.is_valid()
        and report_formset.is_valid()
    ):

        self.object = form.save(commit=False)
        self.object.doctor = self.request.user
        self.object.save()

        Appointment.objects.create(
            date_time=self.object.date,
            patient=self.object.patient,
            medical_examination=self.object,
            is_first=True,
            is_accepted=True,
            is_created_by_patient=False,
        )

        if self.object.next_appointment:
            Appointment.objects.create(
                date_time=self.object.date,
                patient=self.object.patient,
                medical_examination=self.object,
                is_first=True,
                is_accepted=True,
                is_created_by_patient=False,
            )

        referral_formset.instance = self.object
        referral_formset.save()

        recipe_formset.instance = self.object
        recipe_formset.save()

        report_formset.instance = self.object
        report_formset.save()
        messages.success(self.request, "Pregled uspješno kreiran!")
    else:
        messages.error(self.request, "Pregled nije uspješno kreiran!")
    return super().form_valid(form)
```

Sl. 4.27. Formsetovi za medicinske objekte unutar forms.py datoteke.

Na Sl. 4.27. vidljiv je kod metode ***form_valid***. Svrha form_valid metode je spremiti instancu forme u bazu podataka i prosljediti web odgovor na *success_url*. Unutar metode se nalazi kod koji jednostavno povezuje formset sa medicinskim pregledom putem atributa ***instance*** i sprema sve promjene u bazu podataka.

5. POSTAVLJANJE APLIKACIJE

U ovom poglavlju opisati ću proces postavljanja web aplikacije na Heroku platformu. Radi poboljšanja performansi korišten je AWS (engl. *Amazon Web Services*) S3 Bucket servis.

Prije postavljanja aplikacije na oblak (engl. *cloud*), potrebno je pripremiti projekt za produkciju. Postavke aplikacije se nalaze u datoteci ***settings.py***. U settings.py datoteci se također nalaze i osjetljivi podaci kao što su API ključevi, Django tajni ključ, podaci za e-mail servis, AWS tajni ključ i sl. Ti podaci se ne postavljaju na git iz sigurnosnih razloga. Stoga je dobra praksa sakriti te podatke u zasebnu datoteku naziva *.env* i ime te datoteke dodati u *.gitignore* datoteku [8]. U *.env* datoteci se nalaze vrijednosti parova pohranjenih kao ključ i vrijednost. Također je potrebno namjestiti u *settings.py* datoteci iščitavanje podataka iz *.env* datoteke.

5.1. Heroku

Heroku je cloud platforma kao servis (PaaS) (engl. *Platform as a Service*) koja je zasnovana na kontejnerima. Developeri koriste Heroku za implementiranje, upravljanje i skaliranje modernih aplikacija, bez da brinu o održavanju servera. Heroku ima opciju za besplatno postavljanje aplikacija što je prikladno za studente. Aplikacija se pokreće kao lagani (engl. *lightweight*), izolirani Linux kontejner zvan „dynos“. Više kontejnera je tipično pokrenuto na jednom dijeljenom serveru, ali su oni međusobno izolirani jedni od drugih, imaju svoju memoriju, operacijski sustav i datotečni sustav. Heroku zahtjeva da projekt posjeduje datoteke:

- **Procfile** sadrži komande koje se izvršavaju pri postavljanju projekta na Heroku,
- **requirements.txt**,
- **runtime.txt** sadrži verziju Pythona koja će se koristiti pri instaliranju projekta na Heroku.

Proces postavljanja aplikacije na Heroku je vrlo jednostavan i sastoji se od par koraka:

1. Kreiranje Heroku računa,
2. Instalacija Heroku CLI sučelja (potrebno samo ako projekt nije na GitHub-u)
3. Kreiranje praznog Heroku projekta,
4. Ako projekt već nije na git platformi, onda ga je potrebno postaviti (preporučen GitHub),
5. Ako se koristi GitHub onda je postavljanje aplikacije automatizirano, inače se promjene trebaju commitati na git i gurnuti (engl. *push*) na heroku naredbom *git push heroku master* (potreban Heroku CLI).

5.1.1. PostgreSQL

Za proizvodnju aplikacije koristila se SQLite3 baza podataka. SQLite3 je relacijska baza podataka koja se nalazi na disku (engl. *serverless*) i ona nije predviđena kao baza za produkciju jer nije dovoljno pouzdana, te s njom nije moguće istovremeno zapisivati više podataka. Zbog toga se za produkciju koristi PostgreSQL baza podataka. PostgreSQL je objektno-relacijska baza podataka otvorenog koda koja je u razvoju već više od 30 godina s čime je zaslužila veliku reputaciju u vidu pouzdanosti, robusnosti i visokih performansi. PostgreSQL se koristi kada su integritet i pouzdanost podataka od velikog značaja. PostgreSQL je dostupan na Heroku platformi uz projekt s određenim limitacijama. Kako bi se postavila PostgreSQL baza podataka umjesto SQLite3 baze podataka potrebno je napraviti izmjene koje su vidljive na Sl. 5.1. Zamućeni podaci predstavljaju osjetljive informacije koje se spremaju u *.env* datoteku. Drugi i zadnji korak je napraviti migraciju naredbom *python manage.py makemigrations*, te primijeniti zadnju migraciju naredbom *python manage.py migrate*.

