SADRŽAJ

1. Uvod

Web je postao nezaobilazan dio našeg svakodnevnog života. Bez obzira na to jeste li student, radnik ili umirovljenik, velika je vjerojatnost da svakodnevno koristite web za rad, učenje ili zabavu. Od jednostavnih stranica napisanih isključivo u HTML jeziku pa sve do kompleksnih web aplikacija sa različitim tehnologijama, web se razvio u nešto bez čega mi danas ne možemo zamisliti život.

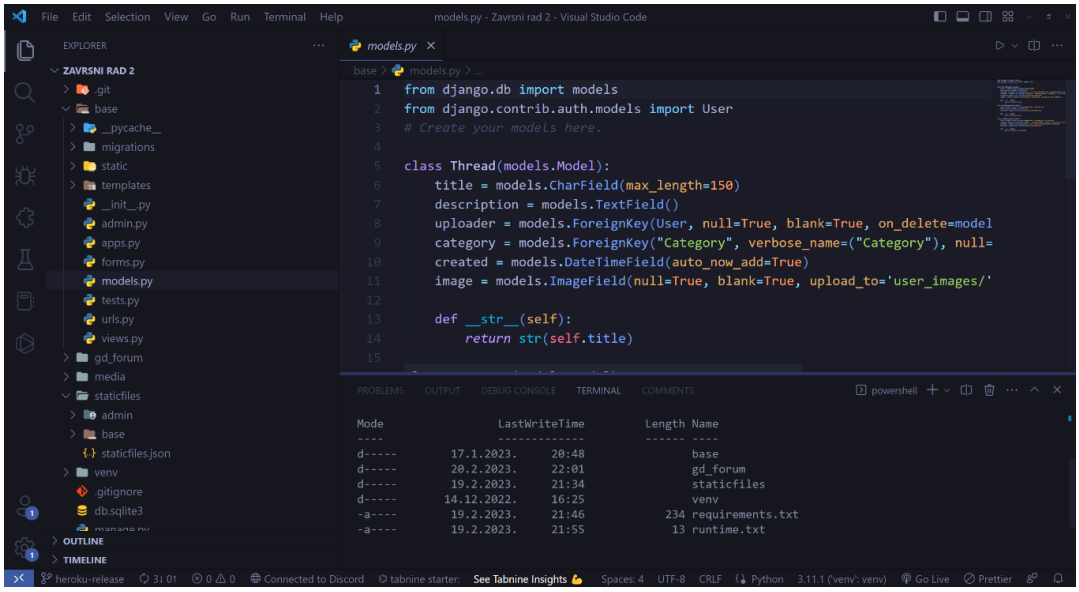
U ovom radu opisan je postupak kreiranja web aplikacije. Aplikacije u obliku foruma gdje posjetitelj imaju mogućnost registracije računa, stvaranje niti (engl. thread) koje sadržavaju tekst i/ili sliku, komentiranje na niti, mijenjanje svoje zaporke i još mnogo toga. Glavna tehnologija koja je bila korištena za izradu web aplikacije je Django. Django je open-source web okvir (engl. framework, u daljnjem tekstu: framework) koji omogućuje brzo i jednostavno razvijanje web aplikacija. On je temeljen na Python programskom jeziku i uključuje mnoge ugrađene funkcije koje omogućuju razvoj aplikacija za upravljanje bazama podataka, autentifikaciju korisnika, slanje e-pošte, i još mnogo toga. Django je poslužio kao glavni dio backend-a, no uz Django korišten je bio i Bootstrap za strukturiranje korisničkog sučelja. Bootstrap je vrlo popularan web framework koji olakšava izradu korisničkih sučelja. Jedna od glavnih prednosti Bootstrap-a je njegova responzivnost, što znači da je aplikacija izrađena s Bootstrap-om prilagođena različitim veličinama zaslona, od računala do mobilnih uređaja.

Uzimajući u obzir sve navedeno, cilj ovog rada bio je predstaviti Django u akciji i pružiti jasne primjere korištenja Django-a za razvoj web aplikacija. U radu su navedeni i detaljno opisani postupci pri stvaranju navedene web aplikacije.

2. Instalacija & konfiguracija

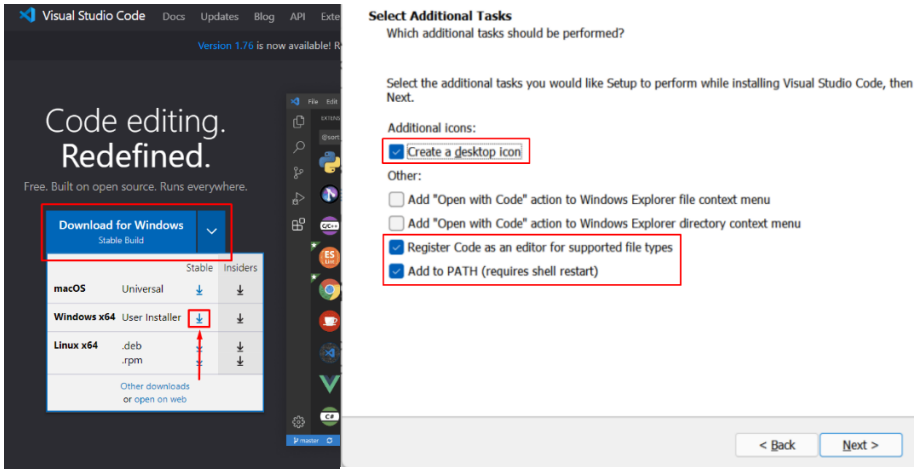
2.1 Integrirano razvojno okruženje

Integrirano razvojno okruženje (engl. Integrated development enviroment, IDE) je softverska aplikacija koji pruža programerima brojne alate za programiranje. Integrirano razvojno okruženje obično uključuje tekstualni uređivač (engl. text editor) i na njemu je provedeno najviše vremena tijekom izrade web aplikacije. Integrirano razvojno okruženje uključuje alate za debug-iranje koji su znatno pomogli pri identificiranju grešaka u kodu, brzo i efektivno. Integrirano razvojno okruženje koje je bilo korišteno u ovom radu je Visual Studio Code (Slika 1.) (kratica: VS Code, u daljnjem tekstu: VS Code). VS Code je besplatan i open-source, te jedan od najpopularnijih integriranih razvojnih okruženja za profesionalce i amatere.



Slika 1. VS Code korisničko sučelje

VS Code installer je bio preuzet sa stranice code.visualstudio.com. Tijekom instalacije označena su bila navedena polja na slici ispod. (Slika 2.).



Slika 2. VS Code instalacija

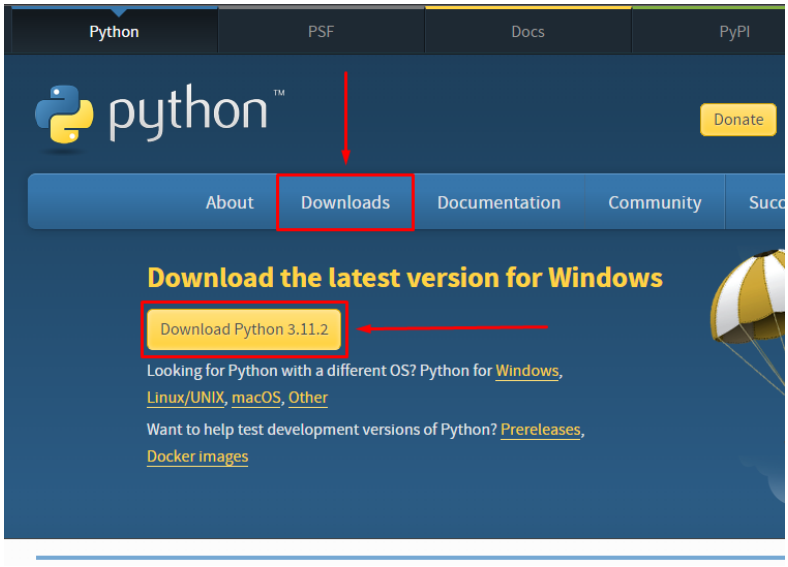
Nakon što je instalacija završena, bilo je nužno instalirati sve potrebne resurse za pravilnu funkcionalnost Django-a.

