Univerzitet u Beogradu
Elektrotehnički fakultet

Bogdan Marković, 2020/0323 i Mirko Krstić, 2020/0194

Preporučivač stručne literature *bookMentor* - django web aplikacija realizovana korišćenjem algoritama mašinskog učenja

projekat iz predmeta Principi modernih telekomunikacija

mentor:
prof. dr Milan Bjelica

Beograd, jul 2023.

Sažetak

U vremenu kada je znanje opšte dostupno, a opet toliko ga je da se ne može na efikasan način selektovati, preporučivači pronalaze svoje bitno mesto kao alat u svakodnevnom životu. Preporučivače nesvesno svaki dan koristimo u raznim sferama redovnog života. ***bookMentor*** je napravljen upravo sa ciljem da na efikasan način, poznajući svoje korisnike - studente, i njihove krugove interesovanja ali i dosadašnje iskustvo, preporuči odgovarajuću literaturu kojom bi studenti mogli brže da napreduju na svom profesionalnom putu.

Ključne reči: stručne knjige, preporučivač, web aplikacija, mašinsko učenje, python, django, scikit-learn

Sadržaj

Spisak slika

Spisak tabela

Glava 1

Uvod

1.1 Cilj projekta

Glavni cilj projekta *bookMentor* je da bude koristan svim studentima svih smerova na Elektrotehničkom fakultetu u Beogradu. Koristnost leži u efikasnosti same web aplikacije, koja na jednostavan način u skladu sa afinitetima studenta, daje odgovarajuće stručne knjige kao preporuku.

1.2 Pozadina projekta

Iako radi naizgled jednostavan posao, implementacija samog projekta odnosno aplikacije iziskivala je korišćenje i kombinaciju nekoliko različitih oblasti Računarske nauke:

- Data engineering
- Database managment
- Web development
- Data science
- Machine learning

U narednim poglavljima je detaljno objašnjeno šta je koja oblast predstavljala konkretno u samom projektu, i na koji način je doprinela funkcionalnosti. Kako je projekat u potpunosti softversko rešenje, u prilogu odgovarajućih korišćenih programskih paradigmi će biti i delovi izvornog koda uz adekvatno objašnjenje. Iz datih razloga, struktura projektne dokumentacije je upravo takva da je tekst podeljen na gorepomenute oblasti Računarske nauke.

1.3 Dalji razvoj

Osnovna ideja za realizaciju ovog projekta je da on ne ostane projekat, već realan sistem u vidu hibridnog modela biblioteke sa preporučivačem, akademske društvene mreže i foruma. U sekciji *Buduća unapređenja projekta* biće detaljno razmatrani planovi za razvoj *bookMentor-a*.

Glava 2

Struktura projekta

U ovom poglavlju se kroz *Project flow (srp. Tok izrade projekta)* detaljno razmatra proces realizacije projekta u smislu glavnih projektnih odluka, odabranih programskih alata i inženjerskih kompromisa. Uz određene delove je prikazan odgovarajući izvorni kod, dok su negde prikazane tabele kao reprezent podataka. Kako projekat kombinuje različite oblasti Računarske nauke u kojima dominantnu ulogu ima programski jezik **Python**, odlučeno je da upravo on bude korišćen u najvećem delu projekta.

2.1 Dobavljanje i skladištenje podataka - Data engineering + Database managment

Osnovni problem i prva velika prepreka nakon izbora samog projekta je bio problem dobavljanja i skladištenja podataka. Kako je projekat osmišljen za korišćenje od strane studenata Elektrotehničkog fakulteta u Beogradu a sa ciljem preporučivanja stručnih knjiga, nametnula su se dva ključna pitanja:

1. Kako dobaviti knjige i na koji način ih čuvati
2. Kako prikazati sve dostupne kurseve sa Elektrotehničkog fakulteta u Beogradu i u kojoj formi

Kada se prave softverske infrastrukture koje iziskuju korišćenje velikih kolekcija podataka, jedan način za njihovo skladištenje je **interni popis** (ukoliko su stavke iz kolekcije ručno dostupne realizatoru projekta), ili **korišćenje javnih *API (Application Programming Interface)***, koji na HTTP zahtev (eng. HTTP request) dostavlja podatke u JSON formatu (ovaj način se koristi kada stavke iz kolekcije nisu dostupne ručno, ili ih uopšte ne poseduje realizator projekta). Iz fizički očiglednih razloga, drugi pristup je izabran za

korišćenje kod knjiga. Na tržištu je bilo dosta dostupnih javnih API-ja za dobavljanje podataka o knjigama, među kojima su se isticali Amazon Books, Packt Publishing i O'Reilly. S obzirom na to da je za neke od pomenutih API-ja bilo neophodno dobiti virtualni token za korišćenje, odnosno metod autentikacije radi unapređenja brzine slanja i primanja zahteva HTTP protokolom, nakon svih razmatranja odlučeno je da se koristi **O'Reilly Platform Search API**.

2.1.1 O'Reilly Platform Search API

Ovaj API jednostavnim HTTP zahtevom sa odgovarajućim parametrima dobavlja podatke u *JSON (JavaScript Object Notation)* formatu. Izabran je zbog obimnosti kolekcije stručne literature koju O'Reilly poseduje, a koja je publisher-independent, odnosno kolekcija poseduje knjige različitih izdavača. Ovo je posebno bilo od interesa za sam projekat jer rad sa većom kolekcijom podataka implicira kvalitetnije rezultate u samoj preporuci. Parametri korišćeni za dobijanje dostupnih knjiga su ujedno i podaci koji su bili neophodni za opisivanje svake knjige pojedinačno. Upravo ovi parametri će biti kolone u tabelama buduće Baze podataka, u kojoj će biti smešteno sve vezano za celu infrastrukturu projekta. U tabeli 2.1 je dostupan izgled strukture Book, čija svaka instanca svoje podatke dobija preko API-ja:

Tabela 2.1: Struktura/Tabela Book za opisivanje svake knjige u sistemu

Atribut	Tip podatka	Opis
isbn	Big Int	Identifikacija tabele
issued	DateTime	Trenutak izlaska knjige
authors	Text[]	Lista autora knjige
publishers	Text[]	Lista izdavača knjige
title	Text	Naslov knjige
description	Text	Opis knige u <html> formatu
average rating	Int	Prosečna ocena na O'Reilly platformi
popularity	Int	Ponderisana vrednost popularnosti knjige
cover url	Text	Hiper veza ka slici korice knige
topic	Text	Oblast pokrivena knjigom

