

Akademija tehničko-vaspitačkih strukovnih studija  
Niš

Predmet: Veb programiranje

**Uputstvo za izradu projekta  
izrade veb aplikacije u python programskom jeziku**

Niš, oktobar 2021.

# Veb programiranje

## Uputstvo za izradu projekta

### izrade veb aplikacije u python programskom jeziku

NAPOMENA: kod pisanja programa u programskom jeziku python je bitna indentacija redova, kopiranjem koda iz ovog dokumenta može doći do grešaka

### Cilj izrade projekta

Izrada ovog projekta ima za cilj da student nauči koncepte, veštine i upotrebu alata za kreiranje osnovnih funkcionalnosti veb aplikacije korišćenjem sledećih veb tehnologija:

- Python - programski jezik za izradu backend-a
- mysql - baziran na sql-u, služi za kreiranje i upravljanje bazom podataka
- phpmyadmin - alat za upravljanje bazom podataka
- Flask - radni okvir koji služi za lakše kreiranje backend funkcionalnosti
- Jinja - Flask biblioteka za kreiranje šablona (html stranica)
- Bootstrap - najpopularniji radni okvir (eng. *framework*) za kreiranje html stranica prilagodljivog izgleda ekrana (eng. *resposnive design*)
- FontAwesome - biblioteka ikonica

### Programski jezik python

Programski jezik Python je objektno orijentisan, interpretativni programski jezik opšte namene. Autor programskog jezika Python je holandski programer Gvido van Rosum (Guido van Rossum). Jezik je nastao ranih devedesetih godina u slobodno vreme, kao jednonedeljni projekat razvoja modernog jezika koji bi bio privlačan Unix/C programerima. Osnova je bio postojeći jezik ABC, zamišljen kao jezik za učenje programiranja i zamena za jezike kao što su BASIC i Pascal. Jezik je razvijan u nacionalnom institutu CWI (Centrum Wiskunde & Informatica), gde je Gvido van Rosum bio zaposlen.

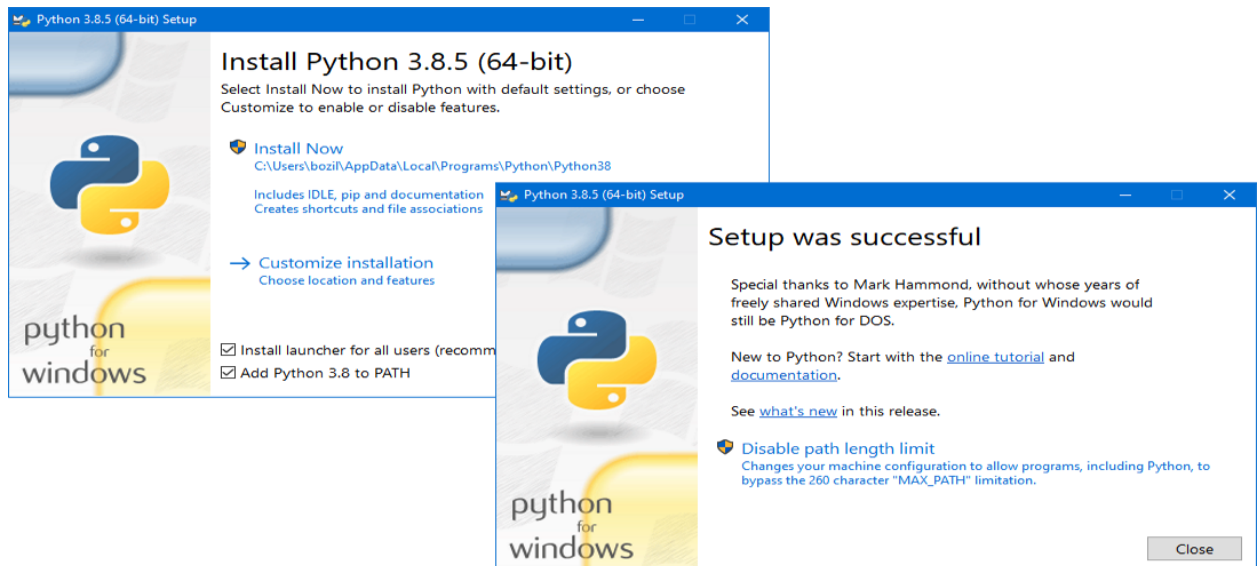
Neke od važnijih verzija programskog jezika Python i godine njihovog objavljivanja:

- Python 1            1994.
- Python 2            2000.
- Python 3            2008.
- Python 3.4          2014.
- Python 3.5          2015.
- Python 3.6          2016.
- Python 3.7          2018.
- Python 3.8          2019.
- Python 3.9          2020.
- Python 3.10        2021. (aktuelna verzija u trenutku pisanja ovog dokumenta)

Programski jezik Python je relativno jednostavan. Međutim, oslanja se na veoma obimne i kvalitetne programske biblioteke. Standardna programska biblioteka Python programskog