2.2 Python

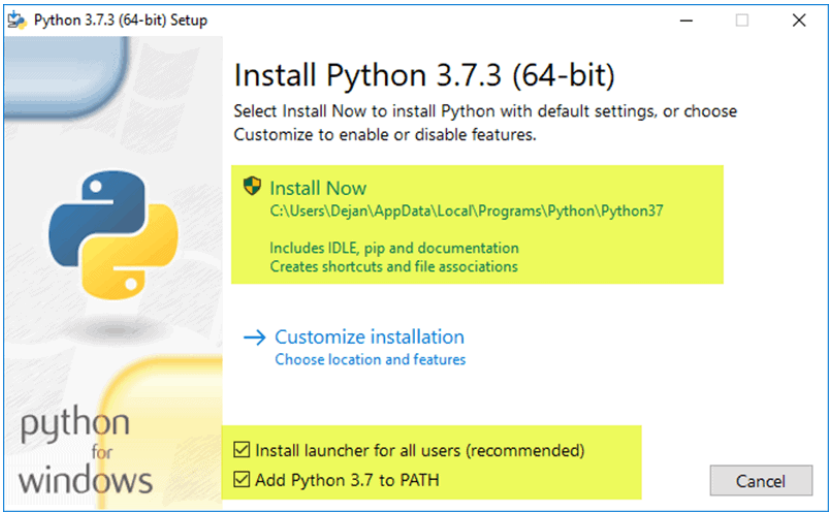
Python je popularni programski jezik za razvoj softverskih aplikacija. Njegova jednostavnost, čitljivost i velik broj modula čine ga popularnim izborom za različita područja pa i za web development. Budući da se Django temelji na Python programerskom jeziku, bilo ga je potrebno instalirati na računalo prije samog Djanga.

Prvo je provedena provjera postoji li Python instalacija već na računalu i to otvaranjem CMD-a (Naredbeni redak), te upisivanjem sljedeće komande: `python --version`. CMD je bio vratio grešku: `'python' is not recognized as an internal or external command, operable program or batch file`, što je značilo da Python nije instaliran na računalu.

Kako je na gore navedeni način utvrđeno da računalo nema instaliran Python, isti je preuzet sa službene stranice www.python.org (Slika 3.), te je pomoću čarobnjaka za instalaciju izvršena instalacija na računalo.



Slika 3. Gumb za Python installer (www.python.org)



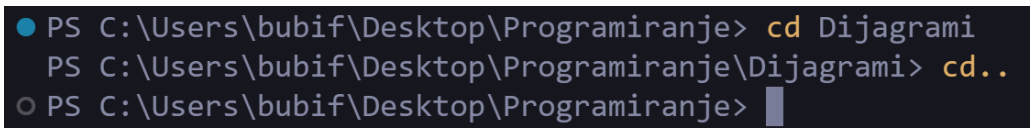
Slika 4. Python installer

Tijekom instalacije, označena su preporučena polja koja su se pojavila na dnu ekrana, kako bi se osigurao ispravan rad Python-a (Slika 4.). Nakon instalacije, računalo je resetirano na temelju preporuke čarobnjaka. Nakon uspješnog podizanja operacijskog sustava, otvoren je CMD te je upisana komanda: `python --version`. CMD je vratio ispis: `Python 3.11.1` što je značilo da je Python uspješno instaliran.

2.3 Django

Postoje dva načina na koje je moguće instalirati Django: globalno na računalo ili pomoću Python virtualnog okruženja. Ovom prilikom, instalacija Django-a provedena je pomoću Python virtualnog okruženja jer je to općenito bolja opcija. Virtualna okruženja omogućuju odvajanje Python paketa od ostalih globalnih instalacija na računalu. To je važno za testiranje novih funkcionalnosti paketa jer promjene u njima mogu uzrokovati pogreške u aplikacijama, a zbog nedostatka backward kompatibilnosti.

Virtualno okruženje kreirano je komandom iz terminala: `python -m venv venv`. Na početku komande, „`python`“ je bio govorio terminalu da koristi Python naredbe. „`-m`“ je bila skraćenica za mod, dok je „`venv`“ bio za „Virtual Enviroment“, te je na kraju navedeno ime okruženja što je bilo “`venv`”. Prije pokretanja komande, provjeren je trenutačni direktorij jer bi se u njemu kreiralo virtualno okruženje. U slučaju da je direktorij bio pogrešan, koristila bi se komanda „`cd`“ što znači „change directory“, te bi se nakon toga navelo ime direktorija koji bi bio poželjan (Slika 5.).



Slika 5. Primjer korištenja „cd“ komande.

Nakon izrade, okruženje je bilo aktivirano pomoću datoteke. Mjesto datoteke je bilo unutar okruženja, u „Scripts“ mapi, te je s komandom „`activate`“ bilo aktivirano. Kada je okruženje aktivirano, Django te ostali paketi bili su instalirani. Instalacija paketa provedena je pomoću „`pip install ime_paketa`“ komande. Dakle, Django je instaliran s „`pip install django`“. Komandom „`pip freeze`“ je provjerena lista instalacija, a koje su bile uspješne.

2.4 Pillow

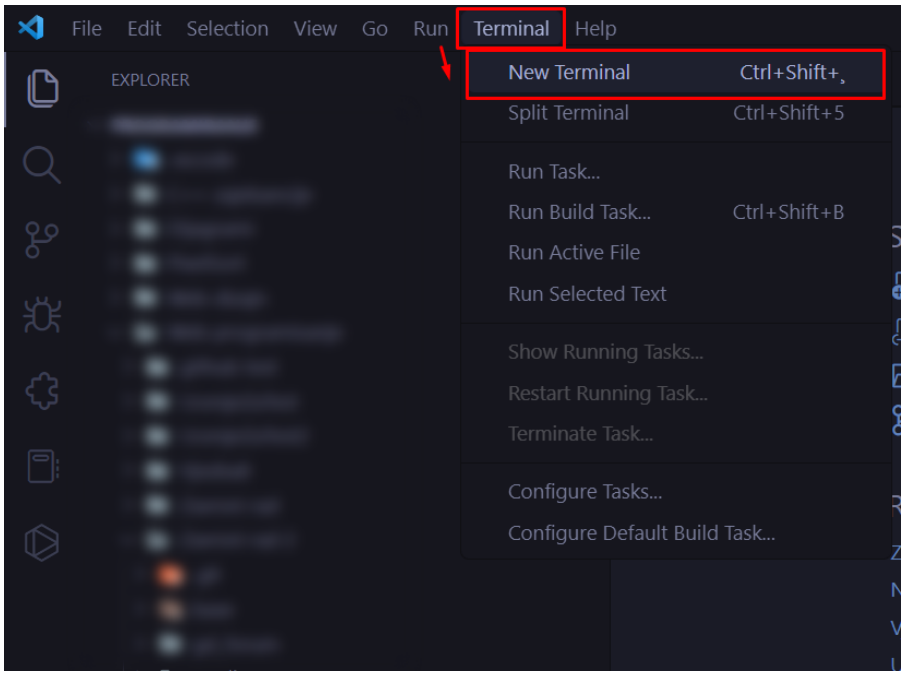
Pillow je Python paket za obradu slika koji pruža širok spektar funkcionalnosti za manipulaciju slikama, uključujući čitanje i pisanje slika u različitim formatima, kao što su JPEG, PNG i BMP, te razne vrste manipulacija slika poput skaliranja, rotacije, konvertiranja boja, i još mnogo toga. Pillow je bio korišten u aplikaciji da doda funkcionalnost objavljenja slika.

Pillow je bio instaliran kao i svaki drugi Python paket - `pip install pillow`, uzimajući u obzir da je virtualno okruženje bilo aktivirano.

3. Pripremanje projekta i aplikacije

3.1 Projekt

Nakon instalacije svih potrebnih resursa, VS Code je bio otvoren, te je bio otvoren novi integrirani terminal. (Slika 6.)

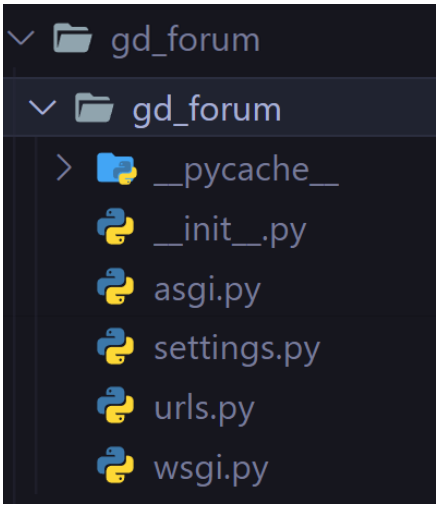


Slika 6. Otvaranje novog terminala pomoću navigacije

Dobra prednost VS Code-a jest njegova mogućnost da se terminal otvara direktno u njemu. To eliminira nužnost korištenja CMD-a s računala i znatno ubrzava rad.

Kada je VS Code terminal otvoren, navigacijom je odabrano željeno mjesto za izradu Django projekta i to korištenjem `cd` komande. Nakon pozicioniranja, napravljen je novi Django projekt korištenjem - `django-admin startproject gd_forum` komande. Naredba „`django-admin`" korištena je za izvođenje različitih podnaredbi, ovisno o zadatku koji je obavljen. U ovom slučaju s njom je započet Django projekt time što je nakon naredbe bio naveden „`startproject`", praćeno s nazivom projekta. Nakon izvedbe komande Django je kreirao mapu i u njoj potrebne datoteke za projekt.

Projekt je otvoren u VS Code-u, pomoću navigacijske trake s File → Open Folder. Kada je mapa otvorena, u VS Code explorer-u se javljalo nekoliko Python skripti. (Slika 7.)