2.1.2 Dobavljanje kurseva Elektrotehničkog fakulteta u Beogradu

Za dobavljanje kurseva fakulteta izabrana je prva metoda predstavljena na početku poglavlja, a koja se odnosi na interni popis dostupnih resursa. Kako su na sajtu fakulteta dostupni svi kursevi sa godinama i smerovima na kojima se polažu, proces je mogao da se obavi najjednostavnijim skladištenjem u CSV tip datoteke. Prilikom popisa, kursevima je ručno dodata lista oblasti koji pokrivaju, a u skladu sa listom oblasti dostupnih preko O'Reilly Platform Search API-ja. Na taj način je postignuta osnovna povezanost između kurseva i knjiga. U tabeli 2.2 je dostupan izgled strukture Course koja jednoznačno opisuje strukturu kurseva interno popisanih, a nakon toga unetih u Bazu podataka o kojoj će dodatno biti reči.

Tabela 2.2: Struktura/Tabela Course za opisivanje svakog kursa u sistemu

Atribut	Tip podatka	Opis
id	Int	Identifikator kursa
Kurs	Text	Ime kursa
o/i	Char	Obavezan/Izborni predmet
ER	Boolean	Da li je predmet sa prve godine smera ER
SIMinGod	Int	Najranija godina slusanja na SI
IRMinGod	Int	Najranija godina slusanja na IR
OSMinGod	Int	Najranija godina slusanja na OS
OTMinGod	Int	Najranija godina slusanja na OT
OGMinGod	Int	Najranija godina slusanja na OG
OFMinGod	Int	Najranija godina slusanja na OF
OEMinGod	Int	Najranija godina slusanja na OE
Topics	Text[]	Lista uređenih parova (Oblast:koeficijent)

Navedene strukture prikazane tabelarno reprezentuju tabele u Bazi podataka. Jasno je da se u svetu tehnologije sve odnosi na podatke i manipulaciju podacima, ali za sve to je potrebno da oni budu negde smešteni. Današnje baze podataka se mogu deliti na više načina, među kojima je osnovna podela na *relacione i nerelacione* baze podataka. Kako autori imaju više iskustva u radu sa relacionim bazama podataka koje koriste standardni *SQL (Structured Query Language)*, odlučeno je da se koristi jedna od najpoznatijih i najefikasnijih *open-source (srp. otvorenog koda)* baza podataka - **PostgreSQL**.

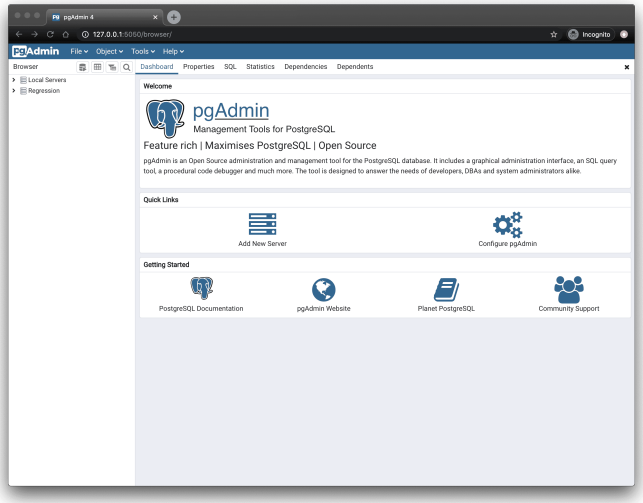
2.1.3 PostgreSQL Database Management System

PostgreSQL DBMS moderan je sistem za kreiranje i upravljanje relacionim bazama podataka započet na Univerzitetu Berkli 1996. godine. On je **predstavnik softvera otvorenog koda** (eng. open-source software). Implementiran je u niskim jezicima kao sto je C/C++ te je vrlo efikasan. Poseduje alate za rad sa različitim tipovima podataka čime ga ističe na tržištu DBMS-ova. Poseduje detaljnu dokumentaciju hijerarhijski struktuiranu radi lakšeg rada. Slogan projekta je *The World's Most Advanced Open Source Relational Database*.

Rad sa PostgreSQL-om (u daljem tekstu PG) moguć je sa nekoliko različitih nivoa pristupa:

- Grafičko korisničko okruženje **pgAdmin** kao najviši pristup
- Pristup iz komandne linije programom **psql**
- Pristup iz programskog koda putem odgovorajucih softverskih **drajvera**

Grafičko okruženje *pgAdmin* korisno je kada je potrebno vršiti ručne promene nad bazom, ili ukoliko je neophodno praviti pojedinačne stavke. Kada je potreban određen nivo automatizacije ili niskog pristupa kako podacima tako i meta podacima (podaci o samoj bazi i tabelama, polise i slično), najbolje je koristiti *psql* iz komandne linije.



Slika 2.1: *Izgled GUI pgAdmin aplikacije*

Treći pristup se odnosi na pristup bazi u programskom kodu višeg reda, kada se koristi tzv DB Driver, odnosno fasada između programera i koda za pristup bazi podataka. Driver omogućava otvaranje konekcije, čitanje podataka, brisanje podataka, menjanje podataka itd. Ovaj pristup je neophodan kada su potrebne sofisticirane operacije nad bazom. Primer je punjenje baze podacima iz API-ja, u prilozenom kodu 2.1.3.