```
DATABASES = {
    "default": {
        "ENGINE": "django.db.backends.postgresql_psycopg2",
        "NAME": [redacted],
        "USER": [redacted],
        "PASSWORD": [redacted],
        "HOST": "[redacted]eu-west-1.compute.amazonaws.com",
        "PORT": "5432",
    }
}
```

Sl. 5.1. PostgreSQL podaci u settings.py datoteci.

5.2. Amazon S3

Kako aplikacija zahtjeva učitavanje datoteka (engl. *file upload*) medicinskih nalaza online korišten je Amazon S3 (engl. *Amazon Simple Storage Service*). Amazon S3 je dizajniran kako bi omogućio jeftinu pohranu podataka na webu. Podaci se pohranjuju u kante (engl. *buckets*). Proces postavljanja S3 Bucketa je vrlo jednostavan:

1. Potrebno je napraviti korisnički račun na AWS (engl. *Amazon Web Services*),
2. Pronaći S3 servis i napraviti bucket sa jedinstvenim imenom,
3. Dodijeliti dopuštenja (engl. *permissions*),

4. U Django projekt instalirati module *django-storages* i *boto3* naredbom *pip install django-storages boto3* [9],

5. U *settings.py* dodati ***AWS_ACCES_KEY_ID***, ***AWS_SECRET_ACCESS_KEY*** i ***AWS_STORAGE_BUCKET_NAME*** s vlastitim vrijednostima, kao što je vidljivo na Sl. 5.2.