Slika 7. Struktura Django projekta

Django je kreirao dvije mape s identičnim nazivima, prva mapa se odnosila na cjelokupni projekt sa svim aplikacijama i ostalim resursima koji su bili nužni za funkcionalnost cijele web aplikacije, dok se podmapa odnosila samo na projekt Django-a i datoteke nužne za funkcionalnost samog projekta. Većina vremena provedena je u prvoj mapi gdje je bila konfigurirana web aplikacija i resursi za nju, a druga je podmapa bila otvorena samo kada je trebalo konfigurirati projekt.

3.2 Aplikacija

Django omogućava stvaranje više aplikacija u jednom Django projektu, svaka aplikacija u tom projektu može imati svoj zadatak koji obavlja, npr. User aplikacija za menadžment korisnika, API aplikacija koja je zadužena za API i slično. Django-ove aplikacije su „pluggable“ što znači da se aplikacija iz jednog projekta može premjestiti u drugi bez prevelikih poteškoća. U ovom radu je napravljena samo jedna aplikacija koja je bila zadužena za sve zadatke.

U terminalu, navigiranjem u prvu mapu projekta upisana je komanda: `django-admin startapp base`. Aplikacija je nazvana „base“. U VS Code explorer-u stvorena je nova mapa pomoću Django-a – mapa aplikacije.

Otvaranjem datoteke „settings.py“ unutar podmape projekta, dodano je ime nove aplikacije u Python listu zvanu „INSTALLED_APPS“. Pošto je „INSTALLED_APPS“ bila Python lista, dodan je zarez na kraju navođenja aplikacije. Pred kraj „settings.py“ datoteke zamijenjena je varijabla „TIME_ZONE“ i postavljena na „'Europe/Zagreb'“, a radi osiguravanja točnog prikaza vremena unutar aplikacije. Na vrh import-an je bio „os“ modul iz Python-a, te su pred kraj datoteke bile dodane varijable „STATIC_URL“, „STATIC_ROOT“, „MEDIA_URL“ i „MEDIA_ROOT“ koje su definirale direktorije gdje će se statične i medijske datoteke spremati. Pomoću „os“ modula spajamo korijenski direktorij sa mapom. (Slika 8.).

```
STATIC_URL = 'static/'
STATIC_ROOT = os.path.join(BASE_DIR, 'staticfiles')
MEDIA_URL = "/media/"
MEDIA_ROOT = os.path.join(BASE_DIR, "media/")

STATICFILES_DIRS = [os.path.join(BASE_DIR, "static")]
```

Slika 8. Kod iz „settings.py“ za konfiguraciju lokacije statičnih datoteka

Unutar „urls.py“ datoteke u podmapi projekta import-ana je bila funkcija „include“ iz „django.urls“ lokacije kako bi URL-ovi aplikacije mogli biti pravilno usmjereni prema URL-ovima od projekta. Također, import-ane su bile funkcije „settings“ i „static“ iz lokacije „django.conf“.

Pritom, u listi „urlpatterns“ dodana je nova putanja. URL od aplikacije postavljen je bio direktno na glavnu domenu, te su pomoću „include“ funkcije, URL-ovi aplikacije bili spojeni sa URL-ovima od projekta. Nakon zatvorene zagrade od liste „urlpatterns“ dodana je funkcija „static“ koja je vraćala lokaciju gdje se spremaju datoteke da bi pritom, aplikacija znala gdje da ih pronađe. Kod upisan u „urls.py“ od projekta može se vidjeti na slici ispod (Slika 9.).

```python
from django.contrib import admin
from django.urls import path, include
from django.conf import settings
from django.conf.urls.static import static


urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('base.urls')),
] + static(settings.MEDIA_URL, document_root = settings.MEDIA_ROOT)
```

Slika 9. Kod iz „settings.py“ za konfiguraciju lokacije statičnih datoteka

Unutar mape imena „base“ stvorene su datoteke „forms.py“ i „urls.py“ pomoću VS Code navigacijske trake: File → New File. Unutar stvorene datoteke „urls.py“ import-ana je bila „path“ funkcija iz „django.urls“. Također, import-ani su bili „views“ iz trenutačnog direktorija gdje se „urls.py“ nalazi (Za import-anje iz trenutačnog direktorija koristi se znak „ **.** “). Nakon import-anja, dodana je bila „urlpatterns“ lista, te je ostavljena prazna. Unutar datoteke „forms.py“ import-an je bio „forms“ modul iz „django“ lokacije. Import-ani su bili obrasci „UserCreationForm“, „UserChangeForm“ i „PasswordChangeForm“ iz „django.contrib.auth.forms“ koji su pred definirani od Django-a, te će sa njima biti oblikovani obrasci za aplikaciju. Importan je bio „User“ model iz „django.contrib.auth.models“, te je importan „ModelForm“ sa lokacije „django.forms“ koja je služila kao predložak za svaki obrazac unutar „forms.py“

3.3 Predlošci

Predlošci (engl. templates, u daljnjem tekstu: templates) je mehanizam koji se koristi za generiranje dinamičkih HTML stranica iz raznih podataka. To su tekstualne datoteke koje sadrže HTML kod, te sadrže Django oznake (engl. tag, u daljnjem tekstu: tag) pomoću kojih se podaci dinamički ispisuju. Templates je koristan alat koji omogućuju programerima da ubace dinamičke podatke poput varijabli, petlji, uvjeta i drugih logičkih izraza.

Unutar mape aplikacije stvorena je bila mapa „templates“ te je unutar nje ubačena podmapa „base“, koja se odnosila na ime aplikacije. Unutar nje nalaziti će se HTML datoteke koje će biti zaslužne za prikazivanje frontend-a.

3.4 Bootstrap integracija

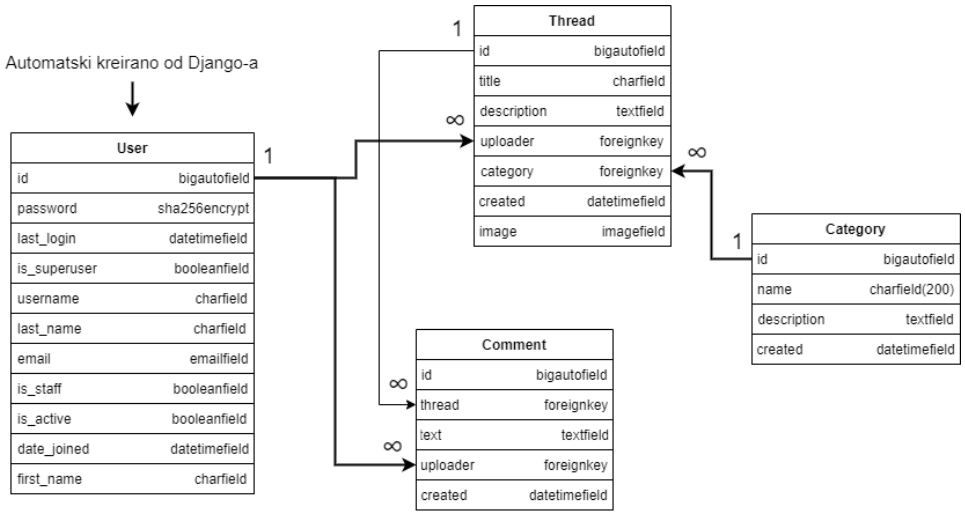
Pomoću Bootstrap-a dizajnirano će biti korisničko sučelje, no Bootstrap je trebao biti integriran kako bi sve funkcioniralo. Bootstrap-ove datoteke su bile preuzete sa Bootstrap-ove dokumentacije pod „Download“ sekcijom. U mapi „base“ stvorena je nova mapa „static“ i u njoj su ubačene mape „js“ i „css“ mape od Bootstrap-a.

4. Baza podataka

4.1 Priprema

Django je backend web okvir, što znači da za njega trebamo bazu podataka gdje će biti sadržane sve bitne informacije. Kada napravimo Django projekt, on već u sebi sadrži SQLite bazu podataka stoga nema potrebe dodavati vlastitu bazu, iako Django omogućava rad s ostalim bazama podataka kao što su MySQL, PostgreSQL, MariaDB i Oracle.

Prije konfiguriranja same baze podataka, bitno je imati plan i logiku u nekakvom slikovnom ili tekstualnom obliku (Slika 8.). U ovom radu baza podataka sadržava tri posebno kreirane tablice, te jednu tablicu iz Django-ovih pred definiranih tablica, a koja je relacijski spojena s kreiranim tablicama.



Slika 9. Dijagram baze podataka za završni rad

„User“ predstavlja korisnika na web aplikaciji dok „Thread“ predstavlja niti koje korisnik može objaviti, te je po strukturi web aplikacije, „Thread“ najbitniji model o kojem ovise svi drugi modeli. „Comment“ predstavlja komentare na nitima dok „Category“ predstavlja kategorije koje se mogu birati tijekom objave niti. Sve relacije između tablica su „One-to-many“ (1 - ∞).