Kao što je već napomenuto, većinski deo projekta je urađen u programskom jeziku Python, te je iz tog razloga korišćen specijalan driver napravljen baš za ovaj jezik - *psycopg2*. U nastavku je dat izvorni kod funkcija za dovlačenje knjiga u bazu, kao i kod funkcije za dovlačenje kurseva u bazu na osnovu CSV datoteke kurseva:

collectBookData.py

```
import json
import psycopg2
from urllib.request import urlopen
import time

"""
Python script to gather book data from O'Reilly Public Search
↪  API for local database

This data is used for implementation of recommendation system
↪  for electrical engineering students. It is combined
with the manually collected Courses data from faculty's site.
↪  All of data is completely for public use.

"""


def get_connection():
    try:
        conn = psycopg2.connect(  # enter your credentials for
        ↪  postgresql database
            host='',
            database='',
            user='',
            password=''
        )
```

```
        conn.autocommit = True
    except Exception as error:
        print('Error occurred while trying to connect to the
        ↪  database:')
        print(error)
        conn = None
    finally:
        return conn


def get_num_of_pages(number_of_books):
    return int(number_of_books / 200)


def get_api_name(topic):
    topic = topic.replace(' ', '%20')
    topic = topic.replace('+', '%2B')
    topic = topic.replace('#', '%23')
    topic = topic.replace('&', '%26')
    return topic


def get_api_url(topic, page):
    api_name = get_api_name(topic)
    url = 'exact url to call API'
    return url


def is_number(data):
    try:
        broj = int(data)
        return True
    except Exception:
        broj = 'Not a number honestly ;)'
        return False


def process_book(book):
    if 'isbn' not in book:
        book = None
        return book
```

```
    elif not is_number(book['isbn']):
        book = None
        return book
    elif 'title' not in book:
        book = None
        return book
    else:
        if 'issued' not in book:
            book.update({'issued': None})
        if 'authors' not in book:
            book.update({'authors': None})
        if 'publishers' not in book:
            book.update({'publishers': None})
        if 'description' not in book:
            book.update({'description': '<span>Learn more about
             ↪  this book on internet!</span>'})
        if 'average_rating' not in book:
            book.update({'average_rating': None})
        if 'popularity' not in book:
            book.update({'popularity': None})
        if 'report_score' not in book:
            book.update({'report_score': None})
        if 'cover_url' not in book:
            book.update({'cover_url': None})
        return book


def import_books(connection, books_list, page):
    cur = connection.cursor()
    iterBook = 0
    for book in books_list:
        try:
            book = process_book(book)
            if book is None:
                continue
            cur.execute(
                "INSERT INTO \"Books\" (isbn, issued, authors,
                 ↪  publishers, title, description,
                 ↪  average_rating, popularity,report_score,
                 ↪  cover_url, topic) VALUES (%s, %s, %s, %s,
                 ↪  %s, %s, %s, %s,%s, %s, %s)",
```

```
                (book['isbn'], book['issued'], book['authors'],
                ↪  book['publishers'], book['title'],
                ↪  book['description'],
                 book['average_rating'], book['popularity'],
                  ↪  book['report_score'], book['cover_url'],
                 book['topics_payload'][0]['name']))
            iterBook = iterBook + 1
        except Exception as error:
            print('ERROR')
            print(
                'Error occurred for the book ISBN:' +
                ↪  book['isbn'] + ' Title: ' + book['title'] +
                ↪  ' at the page ' + str(
                    page))
            print(error)
            continue
    cur.close()
    return iterBook


start_time = time.time()
file_tags = open("TagsJSON.json", "r")
json_data = json.load(file_tags)
file_tags.close()
lista = json_data["topics"]
topics = {}
for i in range(0, len(lista) - 1, 2):
    topics.update({lista[i]: lista[i + 1]})

conn = get_connection()
if conn is None:
    exit(0)

number_of_calls = 0
number_of_processed_books = 0
for topic_raw_name in topics.keys():
    number_of_books = topics.get(topic_raw_name)
    upper_bound = get_num_of_pages(number_of_books) + 1
    for page_count in range(0, upper_bound):  # iterate through
    ↪  all pages for that topic, and import all the books
        url = get_api_url(topic_raw_name, page_count)
```

```
            if number_of_calls % 15 == 0 and number_of_calls != 0:
            ↪  # we reached the limit for API calls
                print(str(number_of_processed_books) + ' books
                ↪  processed so far')
                time.sleep(10)
            response = urlopen(url)
            number_of_calls = number_of_calls + 1
            json_response = json.loads(response.read())
            books = json_response['results']  # list of books to be
            ↪  imported in the database
            number_of_processed_books = number_of_processed_books +
            ↪  import_books(conn, books, page_count)

conn.close()
print('Time of execution: ' + str(time.time() - start_time))
```

U narednom listingu 2.1.3 je prikazan kod kojim se manualno prikupljeni podaci o Kursevima na fakultetu zapisani u CSV formatu, prebacuju u Bazu podataka.

importCSV.py

```
import psycopg2
import csv
import time

"""
Python script to import all data from Courses csv file. CSV
↪  file was made completely manually, and is available
in github repo.

"""


def getTopics(topicsString):
    ret = topicsString.split(',')
    if ret[-1] == "":
        return ret[:-1]
    else:
        return ret


def getFields(row):
    fields = []
    fields.append(row['\ufeffID'])
    fields.append(row['Kurs'])
    fields.append(row['o/i'])
    fields.append(row['ER'] if row['ER'] != "" else None)
    fields.append(row['SIMinGod'] if row['SIMinGod'] != "" else
    ↪  None)
    fields.append(row['IRMinGod'] if row['IRMinGod'] != "" else
    ↪  None)
    fields.append(row['OSMinGod'] if row['OSMinGod'] != "" else
    ↪  None)
    fields.append(row['OTMinGod'] if row['OTMinGod'] != "" else
    ↪  None)
```

```
    fields.append(row['OGMinGod'] if row['OGMinGod'] != "" else
    ↪  None)
    fields.append(row['OFMinGod'] if row['OFMinGod'] != "" else
    ↪  None)
    fields.append(row['OEMinGod'] if row['OEMinGod'] != "" else
    ↪  None)
    fields.append(getTopics(row['Tagovi']))
    return fields


start_time = time.time()
conn = None
print('Testing connection...')
try:  # enter db credentials
    conn = psycopg2.connect(
        host='',
        database='',
        user='',
        password=''
    )

    cur = conn.cursor()
    csv_file = open('CoursesProbaCSVa.csv')  # set the right
    ↪  name for the file
    reader = csv.DictReader(csv_file)
    for row in reader:
        fields = getFields(row)
        cur.execute(
            "INSERT INTO \"Courses\"
            ↪  (\"id\",\"Kurs\",\"o/i\",\"ER\",\"SIMinGod\",\"IRMinGod\",\"OSMi
            ↪  \
            \"OFMinGod\",\"OEMinGod\",\"Topics\") VALUES (%s,
↪   %s, %s, %s, %s, %s, %s, %s,%s, %s, %s, %s)",
            (fields[0], fields[1], fields[2], fields[3],
            ↪  fields[4], fields[5], fields[6], fields[7],
            ↪  fields[8],
             fields[9], fields[10], fields[11]))
    conn.commit()
    cur.close()
    csv_file.close()
except Exception as error:
```

```
    print('Error:')
    print(error)
finally:
    if conn is not None:
        conn.close()
        print('Successfully disconnected. It took ' +
        ↪  str(time.time() - start_time) + 'seconds')
```