```
# AWS S3 podaci
AWS_ACCESS_KEY_ID = 
AWS_SECRET_ACCESS_KEY = 
AWS_STORAGE_BUCKET_NAME = 
AWS_QUERYSTRING_AUTH = False
AWS_S3_REGION_NAME = "eu-west-2"
```

Sl. 5.2. AWS S3 podaci u settings.py datoteci.

6. ZAKLJUČAK

Danas se javlja sve veća potreba za digitalizaciju zdravstvenog sustava. Cilj završnog rada bio je napraviti web aplikaciju za rad ambulante opće prakse kojom bi se omogućilo jednostavniji rad s pacijentima. Aplikaciju karakterizira jednostavno grafičko sučelje koje služi za jednostavnije vođenje i planiranje medicinske ambulante opće prakse. Velika prednost ove aplikacije je mogućnost pacijenta da pošalje opis tegoba i termin pregleda koji mu odgovara, što smanjuje potrebu za komunikaciju telefonom. S doktorove strane aplikacija pomaže pri organizaciji pregleda, generiranjem uputnica i kalendarskim prikazom termina.

LITERATURA

[1] »Online Clinic Management System,« BigProf Software, [Mrežno]. Available: https://bigprof.com/appgini/applications/online-clinic-management-system. [Pokušaj pristupa 5 listopad 2021].

[2] »AppGini, web database applications builder without coding,« BigProf Software, [Mrežno]. Available: https://bigprof.com/appgini/. [Pokušaj pristupa 5 listopad 2021].

[3] »Django introduction - Learn web development | MDN,« [Mrežno]. Available: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Introduction#where_did_it_come_from. [Pokušaj pristupa 1 srpanj 2021.].

[4] »The Web framework for perfectionists with deadlines | Django,« [Mrežno]. Available: https://www.djangoproject.com/. [Pokušaj pristupa 30 June 2021.].

[5] E. Medunoye, »Django : The Fun part — Understanding the Framework | by Eniola Medunoye | Noteworthy - The Journal Blog,« [Mrežno]. Available: https://blog.usejournal.com/django-the-fun-part-understanding-the-framework-1bb4df54ab1f. [Pokušaj pristupa 1 Srpanj 2021.].

[6] »Stack Overflow Developer Survey 2019,« [Mrežno]. Available: https://insights.stackoverflow.com/survey/2019#technology-_-most-popular-development-environments. [Pokušaj pristupa 1 Srpanj 2021].

[7] »About Lunacy,« [Mrežno]. Available: https://docs.icons8.com/about/#what-is-lunacy. [Pokušaj pristupa 5 srpnja 2021].

[8] »Windows Subsystem for Linux Installation Guide for Windows 10,« [Mrežno]. Available: https://docs.microsoft.com/en-us/windows/wsl/install-win10. [Pokušaj pristupa 1 Srpanj 2021].

[9] »Writing views | Django documentation | Django,« [Mrežno]. Available: https://docs.djangoproject.com/en/3.2/topics/http/views/. [Pokušaj pristupa 13 Srpanj 2021.].

[10] »Model Form Functions | Django documentation | Django,« [Mrežno]. Available: https://docs.djangoproject.com/en/3.2/ref/forms/models/#inlineformset-factory. [Pokušaj pristupa 1 rujan 2021].

[11] »How to set up environment variables in Django | by Alice Campkin | Medium,« [Mrežno]. Available: https://alicecampkin.medium.com/how-to-set-up-environment-variables-in-django-f3c4db78c55f. [Pokušaj pristupa 1 rujna 2021].

[12] »Amazon S3 — django-storages 1.11.1 documentation,« [Mrežno]. Available: https://django-storages.readthedocs.io/en/latest/backends/amazon-S3.html. [Pokušaj pristupa 1 Rujan 2021].

SAŽETAK

Naslov: Web aplikacija za rad ambulante opće prakse

Web aplikacija čija je svrha bila olakšati komunikaciju pacijenta s doktorom na način da je omogućeno slanje opisa tegoba putem interneta. Postoje tri moguće uloge u aplikaciji. Doktor ima najviše ovlasti i on može kreirati korisničku ulogu sestre i pacijenta. Medicinska sestra može printati nalaze, dodavati i uređivati podatke o pacijentu, ali ne može uređivati i unositi podatke o pregledu. Pacijent je uloga sa najnižim pravima. Pacijent može poslati opis tegoba sa datumom kada može pristupiti pregledu, a doktor mu može promjeniti taj datum ako mu ne odgovara. Pacijent, također može vidjeti svoje osobne podatke.

Ključne riječi: ambulanta, Django, medicina, rezervacija, web aplikacija

ABSTRACT

Title: Web application for the work of medical clinic

A web application whose purpose was to facilitate the doctor patient communication in such a way that it is possible to send a description of the problem over the internet. There are three possible roles in the application. The doctor role has the most authority and he can create a roles for nurse and patient. The nurse can print medical records, add and edit patients data, but cannot edit and enter examination data. The patient is the role with the lowest rights. The patient can send a description of the problem with the date when he can access the examination, and the doctor can change that date if it does not suit him. The patient can also see their personal information.

Keywords: clinic, Django, medicine, reservation, web application

ŽIVOTOPIS

Tin Anišić rođen je u Zagrebu 15. srpnja 1998. godine. Osnovnu školu Novska završio je u Novskoj. Istovremeno je pohađao Glazbenu školu u Novskoj i svirao gitaru. Svoje školovanje nastavlja u Tehničkoj školi Kutina u Kutini upisujući smjer tehničar za računalstvo. Nakon završetka srednjoškolskog obrazovanja upisuje Preddiplomski stručni studij Računarstvo na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija u Osijeku.