4.2 Uvod u sintaksu

Django je u izrađenoj aplikaciji stvorio datoteku models.py i putem iste je ostvarivana komunikacija s bazom podataka. Modeli predstavljaju klase koje definiraju strukturu podataka u bazi. Django automatski generira SQL upite za stvaranje, čitanje, ažuriranje i brisanje podataka iz baze podataka, na temelju modela. Potom je otvorena models.py datoteka u VS Code editoru. Prvo, na vrh je import-an User model koji se već nalazi u Django projektu, pred definiran. Dakle, upisivanjem naredbe: „`from django.contrib.auth.models import User`“ ubačen je User model, tako da s njime može povezati ostale modele koji su u models.py.

```python
class Thread(models.Model):
    title = models.CharField(max_length=150)
    description = models.TextField()
    uploader = models.ForeignKey(User, null=True, blank=True, on_delete=models.CASCADE)
    category = models.ForeignKey("Category", verbose_name=("Category"), null=True, on_delete=models.DO_NOTHING)
    created = models.DateTimeField(auto_now_add=True)
    image = models.ImageField(null=True, blank=True, upload_to='user_images/')

    def __str__(self):
        return str(self.title)
```

Slika 10. Sintaksa modela

Modeli su definirani s „class“, te je nakon toga, navedeno ime modela i u zagrade stavljeno da nasljeđuje „models.Model“. Imena modela se obično pišu u jednini, to je industrijski standard i olakšava čitanje koda drugim programerima. Unutar klase postavljaju se atributi. Kod modela, atributi su mjesta u kojima se pohranjuju podaci u obliku neke određene vrste podataka, kao što su brojevi, datumi, nizovi znakova ili slično. U „models.py“ datoteci, atributi predstavljaju polja unutar baze. Poljima se mogu postavljati parametri koji konfiguriraju polje na način na koji mi želimo, te se svaki parametar odvaja zarezom. Nakon atributa, stavljaju se metode. Jedine metode koje su bile stavljene u ovom radu su „__str__“ metode, koje se koriste za definiranje „string“ reprezentacije objekta. Za primjer sintakse modela pogledajte sliku iznad (Slika 10.).

4.3 „Category“ model

Prvo je kodiran „Category“ model jer on ne sadrži polja koja su podatkovnog tipa: strani ključ, stoga je najlakši za razumjeti. Prvo, definirano je „name“ polje, za čiji je tip podataka odabran „CharField“. Postavljeno je njegovo znakovno ograničenje do 200 znakova (`max_length=200`), te je isti definiran kao jedinstven (`unique=True`). Nakon toga, definiran je opis kategorije i nazvan „description“, a za tip podataka odabran je „TextField“. Pri kraju, napravljeno je polje koje sprema podatak o datumu izradbe kategorije imena „created“ i tipa podataka „DateTimeField“, te je u zagrade dodan parametar „`auto_now_add=True`“, kako bi Django automatski postavljao trenutni datum i vrijeme tijekom izradbe kategorije. Kada su nadodana sva polja, dodana je i „__str__“ metoda kako bi model imao „string“ reprezentaciju. „__str__“ metode u Python-u su bile definirane s „def“, pri čemu se navodilo ime „__str__“ i u zagrade je bilo postavljeno nasljedstvo „self“. Pritom je bilo postavljeno da nam metoda vraća „string“ reprezentaciju polja, koji sprema zapise imena tog modela za lakše razumijevanje (`return str(self.name)`). Mogla su se navesti i druga polja koja su poželjna da nam metoda vraća u obliku „string-a“, no općenito se stavljaju kratka i jasna polja, poput polja koja pohranjuju zapise imena.

4.4 „Thread“ model

Pri vrhu, definirano je polje „title“ tipa podatka „CharField“ koje sprema zapise naslova niti, te je navedeno ograničenje do 150 znakova. Dodano je „description“ polje, tipa podataka „TextField“, te nakon njega dodano je i „uploader“ polje tipa „ForeignKey“. Kod „ForeignKey“ tipa podataka u zagradama je potrebno navesti naziv modela koji želimo spojiti s tim poljem. U ovom slučaju postavljen je „User“ model od Django-a. Također, u zagradama je nužno specificirati „on_delete“ parametar kojim se određuje što će se dogoditi sa zapisima kada se izbriše povezani model naveden u zagradama iz baze podataka. U ovom slučaju postavljen je bio „`on_delete=models.CASCADE`“ parametar, što je poručivalo bazi da, kada korisnik izbriše svoj korisnički račun moraju biti izbrisane i sve njegove niti. Pred kraj, pridodana su još dva dodatna parametra „`blank=True`“ i „`null=True`“. To je bilo napravljeno iz razloga da, u slučaju da bi postojao model u bazi, ali bez „ForeignKey“ polja, te se odluče dodati ta polja kasnije,

prilikom migracije, baza podataka bi morala nadodati sva nova polja u sve zapise koji su pohranjeni od prije. Iz tog razloga ti parametri moraju biti prisutni kako bi baza mogla popuniti prijašnje zapise s „null“ vrijednostima. Naredno polje je povezalo kategorije i niti postavljanjem naziva „category“ s tipom podataka „ForeignKey“ i s povezanim modelom „Category“. Postavljen je bio i „`on_delete=models.DO_NOTHING`“ parametar, jer kategorija ne bi smjela biti izbrisana iz baze kada se nit izbriše, te je na kraju dodano „`null=True`“. Dodano je bilo polje „created“ s tipom „TimeDateField“, te je u zagradama opet navedeno „`auto_now_add=True`“, kako bi Django automatski dodao trenutni datum u polje. Naposljetku, dodano je „image“ polje, tipa „ImageField“ s parametrima „`blank=True`“ i „`null=True`“. Pošto je ovo tip podatka „ImageField“ bilo je potrebno dodati još jedan obvezni parametar - mjesto na kojem će se pohranjivati slike, stoga je postavljen „`upload_to='user_images/'`“ parametar. Nakon što su bila navedena polja, postavljena je i „__str__“ metoda koja je vraćala „string“ reprezentaciju polja „title“.

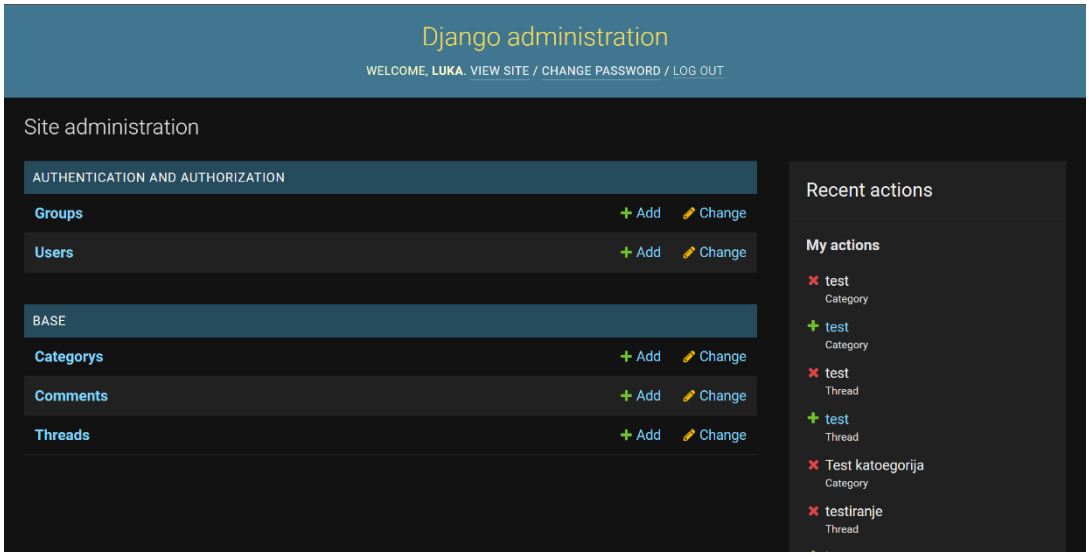
4.5 „Comment“ model

Dodano je „text“ polje, tip podataka „TextField“ koji ima ograničenje do 700 znakova, te je postavljanjem parametra „`blank=False`“ osigurano bilo da to polje ne smije biti prazno tijekom stvaranja. Potom je dodano „`null=False`“ kako bi se postavilo ograničenje da takva vrijednost ne može biti spremljena kao „null“ vrijednost u bazi podataka. Dodano je „thread“ polje tipa „ForeignKey“, unesen model „Thread“, te su postavljeni parametri „`null=True`“ i „`on_delete=models.CASCADE`“. Obzirom da komentare unose korisnici aplikacije bilo je potrebno dodati i polje za isto. To je učinjeno postavljanjem „uploader“ polja, tipa „ForeginKey“, unošenjem modela „User“ od Djanga, te postavljanjem parametara „`null=True`“ i „`on_delete=models.CASCADE`“. Potom je bilo dodano i „created“ polje tipa „DateTimeField“ i postavljen parametar „`auto_now_add=True`“. Na samom kraju stavljena je „__str__“ metoda, te je bilo postavljeno da vraća prvih 20 znakova od polja „text“. (`return str(self.text[0:20])`). Razlog zašto je bilo stavljeno samo prvih 20 znakova jest „text“ polje, sprema komentare korisnika. Komentari obično imaju puno znakova, radi uređivanja „string“ ispisa istih postavljeno ovo ograničenje.