2.2 Web aplikacija u django frameworku - Web development + Data Science

Nakon obavljenog Data Engineering-a, podaci se nalaze u PG Bazi podataka i spremni su za korišćenje. Naredno pitanje u Project flow-u bilo je kako manipulisati podacima, odnosno koji tip infrastrukture treba koristiti da bi bio što bolji *UX (User Experiance, srp. Korisničko iskustvo)*. Kada se govori o infrastrukturi projekta, govori se o *programu nosiocu*. U ovom slučaju je to aplikacija (Desktop/Web) realizovana u odgovarajućem framework-u. U razmatranju su bile sledeće opcije:

- Native desktop Python GUI pristup (npr. Tkinter)
- MEAN/MERN pristup
- Python based web frameworks

Prva od navedenih varijanti je najteže izvodljiva zbog *niskog nivoa pristupa (eng. low-level)* preko psycopg2 DB Driver-a, ali i činjenice da je u pitanju desktop aplikacija. Iako su desktop aplikacije robusnije, njihova proizvodnja opada već godinama zbog platformske zavisnosti. Upravo platformska zavisnost kao mana implicira platformsku nezavisnost kao vrlinu, što danas imaju apsolutno sve web aplikacije. Iz tog razloga, naredna dva pristupa su bila mnogo interesantnija pri izboru tipa infrastrukture.

MEAN/MERN je skraćenica koja označava pristup Web development-u preko sledećih tehnologija:

1. **MongoDB**, nerelaciona baza podataka
2. **Express.js**, framework za stvaranje RESTful API-ja
3. **Angular** / **React.js**, frontend web framework
4. **Node.js**, backend web framework

Sa ovim skupom tehnologija je omogućena visoko kvalitetna proizvodnja web aplikacija široke namene. U pitanju su tehnologije široko prihvaćene i poprilično tražene na tržištu Web developmenta. Ipak, jedna stvar se nameće kao problem pri izboru u projektnoj infrastrukturi - glavni jezik koji se koristi u MEAN-u je **JavaScript**, dok je glavni jezik odabran za većinu rešenja u projektu jezik Python. Iako je tehnički izvodljivo koristiti nekoliko različitih jezika u proizvodnji nekog softverskog rešenja, preporuka je raditi sa onim jezicima i paradigmama koje su lako uparljive kako fizički tako i

za programera koji je iza projekta. Iz tog razloga, nametnula se potražnja za **Python based web frameworkom** koji bi obavio isti posao koji bi se odradio korišćenjem MEAN pristupa.

Najpoznatiji takvi framework-ci su ***Flask* i *django***. Analiza i biranje "boljeg"između dva frameworka iziskuje veliki napor, jer oba pružaju veliku kontrolu i dosta dobrih mogućnosti. Kombinuju *backend i frontend*, odnosno manipulaciju serverskom i klijentskom stranom.

Možda i najvažnija projektna odluka bila je izabrati kvalitetan i odgovarajući framework koji moze da podrzi sve zahteve navedene u samom početku projektnog teksta. Upravo što je dobar za izradu sofisticiranih web aplikacija, **django** je izabran kao nosilac cele infrastrukture. I to nije slučajno, struktura frameworka je idealna za potrebe projekta upravo zbog skoro idealne povezanosti sa PostgreSQL bazom podataka.

Slika 2.2: *Logo django framework-a*

Primer jedne od viših funkcionalnosti koja je obezbeđena iz ove kvalitetne povezanosti je integracija PG Text Search-a, koji predstavlja dubinsku pretragu po nekoliko (specificiranih) polja u bazi, što je u ovom projektu iskorišćeno za detaljnu pretragu knjiga. Ovo je samo jedan od benefita ove veze koji je uticao na krajnju odluku, a ima ih mnogo o kojima će biti tek reči.

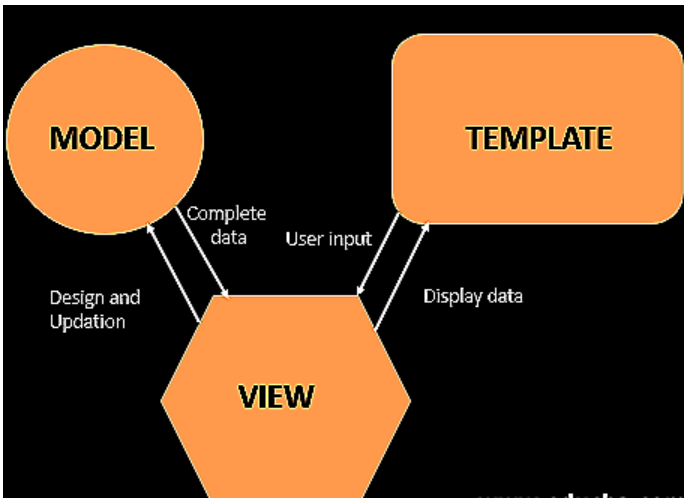
2.2.1 Framework za izradu web aplikacija - django

Kao što je prethodno napomenuto, integrisanost django-a sa PostgreSQL-om stavila ga je na mesto idealnog okvira za rad. Iako složen, ovaj framework ne predstavlja problem da se savlada nakon nekog vremena zbog dokumentacije koja je napisana tako da za svako pitanje postoji odgovor. Zajednica ljudi koji ga koriste je poprilično aktivna čime su dostupni skoro svi odgovori na sva postavljena pitanja.