4.6 Stvaranje zapisa

Kada su svi modeli bili kreirani, bila je nužna migracija kako bi se kod iz „models.py“ datoteke prenio na bazu. U terminalu, pomoću „`cd`“ komande, pozicioniralo se unutar mape gdje se nalazi „manage.py“ datoteka. Komandom „`python manage.py makemigrations`“ stvoreni su SQL upiti za ažuriranje baze podataka, te komandom „`python manage.py migrate`“ SQL upiti su bili izvršeni i uspješno su ažurirali bazu podataka.

Nakon migracije modela, komandom „`python manage.py createsuperuser`“ napravljen je bio administratorski račun iz razloga da se omogući pristup Django admin sučelju. Django admin sučelje je ugrađeno korisničko sučelje u Django web framework-u koje omogućuje administratorima aplikacije da upravljaju podacima u bazi i obavljaju druge administrativne zadatke. Sučelje se sastoji od preglednika koji omogućuje pregledavanje, dodavanje, uređivanje i brisanje podataka u bazi, te različitih alata za upravljanje korisnicima, grupama, dozvolama i drugim postavkama aplikacije (Slika 11.).



Slika 11. Django admin sučelje

Da bi Django pokazivao kreirane modele iz „models.py“ u admin sučelju, trebali bi biti registrirani u „admin.py“ datoteci unutar mape aplikacije. U „admin.py“ datoteci import-an je bio „admin“ modul iz „django.contrib“, te su bili import-ani modeli „Thread“,

„Category“ i „Comment“ iz „models.py“ datoteke. Nakon importanja, koristeći `admin.site.register()` metodu bila su dodana sva tri modela, time što su bili navedeni unutar zagrada od metode.

U terminalu je bila unesena komanda „`python manage.py runserver`“ i time je pokrenut lokalan poslužitelj koji poslužuje web aplikaciju na Internet priključku (port) 8000. Pomoću web preglednika unesena je bila poveznica: „127.0.0.1:8000/admin“, te je time otvoreno Django admin sučelje.

U Django admin sučelju prikazana su 5 modela: „Groups“, „Users“, „Categorys“, „Comments“ i „Threads“. Modeli „Groups“ i „Users“ su zadani su od Django-a, dok su ostala tri modela prikazana zbog registracije unutar „admin.py“ datoteke. Ulaskom u „categorys“ model, te klikom gumba „ADD CATEGORY +“ bilo je dodano nekoliko kategorija.

„Category“ model bio je zamišljen da bude većinski statičan, običan korisnik unutar stranice ne bi imao pristup stvaranju, brisanju ili izmjenjivanju zapisa od kategorija. Stoga, zapisi kategorija su smišljeni i dodani u bazu, svi odjednom.

5. Frontend dizajn

5.1 Osnovni elementi

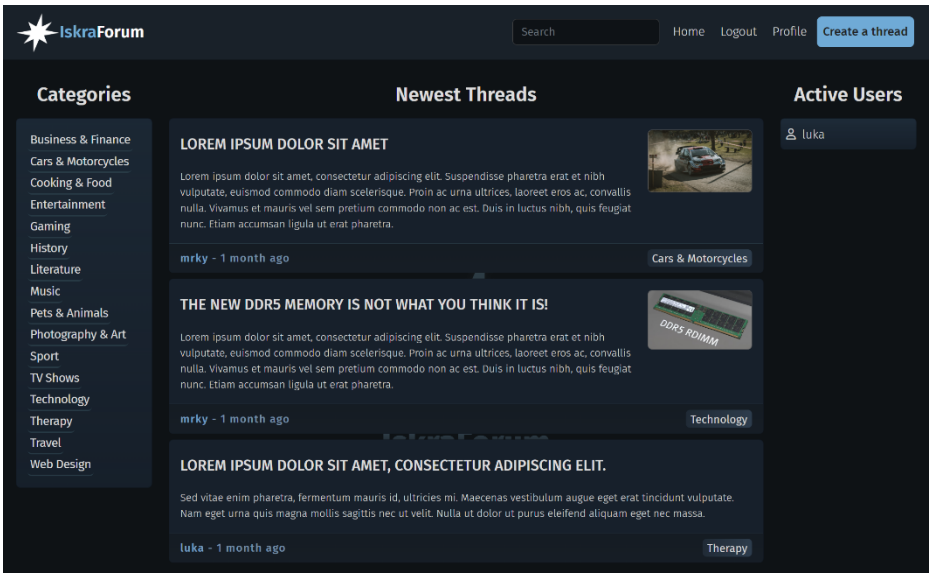
Važnost frontend-a u web developmentu je velika, jer predstavlja prvi dojam koji korisnici imaju o web aplikaciji. Ako korisnici ne mogu lako koristiti aplikaciju ili ako izgleda neprivlačno, vjerojatno će brzo odustati i otići na drugu web stranicu. Stoga je važno imati dobar i funkcionalan frontend koji će korisnicima pružiti ugodno iskustvo korištenja.

Korištenjem Bootstrap-a, frontend aplikacije je napravljen u modernom i jednostavnom stilu. Boje unutar aplikacije su bile tamne i sadržavale su dodir plavkaste boje. Osnovni elementi stranice dizajnirani su unutar „base.html“ datoteke. Svaka ostala template datoteka bila je povezana s „base.html“ datotekom pomoću Django-ove "extends" oznake.

Na svakoj podstranici aplikacije, gornji dio prikaza sadržavao je navigacijsku traku za kretanje po podstranicama aplikacije, te se na kraju sadržaja stranice, nalazilo podnožje gdje se centriranim tekstom navode autorska prava aplikacije i godinu izdanja.

Navigacijska traka je sadržavala logo, te ime aplikacije pored nje. S desna je sadržavala tražilicu, te gumbove „Home“, „Logout“, „Login“, „Register“, „Profile“ i „Create a thread“. Pomoću Django tag-ova postavljen je „if“ uvjet koji je prikazivao gumbove „Logout“, „Profile“ i „Create a thread“ ako je korisnik bio prijavljen, u suprotnosti, prikazivao je gumbove „Login“ i „Register“. Gumb „Home“ prikazan je bio u oba slučaja. Na pozadini cijele aplikacije nalazi se slika dizajnirana da popuni prazan prostor.

5.2 Glavna stranica

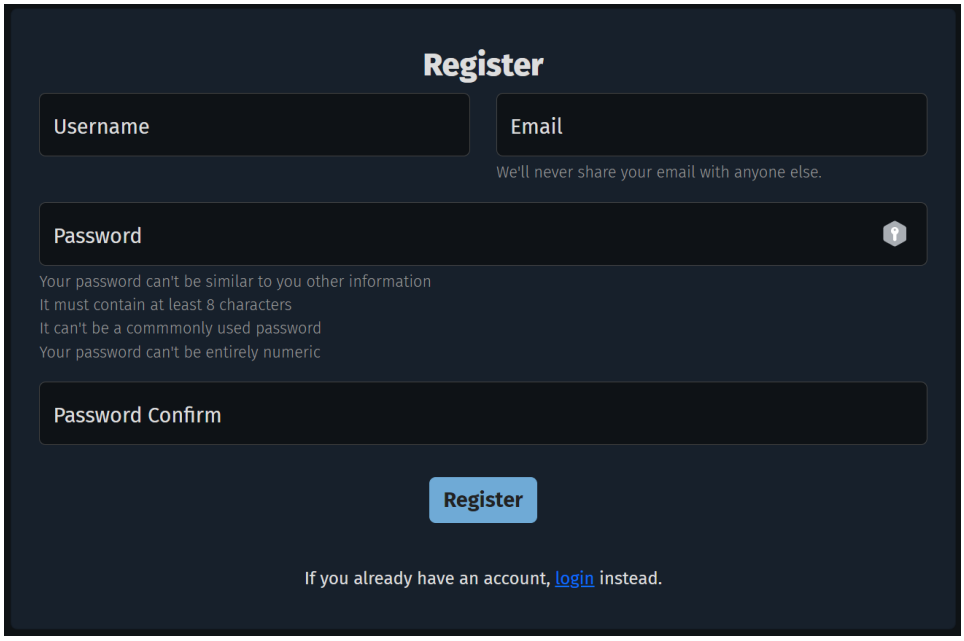


Slika 12. Dizajn glavne stranice

Glavna stranica sadržavala je tri prikaza (Slika 12.) „Categories“ „Newest Threads“ i „Active Users“. Posloženi su bili jedan pored drugog pomoću Bootstrap-ovog „Grid“ sustava. „Categories“ je planiran da ispisuje sve kategorije, „Newest Threads“ bi ispisivao sve niti napravljene na stranici poredani od najnovijeg prema najstarijem, dok bi „Active Users“ ispisivao sve korisnike koji trenutačno koriste stranicu.

5.3 Obrasci

Obrasci su se redovito pojavljivali unutar stranice, te su svi bili dizajnirani istim stilom (Slika 13.). Pomoću Bootstrap-ovih klasa „Floating label“ napravljena su polja unutar obrasca. Pomoćni tekst ispod polja napravljen je pomoću Bootstrap klase „form-text“, te su submit gumbovi dizajnirani s Bootstrap „btn“ klasama.



Slika 13. Dizajn obrasca

5.4 Niti

Niti unutar stranice dizajnirane su bile pomoću Bootstrap kartica. Unutar kartice nalazile su se klase „card-body“ i „card-footer“. Unutar klase „card-body“ nalazile su se glavne informacije o niti, a to su naslov, opis i slika, dok su se u „card-footer“ klasi nalazile informacije o autoru, datumu izrade i kategorija vezana uz nit. Ako je autor unutar podstranice vlastite niti, ispod kartice niti nalazila bi se dva gumba. Crveni gumb s ikonicom koša za smeće, je služio za brisanje niti, dok je drugi služio za izmjenjivanje polja unutar niti. Ispod niti, nalazio se prostor za komentare.

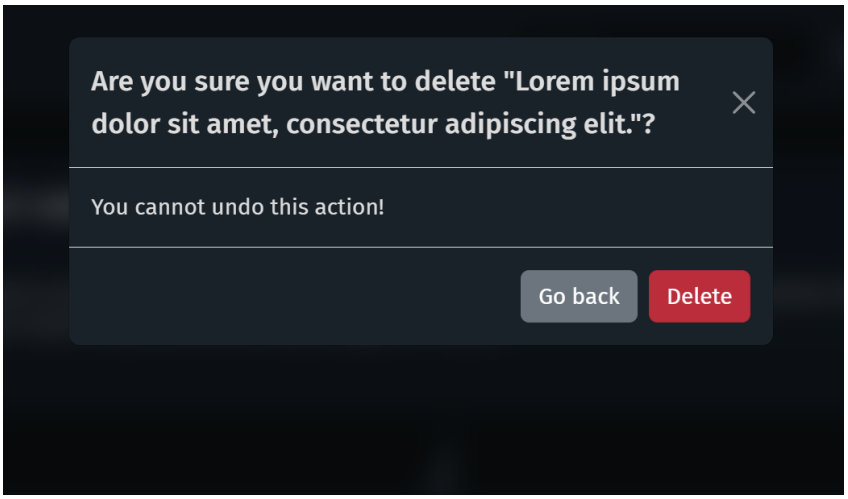
5.5 Komentari

Komentari su se prikazivali unutar „/thread/“ podstranice. Komentari su bili također dizajnirani pomoću Bootstrap kartica. Klase unutar same kartice bile su „card-header“ i „card-body“. „card-header“ klasa sadržavala je s lijeva, ime autora, i s desna, je sadržavala datum, dok je „card-body“ sadržavao tekst komentara. U slučaju da je korisnik bio registriran, iznad kartica komentara, nalazilo bi se polje za unos komentara, te ispod njega, gumb za slanje. Svi komentari napravljeni od korisnika koji se

podudarao s registriranim korisnikom, sadržavali su crveni gumb s desne strane kartice od komentara, za brisanje komentara.

5.6 Modal

Modal je funkcionalnost koja omogućava prikazivanje dinamičkih i interaktivnih prozora na web stranici. Modal u aplikaciji, se pojavljivao na vrhu korisnikovog preglednika i zamračivao bi pozadinu kako bi se fokus prebacio na sadržaj modala. U aplikaciji, bilo kakva radnja koja je uključivala brisanje nekakve informacije, sastojala se od modala kao sigurnosnog koraka i potvrdu od korisnika da li želi, ili ne želi, izbrisati navedenu informaciju (Slika 14.).



Slika 14. Dizajn modala

5.7 Korisnik

Prikaz korisnika postavljen je na podstranici „/user/*ime_korisnika*“. Na vrhu, stranica je prikazivala korisničko ime, te je ispod ispisivala sve niti stvorene od navedenog korisnika. U slučaju da se podstranica podudara s korisnikom koji je prijavljen, prikazana bi bila dva gumba ispod imena korisnika. Sivi gumb „Change username“ će mijenjati korisničko ime, dok će crveni gumb „Change password“ mijenjati korisnikovu zaporku.

6. Logika aplikacije

6.1 Registracija korisnika

Registracija korisnika za web aplikaciju je proces u kojem se korisnik prijavljuje za stvaranje računa kako bi mogao pristupiti različitim dijelovima navedene web aplikacije. Tijekom registracije, od korisnika će biti traženo da popuni polja „Username“, „Email“, „Password“ i „Password Confirm“ unutar obrasca koji će biti modelirani unutar „forms.py“ datoteke. Pri unosu svih podataka, računalo od korisnika će poslati „POST“ metodu nazad poslužitelju. Poslužitelj će provjeriti ispravnost informacije, te će na temelju toga registrirati korisnika i upisati zapis u bazu podataka ili javiti grešku te je ispisati na sučelju.

Povratkom u „forms.py“, napravljena je bila klasa „RegisterUserForm“ koja nasljeđuje pred definiran obrazac „UserCreationForm“. U ovoj klasi, dodano je polje „email“ te je definirano sa `forms.EmailField` koji prima parametre „`required=True`“ i „`widget=forms.EmailInput(attrs={'class': 'form-control', 'placeholder': 'e-mail'})`“ Prvi parametar je kazao da polje ne smije ostati prazno, a drugi je dodao HTML klase na polje pomoću kojih je frontend bio dizajniran. Unutar klase „RegisterUserForm“ dodana je bila podklasa „Meta“ koja se odnosila na metapodatke. U „Meta“ klasi definran je izvor metapodatka – `model=User`, te je dodana varijabla „fields“ koja je sadržila „`('username', 'email', 'password1', 'password2')`“. Varijabla „fields“ je određivala koja sva polja se smiju prikazati na korisničkom sučelju, jer u „User“ modelu od Django-a se nalaze polja koja neće biti korištena unutar aplikacije. Također, dodana je bila metoda „__init__“ koja je mijenjala izgled unutar polja obrasca, dodavajući odgovarajuće CSS klase i atribute pomoću „attrs“ parametra (Slika 15.).

```python
def __init__(self, *args, **kwargs):
    super(RegisterUserForm, self).__init__(*args, **kwargs)

    self.fields['username'].widget.attrs['class'] = 'form-control'
    self.fields['username'].widget.attrs['placeholder'] = 'username'
    self.fields['username'].widget.attrs['id'] = 'floatingInput'
    self.fields['password1'].widget.attrs['class'] = 'form-control'
    self.fields['password1'].widget.attrs['placeholder'] = 'password'
    self.fields['password1'].widget.attrs['id'] = 'floatingInput'
    self.fields['password2'].widget.attrs['class'] = 'form-control'
    self.fields['password2'].widget.attrs['placeholder'] = 'password confirmation'
    self.fields['password2'].widget.attrs['id'] = 'floatingInput'
```

Slika 15. CSS klase i atributi u obrascu za stilsko oblikovanje u frontend-u

Otvorena je bila datoteka „views.py“ unutar mape od aplikacije „base“. U datoteci import-ana je bila „redirect“ funkcija iz „django.shortcuts“. Iz „django.contrib.auth“ import-ane su bile funkcije „login“, „authenticate“ i „logout“, te je modul „messages“ bio import-an iz „django.contrib“. Pred kraj import-ani su bili svi modeli iz „models.py“, i „User“ model iz „django.contrib.auth.models“, te je bio import-an novostvoreni obrazac „RegisterUserForm“ iz „forms.py“ datoteke.

U prvom koraku, definirana je funkcija „register_user“ koja je bila pozvana prilikom pokušaja registracije korisnika. Zatim, provjereno je da li je zahtjev poslan s HTTP metodom „POST“ pomoću „if“ uvjeta. Ako jest, to je značilo da je korisnik poslao podatke putem web obrasca i potrebno ih je bilo obraditi. Ako je metoda slanja bila „POST“, instanca forme RegisterUserForm je bila inicijalizirana s podacima poslanim od strane korisnika pomoću „`request.POST`“ argumenta. Nakon toga, provjerena je bila ispravnost podataka pomoću metode „`is_valid()`“. Ako su podaci bili ispravni, korisnik je bio spremljen u bazu podataka pomoću metode „`save()`“, no ako nisu, pomoću funkcije `messages.error` poslana bi bila poruka na frontend koja bi obavijestila korisnika o grešci. Nakon spremanja korisnika, podaci su bili pohranjeni u varijablu „username“ i „password“, te je stvorena varijabla „user“ koja je ovjerila korisnika. Nakon autentifikacije, pozvana je funkcija „login“, te je pomoću `messages.success` funkcije, poslana poruka na frontend. Pred kraj, vraćena je funkcija „render“, te su joj pridodani parametri „request“, lokacija template-a koja će prikazati podatke – (base/register.html) i „context“.

"context" parametar se odnosi na Python rječnik (engl. dictionary) koji se koristi za prenošenje podataka iz views.py datoteke u template kojeg Django renderira.