Struktura framework-a se zasniva na **MVT (Model-View-Template)** pristupu, gde je neophodno spomenuti sledeće stvari:

- **Modeli** -> Klase koje opisuju tabele u bazi podataka na koju je povezan framework
- **Pogledi** -> Interfejsi ka krajnjim korisnicima. Funkcije koje pozivaju odgovarajući HTML sadržaj
- **Templejti** -> Statički HTML+CSS+JS fajlovi koje pretraživač renderuje kada se Pogled izvrši

Način na koji komuniciraju je prikazan na slici:



Slika 2.3: *MVT Struktura django framework-a*

Pored MVT-a, važno je spomenuti da django ima fasadu između stvarnog pristupa bazi podataka (preko psycopg2) i programera koji koristi podatke. Ta fasada se naziva **django ORM (Object Relational Mapping)**, i predstavlja elegantan način da se dobiju podaci, da se njima manipuliše i slično.

django ORM koristi klase Modele, interfejs Manager i dr. kako bi se podacima pristupalo kao da su najobičniji objekti *OOP (Objektno-orijentisano Programiranje)* paradigme. Na taj način je višestruko ubrzan proces rada, ali i olaksan rad programeru.

Rad u frameworku je maksimalno dekomponovan sa ciljem lakše hijerarhijske organizacije. To znači da svaka zasebna funkcionalnost ide u specijalne podgrupe koje se nazivaju aplikacijama. Može se reči da je framework podeljen u aplikacije (uz naznaku da su to u stvari podaplikacije) koje međusobno interaguju i koriste podatke. Svaka aplikacija sadrži sledeće fajlove neophodne za rad (ali i još neke koji nisu navedeni):

- Direktorijum **migrations** -> sadrži fajlove koji pamte sve promene nad strukturom modela kako bi u slučaju greške moglo da se pređe na neko prethodno stanje
- **admin.py** -> konfiguracioni fajl u kojem se opisuju modeli u adminskoj stranici
- **forms.py** -> fajl koji sadrži forme koje se koriste u datoj aplikaciji. Forma u ovom slučaju predstavlja klasu (može biti i metoda) koja se translira u HTML formu pri komunikaciji Templejta i Pogleda
- **models.py** -> fajl u kojem se nalaze svi modeli korišćeni za datu aplikaciju. Modeli su klase koje preko django ORM-a mapiraju podatke u bazi podataka
- **urls.py** -> fajl u kojem se nalaze sva mapiranja url adresa za datu aplikaciju
- **views.py** -> fajl u kojem se nalaze svi Pogledi o kojima je bilo reči pri komentarisanju MVT modela koji django koristi

Kako bi se lakše objasnili gorepomenuti pojmovi, najlakše je objasniti ih na primeru u projektu. U okviru projekta su korišćene 2 aplikacije:

- books -> sa modelima Book, Course
- accounts -> sa modelima Profile, GradedCourse

Neka se posmatra aplikacija *books*. Njena svrha je prikazivanje podataka u obliku pogodnom za korišćenje celokupne infrastrukture. To podrazumeva listing svih knjiga, detaljan pregled pojedinačnih knjiga, listing svih kurseva i detaljan pregled pojedinačnih kurseva. Ukoliko je korisnik autentikovan na sistemu, omogućeno mu je da knjigu doda u svoju kolekciju knjiga koje je pročitao. Detalji o sistemu autentikacije će kasnije biti predstavljeni.

U fajlu **admin.py** su konfigurisani Book i Course modeli kako bi mogli da se u odgovarajućem obliku predstave na adminskoj stranici. Adminska stranica predsavlja built-in funkcionalnost frameworka koji superuser-u (a to je adminski profil) omogućava da na način na koji on to zatraži framework-u (posredstvom admin.py fajla) vidi sve zapise u bazi reprezentovane Modelima.

U fajlu **models.py** su, kao što je već rečeno, svi modeli koji reprezentuju određebe tabele iz baze. Konkretno, modeli koji se ovde nalaz su *class Book i class Course*. Stvaranje ovih modela je bilo specifično sa tačke gledišta samog framework-a s obzirom na to da ovi modeli nisu prvo napisani u Python-u, već su uvezeni iz same baze. Ova specifična situacija se naziva *Stvaranje modela na osnovu tabela iz legacy baze podataka*, i iziskuje pomoć paketa potprograma django framework-a ali i ručne prepravke. Ova sitauacija predstavlja tzv. *reverse engineering (srp. obrnuti inženjering)* jer se inače tabele stvaraju na osnovu modela a ne obrnuto. Ipak, ovde je sitacija bila obrnuta, pa se koristio potprogram **inspectdb**. Naredni listing 2.2.1 pokazuje izgled ovog fajla jer je od interesa za razumevanje celog projekta.

```
from django.db import models
from django.urls import reverse


class Book(models.Model):
    isbn = models.BigIntegerField(primary_key=True)
    issued = models.DateTimeField(blank=True, null=True)
    authors = models.TextField(blank=True, null=True)  # This
    ↪  is a list of authors
    publishers = models.TextField(blank=True, null=True)  #
    ↪  This is a list of publishers. Usually there's just
    ↪  one.
    title = models.TextField(blank=True, null=True)
    description = models.TextField(blank=True, null=True)  #
    ↪  HTML text to display
    average_rating = models.IntegerField(blank=True, null=True)
```

```
    popularity = models.BigIntegerField(blank=True, null=True)
    report_score = models.IntegerField(blank=True, null=True)
    cover_url = models.TextField(blank=True, null=True)  # Link
    ↪  where cover is stored
    topic = models.TextField(blank=True, null=True)  # This is
    ↪  a tag that distinguishes the book from other books

    def get_absolute_url(self):
        return reverse('books:book_detail', args=[self.isbn])

    class Meta:
        db_table = 'Books'
        ordering = ('-issued',)

    def __str__(self):
        return self.title


def get_courses_by_module(module, year, profile):
    if module is None or year is None:
        return Course.objects.all()
    if module == 'er':
        return
        ↪  Course.objects.all().filter(er=True).exclude(gradedcourse__profile=p
    elif module == 'si':
        return
        ↪  Course.objects.all().filter(simingod__lte=year).exclude(gradedcourse
    elif module == 'ir':
        return
        ↪  Course.objects.all().filter(irmingod__lte=year).exclude(gradedcourse
    elif module == 'os':
        return
        ↪  Course.objects.all().filter(osmingod__lte=year).exclude(gradedcourse
    elif module == 'ot':
        return
        ↪  Course.objects.all().filter(otmingod__lte=year).exclude(gradedcourse
    elif module == 'og':
        return
        ↪  Course.objects.all().filter(ogmingod__lte=year).exclude(gradedcourse
    elif module == 'of':
```

```
            return
             ↪ Course.objects.all().filter(ofmingod__lte=year).exclude(gradedcourse
        elif module == 'oe':
            return
             ↪ Course.objects.all().filter(oemingod__lte=year).exclude(gradedcourse
        else:
            return Course.objects.all()


class Course(models.Model):
    CHOICES = (
        ('o', 'Obavezan'),
        ('i', 'Izboran'),
    )
    id = models.IntegerField(primary_key=True)
    kurs = models.TextField(db_column='Kurs')  # Field name
     ↪ made lowercase.
    o_i = models.CharField(db_column='o/i', blank=True,
                           null=True, choices=CHOICES,
                           default='i')  # Field renamed to
                            ↪ remove unsuitable characters.
    er = models.BooleanField(db_column='ER', blank=True,
     ↪ null=True)  # Field name made lowercase.
    simingod = models.IntegerField(db_column='SIMinGod',
     ↪ blank=True, null=True)  # Field name made lowercase.
    irmingod = models.IntegerField(db_column='IRMinGod',
     ↪ blank=True, null=True)  # Field name made lowercase.
    osmingod = models.IntegerField(db_column='OSMinGod',
     ↪ blank=True, null=True)  # Field name made lowercase.
    otmingod = models.IntegerField(db_column='OTMinGod',
     ↪ blank=True, null=True)  # Field name made lowercase.
    ogmingod = models.IntegerField(db_column='OGMinGod',
     ↪ blank=True, null=True)  # Field name made lowercase.
    ofmingod = models.IntegerField(db_column='OFMinGod',
     ↪ blank=True, null=True)  # Field name made lowercase.
    oemingod = models.IntegerField(db_column='OEMinGod',
     ↪ blank=True, null=True)  # Field name made lowercase.
    topics = models.TextField(db_column='Topics', blank=True,
```

```
                                    null=True)  # Field name made
                                    ↪ lowercase. This is a list of
                                    ↪ tags that represent the
                                    ↪ course.

    def get_topics_dict(self):
        tags = {}
        for topic in self.topics:
            vals = topic.split(':')
            tags.update({vals[0]: float(vals[1])})
        return tags

    def get_absolute_url(self):
        return reverse('books:course_detail', args=[self.kurs])

    def __str__(self):
        return self.kurs

    class Meta:
        db_table = 'Courses'
```