U „urls.py“ od mape „base“ dodan je nova putanja koja se nalazila na „register/“ i poprimala je gore navedenu funkciju „register_user“, te je dodano ime - `name=‘‘register‘‘.`

Unutar datoteke „base.html“ na mjesto koda gdje je napravljen „Register“ gumb dodan je „href“ atribut i postavljena je putanja na „/register“, te je time registracija postala funkcionalna.

6.2 Prijavljivanje korisnika

Izrađena je nova funkcija – „login_user“, unutar „view.py“. Pomoću *if* izjave provjerava se je li korisnik već ovjeren s `request.user.is_authenticated,` ako je izjava točna vraća korisnika na „home“ stranicu jer korisnik koji je već prijavljen ne smije imati mogućnost prijave. Zatim, ako je zahtjev bio metoda POST, dohvaćeni bi bili podaci o korisničkom imenu i zaporki iz zahtjeva. Pomoću try/except bloka pokušaj je napravljen da se pronađe korisnika u bazi podataka čije korisničko ime odgovara onome koje je bilo uneseno u zahtjevu. Ako takav korisnik ne postoji, prikazivala bi se poruka o grešci s `messages.error.` Ako je korisnik s danim korisničkim imenom bio pronađen, pozvana bi bila se funkcija „authenticate“ da se napravi provjera je li unesena zaporka ispravna. Ako je zaporka bila ispravna, korisnik je bio prijavljen pomoću funkcije „login“, te je postavljena poruka o uspjehu i korisnika je preusmjerila na početnu stranicu. Pred kraj, vraćena je funkcija „render“, koja je primala „request“, login.html template, te prazan kontekst.

U „urls.py“ od mape „base“ dodana je nova putanja koja se nalazila na „login/“ i poprimala je funkciju „login_user“, te je dodano ime „login“.

Unutar datoteke „base.html“ na mjesto koda gdje je napravljen „Login“ gumb dodan je „href“ atribut s putanjom „/login“.

6.3 Odjavljivanje korisnika

Unutar „views.py“ napravljena je funkcija „logout_user“. U funkciji je pozvana funkcija „logout“ koja je primala „request“, te je funkcija vratila poruku o uspjehu s `messages.success` i preusmjerila je korisnika na početnu stranicu.

Dodana je nova putanja u „urls.py“ na „logout/“. Prima „logout_user“ funkciju i imenovana je „logout“.

Na „Logout“ gumbu u template-u dodan je „href“ atribut koji je sadržavao Django tag za „Python-like“ naredbe te unutar njih je napisano: „`url 'logout'`“. U navodnicima bitno je bilo upisati točan naziv koji je bio zadan putanji unutar „urls.py“ datoteke.

6.4 Mijenjanje korisničkog imena

Dodana je klasa „ChangeUsernameForm“ koja je poprimala „UserChangeForm“. Unutar nje, nalazila se podklasa „Meta“ koja je imala „model“ atribut u kojoj je pohranjena „User“ klasa, te atribut „fields“ koja je sadržavala samo „username“ polje. Dodane su Bootstrap klase pomoću „__init__“ metode kako bi mogle biti pravilno dizajnirane.

Unutar „views.py“ datoteke import-an je bio „login_required“ dekorater iz „django.contrib.auth.decorators“ , te novi obrazac iz „forms.py“. Napravljena je funkcija „edit_username“, te je iznad nje postavljen „login_required“ što je osiguralo da samo prijavljeni korisnici mogu pristupiti stranici za mijenjanje korisničkog imena.

Unutar funkcije provjereno je ako je metoda HTTP zahtjeva 'POST', ako je to istina, u varijabli „form“ se pohranjuju podaci od obrasca, te uz to se navodi instanca od trenutačno prijavljenog korisnika koja kazuje da taj korisnik već postoji i da se na njemu vodi operacija mijenjanja imena.

Prilikom bilo kakvog izmjenjivanja već postojeće informacije, mora biti navedena instanca, jer u protiv, obrazac će napraviti novog korisnika, te neće izmijeniti trenutačnog. Primjer instance može se pronaći na liniji koda 5 od slike ispod (Slika 16.).

```python
@login_required(login_url='login')
def edit_username(request):

    if request.method == 'POST':
        form = ChangeUsernameForm(request.POST, instance=request.user)
        if form.is_valid():
            form.save()
            messages.success(request, 'Username changed successfully!')
            return redirect('home')
        else:
            messages.error(request, 'This username already exists!')

    else:
        form = ChangeUsernameForm(instance=request.user)
        context = {
            'form': form,
        }
        return render(request, 'base/edit_user.html', context)
    return redirect('edit-username')
```

Slika 16. Funkcija „edit_username“

Zatim, je provjerena ispravnost informacije popunjene sa „`is.valid()`“ Ako je obrazac ispravan, korisničko ime će biti ažurirano, a korisnik bi bio preusmjeren na početnu stranicu uz poruku o uspješnoj promjeni korisničkog imena. Ako obrazac nije ispravno popunjen, vratila bi se poruka o grešci koja kazuje da to korisničko ime već postoji. Zatim, u slučaju da HTTP metoda nije „POST“ obrazac bi ostao prazan samo s instancom. Na kraju funkcije, vraćena je „render“ funkcija za template.

Unutar „urls.py“ datoteke dodana je putanja koja vodi do promijene korisničkog imena na lokaciji „edit-username/“ koja prima gore navedeni view, te je nazvan „edit-username“

U template datoteci za prikaz korisnika dodan je href atribut na sivi gumb „Change username“ te je u njega dodan Django tag *{%%}* koji je sadržavao ime URL-a za mijenjanje korisničkog imena.

6.5 Mijenjanje zaporke

Unutar „forms.py“ datoteke dodana je klasa „ChangePasswordForm“ koja je poprimala „PasswordChangeForm“ klasu. Unutar klase dodana je klasa za metapodatke koja je primala model „User“ te je u varijabli „fields“ imala vrijednost „__all__“, što je značilo da uzima sva polja iz klase koje nasljeđuje. Pomoću „__init__“ metode dodane su Bootstrap klase na svaki od polja unutar obrasca.

Unutar „views.py“ datoteke dodana je klasa „change_password“, te je iznad nje dodan dekorater „login_required“. U klasi, prvo je provjerena HTTP metoda „POST“, u slučaju da korisnik koristi metodu „POST“ poprimaju se podaci iz popunjenog obrasca te se unutar obrasca argumentiraju „`user=request.user`“ i „`data=request.POST`“. Nakon pohranjivanja podataka iz obrasca provjerava se njihova ispravnost, te ako jesu ispravni, ažuriraju zapis korisnika unutar baze te ispisuju poruku o uspješnoj promijeni i vraćaju korisnika na početnu stranicu. U slučaju da podaci nisu ispravni javlja se poruka o grešci. Pred kraj funkcije dodan je kontekst koji uzima „form“, te je dodana funkcija „render“ koja vraća template.

Dodana je putanja koja vodi na poddomenu „change-password/“ unutar „urls.py“ datoteke, te prima „change_password“ view i nazvana je „change-password“.

Naposlijetku, ažurirana je „user_view.html“ template datoteka, time što je dodan href atribut na crveni gumb za mijenjanje zaporke te je unutar atributa dodan Django tag koji sadrži URL ime „change-password“

6.6 Kategorije

Ispis kategorija napravljen je bio unutar glavne strance na aplikaciji, s lijeve strane. Unutar „views.py“ stvorena je funkcija koja se odnosila na glavnu stranicu, te je nazvana „home“. U funkciji dodana je varijabla „categories“ koja je pohranjivala ispis svih zapisa kategorija iz base poredani po imenu (`Category.objects.all().order_by('name'))`, te je ta varijabla definirana u „context“ varijabli pod istim nazivom. Pred kraj je vraćena „render“ funkcija s HTML template datotekom i kontekstom.

Unutar template datoteke od glavne stranice, na mjesto gdje su se trebale nalaziti kategorije dodana je „for“ petlja pomoću Django tag-ova te je glasila: `{% for category in categories %}`, što je značilo da petlja prolazi kroz sve unutarnje vrijednosti od „categories“, te svaku vrijednost koja se nalazi u varijabli pridružen je naziv „category“. Unutar petlje postavljeni su HTML tag-ovi koji su se stvorili prilikom svake instance jedne kategorije, te je unutar jednog tag-a postavljena dinamička

vrijednost – `category.name` te se s tom vrijednošću ispisivalo ime svake kategorije unutar varijable „categories“, na frontend-u.

6.7 Niti

U „views.py“ import-an je „Q“ iz „django.db.models“, te će sa njime biti ugrađen sustav za filtriranje niti po kategorijama i tražilici. U „home“ funkciji dodana je varijabla „q“ koja je imala vrijednost „`request.GET.get('q') if request.GET.get('q') != None else ''`“, te je dodana varijabla „threads“ koja je s „`Thread.objects.filter`“ filtrirala niti po kategoriji s kojom je bila spojena, s nazivom niti koja je unesena u tražilicu, i s imenom korisnika koji je objavio nit. Varijabla „threads“ je proslijeđena unutar konteksta, te je u template-u od početne stranice stavljena petlja koja ispisuje sve niti.

U „forms.py“ napravljena je klasa „ThreadForm“ koja je primala „ModelsForm“. U „meta“ podklasi je proslijeđen „Thread“ model, i u „fields“ varijabli uzima sva polja osim „created“ i „uploader“ jer ta polja Django automatski postavi pri stvaranju niti. S „__init__“ metodom su dodani svi potrebni atributi za dizajn obrasca.

U „views.py“ datoteci dodane su tri funkcije: „create_thread“, „edit_thread“ i „delete_thread“, te je importan novoizrađeni obrazac.

„create_thread“ funkcija se koristila prilikom izrade nove niti. U funkciji se provjerava metoda POST te se pomoću „ThreadForm“ obrasca spremaju podaci s `request.POST` i `request.FILES` te ako su podaci ispravni, spremaju se u bazu i preusmjerava korisnika na link gdje se nalazi novoizrađena nit.

Funkcija „edit_thread“, uz „request“ je primala i „pk“ parametar jer se s njime određuje koja se nit točno izmjenjuje u funkciji. Nit koja se izmjenjuje, pohranjena je bila u varijabli „thread“ s „`Thread.objects.get(id=pk)`“ te se s tom varijablom određuje instanca koja će biti ispisana unutar obrasca. Nakon toga, provjerava se POST metoda i ispravnost podataka unutar obrasca te ako su oba uvjeta točna, ažurira se zapis u bazi podataka i korisnika se preusmjeruje na mjesto niti koje je mijenjao.

„delete_thread“ slična je funkciji „edit_thread“ po tome što isto sadrži „pk“ parametar za određivanja koja nit se točno briše. U funkciji se provjerava uvjet POST te ako istinit, briše nit koja je definirana s „pk“ i preusmjerava korisnika na početnu stranicu.

6.8 Komentari

Komentari su se nalazili unutar prikaza svake od niti u aplikaciji, te je svaki korisnik mogao ostaviti komentar na nit ako je bio registriran i prijavljen.

Napravljena je „thread“ funkcija unutar „views.py“ datoteke koja je zaslužna za prikaz niti i komentara unutar nje. Funkcija prima „pk“ za definiranje specifične niti za prikaz te je nit pohranjena unutar „thread“ varijable. Komentari su pohranjeni u „comments“ varijabli pomoću `all()` funkcije. Postavljen je uvjet za POST te ako istinit, izrađuje novi komentar pomoću „`Comment.objects.create`“ koji sadrži parametre koji određuju koje informacije moraju bit pohranjeni prilikom izrade.

Za brisanje komentara, napravljena je funkcija „delete_comment“ koja prima „pk“ parametar za komentar i pomoću nje komentar se odredi i sprema u varijablu „comment“. S „if“ uvjetom provjerava se ako korisnik koji šalje „request“ je isti korisnik koji je napravio komentar, te briše komentar iz baze ako je uvjet istinit.

6.9 Ispis aktivnih korisnika

Ispis aktivnih korisnika napravljen je bio pomoću posebnog middleware-a. Middleware su komponente softvera koje se izvršavaju između web servera i Django aplikacije. Middleware se može koristiti za različite stvari, kao što su obrađivanje zahtjeva, modifikacija odgovora, provjera autentikacije i autorizacije, podešavanje sesija i sl.

Unutar mape od Django projekta – „gd_forum“, napravljena je datoteka „middleware.py“. Unutar datoteke ubačen je kod s Github-a od korisnika „agusmakmun“ koji se zvao „onlinenowMiddelware.py“.

Na liniji koda 6, izmijenjena je varijabla „ONLINE_TRESHOLD“, time što je zadnji broj bio izmijenjen na 10. Zadnji broj u navedenoj varijabli diktira koliko će minuta korisnik ostati prikazan na ekranu od trenutka kada je postao neaktivan.

U „settings.py“, na kraj liste „MIDDLEWARE“ je dodana putanja „`gd_forum.middleware.OnlineNowMiddleware`“. Pomoću toga, middleware je bio registriran u Django-u, te su bile pridodane nove dinamičke varijable u template tag-ovima, a to su: „`{{request.online_now}}`“, „`{{request.online_now_ids}}`“ i „`{{request.online_now.count}}`“

Middleware je bio importan unutar „views.py“, te unutar „home“ funkcije je dodana varijabla „users“ koja je primala vrijednost „`request.online_now`“ i proslijeđena je u kontekst. U template-u je napravljena „for“ petlja koja je ispisivala sve korisnike koji su bili aktivni.

7. Zaključak

Pomoću Django-a uspješno je razvijena stabilna i sigurna web aplikacija koja sadrži uobičajene funkcionalnosti koja se očekuje od web aplikacije po današnjim standardima. Django je bio koristan u razvoju aplikacije na nekoliko načina. Prvo, pružio je osnovnu strukturu i arhitekturu za aplikaciju koja je bila jednostavna za razumijevanje i korištenje. Django je također pružio iznimno koristan ORM (Object-Relational Mapping) koji je omogućio povezivanje aplikacije s bazom podataka na intuitivan način, te još mnogo toga.

Iako je Django napravljen da bude što lakše razumljiv početnicima, nailazio sam na dijelove tijekom razvoja aplikacije koji su mi bili teški za razumjeti. Najteži dio na koji sam nailazio pri razvoju aplikacije je razumijevanje logike unutar „views.py“ datoteke i razumijevanje migracija baze podataka i kako se obrađuju. Pošto nisam razumio osnove logike HTTP protokola imao sam poteškoće s POST metodom u „views.py“ i na koji način se podaci prenose preko njega. S modelima, nastao je problem pri migraciji u bazu, jer polja koja su bila tipa podataka „foreign key“ su dodana u modele nakon migracije, te pri novoj migraciji konzola je vraćala grešku da polja moraju biti postavljena na null da bi baza popunila sve prošle zapise unutar baze s praznim vrijednostima. Uzimajući u obzir sve probleme na koje sam naišao, cijeli tijek rada aplikacije prošao je bez prevelikih zastajanja zbog problema, iako je problema bilo, brzo su bili razriješeni.

Prilikom rada s Django web framework-om, usvojio sam temeljno znanje za rad backend-a u web aplikacijama, savladao sam osnove baze podataka i njihove relacije s aplikacijom, te sam savladao logiku CRUD operacija i njihovu implementaciju unutar „views.py“ datoteke. Nakon ovog rada, Django mogu koristiti bez problema u izgradnji svojih osobnih projekata.

Aplikacija je funkcionalna, no uvijek postoji prostora za poboljšanje i dodavanje novih funkcionalnosti. Na primjer, umetanjem modela za korisničke profile aplikaciji bi dodalo malo više života i personalizacije svojeg korisničkog računa, dodavanjem opcija za umetanje video i audio datoteka unutar niti, isto bi mnogo značilo za aplikaciju i korisnike. Web aplikacije su dinamične i konstantno se moraju nadograđivati da ostanu u skladu sa web standardima.

Na kraju, mogu zaključiti da je Django bio nezamjenjiv alat pri izradi ove web aplikacije. Pružio je framework za brz i jednostavan razvoj, laku integraciju s bazom podataka te veliki broj gotovih rješenja za česte probleme u izradi web aplikacija. Django je također omogućio proširivost aplikacije u budućnosti te osigurao sigurnost i pouzdanost u radu.

8. Izvori

1. „Django documentation“, Django Documentation, Django Software Foundation, https://docs.djangoproject.com/en/4.1/, 15.12.2022.
2. „Django tutorial“, Python Django 7 Hour Course, Traversy Media, https://youtu.be/PtQiiknWUcI, 16.12.2022.
3. „Bootstrap documentation“, Bootstrap Documentation, Bootstrap team, https://getbootstrap.com/docs/5.2/getting-started/introduction/, 20.12.2022.
4. „Active Users Middleware“, OnlineNowMiddleware, agusmakmun, https://gist.github.com/agusmakmun/c0c81ee21245578b1f13c003d1def401, 13.1.2023.
5. „Python installation“, How To Install Python 3 on Windows 10, phoenixNAP, Dejan Tucakov https://phoenixnap.com/kb/how-to-install-python-3-windows, 7.3.2023.
6. „Интегрисано развојно окружење“, Wikipedija https://sr.wikipedia.org/wiki/Интегрисано_развојно_окружење, 8.3.2023.
7. „Visual Studio Code installer“, Visual Studio Code, Microsoft https://code.visualstudio.com/, 9.3.2023.
8. „Python documentation“, Python Documentation, Python Software Foundation, https://docs.python.org/3/, 11.3.2023.

DATUM OBRANE ZAVRŠNOGA RADA

_____

PRIJEDLOG OCJENE ZAVRŠNOGA RADA

_____

KONAČNA OCJENA

_____

ČLANOVI ISPITNE KOMISIJE

1. _____
2. _____
3. _____