U fajlu **urls.py** se nalaze sva mapiranja između Templejta i Pogleda za datu aplikaciju. To znači da mora postojati uređen par (pogled, url) kako bi Templejt znao na čiji HTTP zahtev odgovara. Upravo u fajlu **views.py** se nalaze svi pogledi za datu aplikaciju, pa to u slučaju aplikacije books znači sledeće:

- Pogled za prikaz svih knjiga -> book list
- Pogled za prikaz pojedinačnih knjiga -> book detail
- Pogled za prikaz svih kurseva -> course list
- Pogled za prikaz pojedinačnih kurseva -> course detail

Pogledi koji pokazuju liste koriste tzv. *Paginatore*, koji omogućuju stranični prikaz podataka, pristup mnogo efikasniji i po korisnika i po pretraživač. Pogledi su najobičnije Python funkcije koje obrađuju HTTP zahteve i to dvojako u zavisnosti od toga da li je zahtev *GET ili POST*.

Na ovaj način su obrađeni svi scenariji za ovu aplikaciju koji podrazumevaju prikaz podataka. Sličan rezon se koristi za bilo koju drugu aplikaciju u smislu strukture aplikacije - funkcionalnosti se naravno menjaju.

U aplikaciji accounts se pravi podinfrastruktura za sistem registracije i prijave, kao i sve funkcionalnosti koje su planirane da korisnici imaju, a to su:

- Promena ličnih podataka
- Dodavanje knjiga u ličnu kolekciju pročitanih
- Dodavanje i ocenjivanje kurseva dostupnih na osnovu smera i godine studija korisnika
- **Dobijanje preporučenih knjiga**

Kako je izvorni kod preobiman za projektnu dokumentaciju, ohrabruju se zainteresovani da kontaktiraju autore za dobijanje koda, koji će relativno brzo od trenutka pisanja dokumentacije biti objavljen na platformi GitHub. Za sve ostale koji su zainteresovani da nauče detaljnije o samoj strukturi django projekata, dostupna je visoko kvalitetna online dokumentacija.

2.3 Preporučivanje stručne literature - Machine learning

Glavna funkcionalnost ovog sistema i ovakve infrastrukture je koriscenje biblioteka **pandas, numpy, scikit-learn** koje imaju fundamentalnu upotrebu u oblasti *Veštačke inteligencije (eng. AI)* koja se zove *Mašinsko učenje (eng. Machine Learning)*. Algoritmi koji su razvijeni u svrhu preporučivanja sadržaja na osnovu ličnih preferenci i preferenci međusobno sličnih ljudi koriste se već dugi niz godina u ogromnim sistemima kao što su društvene mreže, mreže za deljenje sadržaja i slično.

Projekat *bookMentor* upravo koristi ove biblioteke i takve algoritme Mašinskog učenja kako bi na što kvalitetniji način preporučio knjige korisnicima koji tu uslugu zatraže. Algoritmi koji su korišćeni u projektu iziskuju specifičnu dostavu podataka, odnosno specijalnu strukturu podataka koja u sebi sadrži sve koeficijente za sve oblasti koje su obuhvaćene kako knjigama tako i kursevima, i to za svakog korisnika. Da bi bilo jednostavnije za razumeti, u narednoj tabeli je prikazano zaglavlje takve strukture podataka, reprezentovane u infrastrukturi sistema kao statički CSV fajl, kojim manipuliše dataFrame struktura biblioteke pandas:

Ovi podaci predstavljeni u specijalnom statičkom CSV fajlu se dohvataju preko python biblioteke pandas u strukturu dataFrame, pomoću koje se manipuliše podacima na mnogo jednostavniji način.

Tabela 2.3: Primer zaglavlja strukture ponderisanih koeficijenata oblasti interesovanja po korisnicima

profile.id	topic1	topic2	...	topicN
profileX	0.47	1.2	...	1.14
profileY	0.97	0.14	...	0
profileZ	0	1.52	...	1

Algoritmi preporuke koriste ove podatke i u 2 etape pronalaze kolekciju knjiga za korisnika. Te etape predstavljaju 2 zasebna pristupa preporučivanju:

- Preporuka na osnovu sličnosti korisnika
- Preporuka na osnovu omiljenih ocenjenih kurseva

2.4 Frontend web aplikacije

Kako je projekat u svojoj inicijalnoj fazi, nije preterano značaja posvećeno samom izgledu web stranica. Korišćen je Bootstrap frontend CSS framework, u cilju ulepšavanja stranica. Ipak, autori smatraju da je neophodno uvesti dodatan JavaScript kod uz *AJAX (Asynchronous JavaScript And XML)*, koji bi višestruko poboljšali kvalitet korisničkog iskustva.

JavaScript i AJAX pristup čine ogromnu razliku kod svih softverskih proizvoda realizovanih kao web aplikacije. Projekat u vidu django web aplikacije *bookMentor* bi iskoristio ove softverske alate u cilju poboljšanja korisničkog interfejsa, koji je u trenutnoj fazi poprilično jednostavan iako je potpuno operativan.

Glava 3

Buduća unapređenja projekta

3.1 Ideje

Kao što je naznačeno u Uvodnom poglavlju dokumentacije, postoji više različitih ideja na kojima će se raditi na projektu. S obizorm na to da je inicijalna ideja da projekat ne ostane samo projekat, već komercijalni besplatni proizvod koji bi zapravo služio studentima Elektrotehničkog fakulteta u Beogradu, potrebno je da se obrati pažnja i posveti vreme određenim stvarima.

Pre nego projekat može da bude pušten u *Production phase*, neophodno je izvršiti pronalaženje adekvatnog servera (adekvatnog u smislu procesorksih mogućnosti i količine memorije). Određene algoritme treba još unaprediti u smislu vremenske efikasnosti, iako je većina rađena da već bude dovoljno optimizovana.

Jedna od najvažnijih stavki u životnom ciklusu svih projekata je rešavanje postojećih problema kako ne bi došlo do nagomilavanja u budućnosti. Ideja je da se kod što više prilagodi prema konvenciji **Čistog koda**, radi lakšeg dodavanja novih funkcionalnosti. Poželjno je izvršiti dodatnu normalizaciju baze podataka sa ciljem dodavanja novih tabela u budućnosti. Dodatni alati ažuriranja korisnika su takođe neophodna nova stavka.

Prethodno opisana ažuriranja koda direktno omogućuju dodavanje novih funkcionalnosti kao što su forum/blog sistem na kojem bi korisnici mogli da dele iskustva čitanja svojih knjiga.

Razmatrano je uvođenje sistema zapraćivanja, čime bi se projekat podigao na nivo akademske društvene mreže Elektotehničkog fakulteta u Beogradu. Naravno, autori su svesni pompeznosti ovakvog zahteva u krakom vremenskom periodu, te ova ideja ostaje jedna od onih koje se mogu realizovati u nekom trenutku u budućnosti.

Glava 4

Uputstvo za korišćenje web aplikacije bookMentor

4.1 Neregistrovani korisnici

Neregistrovanim korisnicima je dozvoljen pristup svim knjigama i kursevima na pregled. Ipak, radi dobijanja potpune funkcionalnosti sistema, preporučuje im se registrovanje na sistem, koje je poprilično jednostavno i iziskuje unošenje korisničkog imena, imena, prezimena, mejl adrese i šifre. Neophodno je da u sistemu nije zaveden korisnik sa istim korisničkim imenom.

4.2 Registrovani korisnici

Nakon registracije, korisnici treba da urede svoje dodatne privatne podatke, koji su Usmerenje na Elektrotehničkom fakultetu u Beogradu, i godina studija. Na osnovu ova 2 podatka se dobija lista svih kurseva koje je korisnik mogao do tada da sluša. Ukoliko korisnik ne promeni defaultne podatke, podrazumeva se da je prva godina studijskog programa Elektrotehnika i računarstvo.

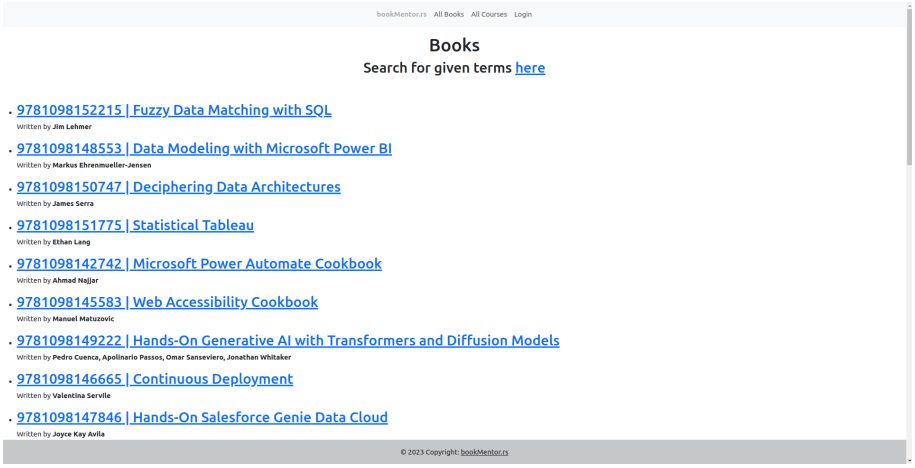
Pored uređivanja podataka, registrovanom korisniku je omogućeno da dodaje knjige u kolekciju knjiga koje je do tada pročitao, da ocenjuje kurseve koje je voleo da sluša na skali od 1 do 5, kao i da dobije preporucene knjige.

Sistem je jednostavan za korišćenje - knjige koje su već pročitane imaju jasnu naznaku da jesu, dok kursevi koji su ocenjeni bivaju fizički odvojeni od ostalih kurseva koje korisnik može da oceni, s ciljem da ne dođe do situacije ocenjivanja već ocenjenog kursa.

4.3 Rad u sistemu kroz slike (early phase)

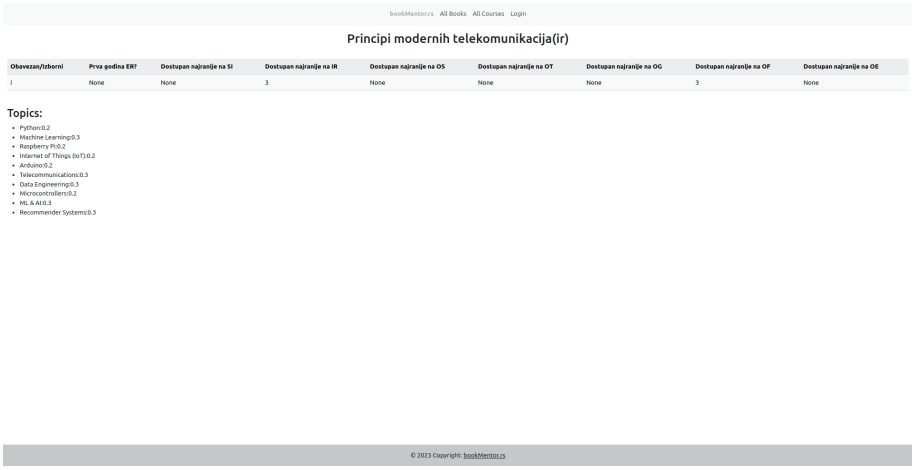
U prilogu su slike gde se vide odredjeni segmenti korišćenja web aplikacije *bookMentor*. U trenutku pravljenja snimaka ekrana, projekat je bio u najranijoj fazi.

Autori ohrabruju sve zainteresovane da probaju early phase verziju projekta web aplikacije *bookMentor* i time dodatno unaprede bazu podataka. S povecanjem baze, rezultati preporuke postaju sve kvalitetniji a time i korisnicko iskustvo.

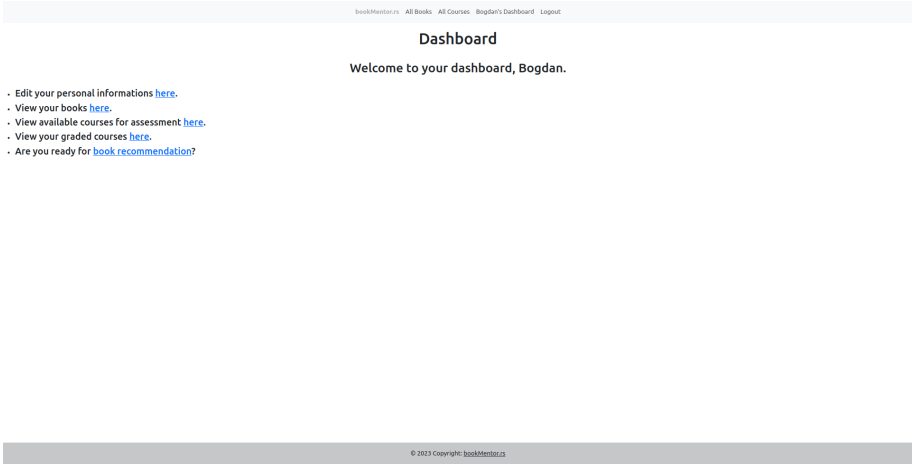


Slika 4.1: *Početna stranica - Listing svih knjiga*

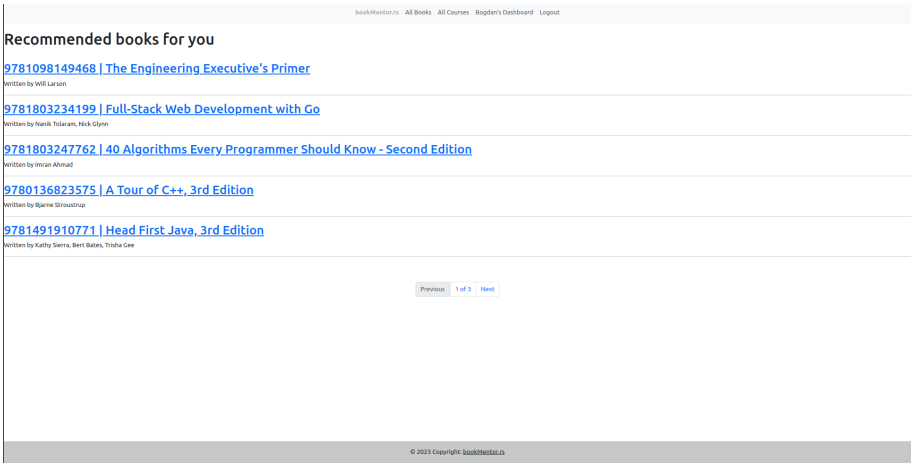
Slika 4.2: *Detaljan prikaz jedne od knjiga*



Slika 4.3: *Detaljan prikaz jednog od kurseva*



Slika 4.4: *Dashboard stranica ulogovanog korisnika*



Slika 4.5: *Preporucene knjige za ulogovanog korisnika*

Glava 5

Zaključak

bookMentor je projekat studenti - studentima, koji za cilj ima da svi jednako kvalitetno napredujemo na svom stručnom putu. Kombinuje nekoliko različitih, a opet povezanih paradigmi i tehnologija Računarske nauke, s ciljem da bude kvalitetan proizvod. Na osnovu ličnih afiniteta ali i afiniteta sličnih korisnika, sistem preporučuje knjige svojim korisnicima. Predstavlja hibridni oblik studentske biblioteke širokog spektra znanja.

Projektna dokumentacija je za cilj imala da na najjednostavniji način prikaže isrcpan proces integracije različitih segmenata s ciljem dobijanja kvalitetnog završnog proizvoda.

Projekat je tek u početnoj fazi svog razvijanja i sklon je minornim ali i onim većim promenama. Ceo programski kod je dostupan na zahtev, čime se makar malo doprinosi ideji slobodnog softvera, što ovaj projekat i jeste.

Autori su dostupni za sve predloge i kritike, a s ciljem dobijanja još kvalitetnijeg proizvoda.

Literatura

[1] *PostgreSQL dokumentacija za verziju 15*, elektronski dokument, 2023; dostupno na: https://www.postgresql.org/docs/15/index.html;

[2] *django dokumentacija za verziju 4*, elektronski dokument, 2023; dostupno na: https://docs.djangoproject.com/en/4.2/;

[3] *Python dokumentacija*, elektronski dokument, 2023; dostupno na: https://docs.python.org/3/;

[4] *Pandas dokumentacija*, elektronski dokument, 2023; dostupno na: https://pandas.pydata.org/docs/;

[5] A. Mele, *Django 3 By Example: Build powerful and reliable Python web applications from scratch 3rd Edition*, Packt Publishing, 2020;

[6] K. Falk, *Practical Recommender Systems*, Manning, 2019;