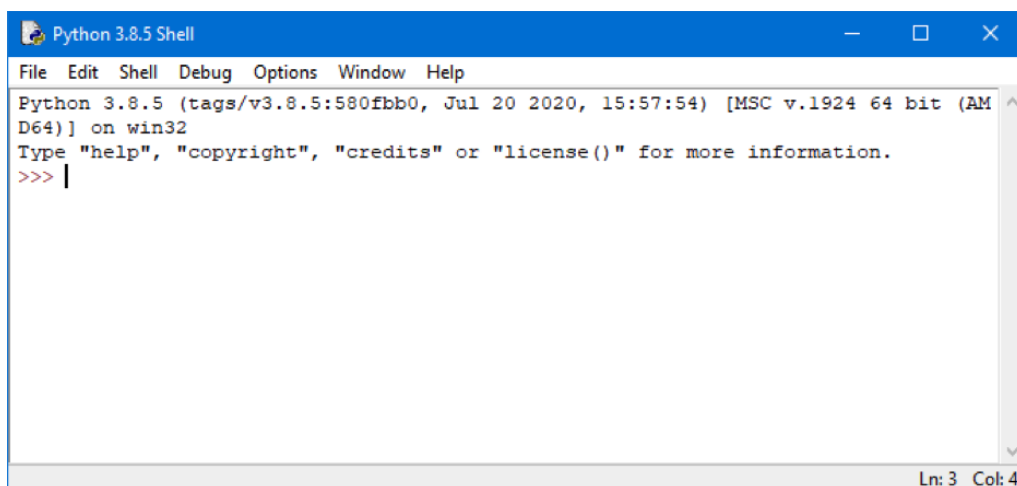
jezika (Python Standard Library) obuhvata tipove i strukture podataka, ugrađene funkcije i obradu izuzetaka, kao i veliki broj međusobno zavisnih modula, koji proširuju same mogućnosti programskoj jezika nakon što se uključe u program programskom naredbom „import”.

U zavisnosti od vrste računara, i operativnog sistema na adresi <https://www.python.org/downloads/>, preuzima se odgovarajuća verzija instalacionog programa i pokreće se instalacija.



Početak i završetak instalacije

Uz samu instalaciju programskog jezika, uključeno je i razvojno okruženje IDLE koje se koristi kao osnovni alat za razvoj programa u programskom jeziku Python.



Razvojno okruženje IDLE

## Osnovni elementi programskog jezika python

Program u Python programskom jeziku je niz naredbi, čiji je delimiter kraj linije teksta, za razliku od drugih programskih jezika gde je često kraj linije tačka zarez (;).

Posebnu ulogu u pisanju programa ima uvlačenje (indentacija), koja takođe služi kao delimiter u složenim višelinijskim naredbama.

U naredbama se razlikuju mala i velika slova (*case sensitive*), tako da se na primer naredbe `print` i `Print` međusobno razlikuju.

Sintaksa programskog jezika Python će se u ovom poglavlju uvoditi postepeno, kroz kratak opis osnovnih verzija naredbi i tipične primere njihove upotrebe. Prikazaće se:

- komentari,
- promenljive i izrazi,
- osnovne strukture podataka,
- upravljanje tokom programa (grananje i ponavljanje),
- interaktivni ulaz-izlaz.

## Komentari

Komentari predstavljaju proizvoljan tekst koji prevodilac zanemaruje, ali predstavljaju važan deo programskog koda. Sadrže informacije namenjene osobama koje čitaju program, tako da su istovremeno i neophodna minimalna dokumentacija softvera. U programskom jeziku Python postoje dve vrste komentara: komentari u jednoj liniji i višelinijski komentari.

Komentari dužine jedne linije počinju znakom „#”, a završavaju se oznakom kraja linije programa.

```
# Komentar u jednoj liniji
```

Višelinijski komentari omogućavaju unos proizvoljnog teksta, između dve programske linije, i počinju sa tri uzastopna znaka navoda.

```
"""
Primer
komentara u
više linija
"""
```

## Promenljive i izrazi

Promenljive u programskom jeziku Python se ne deklariraju i nemaju unapred definisan tip, već je tip svake promenljive određen vrednošću koja joj se dodeli.

```
x = 2020
```

```
y = 'Mirko'
```

Tip vrednosti promenljive se može se ispitati pomoću ugrađenje funkcije `type()`.

```
type(x)
```

Vrednost promenljive se može izračunati pomoću nekog izraza.

```
y = 'Mirko'
pozdrav = 'Zdravo, ' + y
# promenljiva pozdrav ima vrednost 'Zdravo, Mirko'
x = 10
proizvod = 2 * x
# promenljiva proizvod ima vrednost 20
```

## Osnovni tipovi i strukture podataka

Tip podatka čini skup vrednosti i skup operatora koji se mogu primeniti na te vrednosti. Osnovni numerički tipovi su celi i decimalni brojevi, kao i logičke vrednosti.

```
int    5
float  5.1
bool   1
```

Tip **bool** predstavlja podtip tipa int, tako da logičke vrednosti istinitosti imaju celobrojnu iterpretaciju: True == 1 i False == 0.

Osnovne strukture podataka u programskom jeziku Python su stringovi, liste, n-torke i rečnici.

Stringovi su nizovi znakova (karaktera), čija se pozicija u nizu računa od nule.

```
ime = "Janko"
```

Brojanje elemenata stringa vrši se tako da prvi element ima poziciju ili indeks 0, a svaki naredni element za jedan veći. Elementi se mogu brojati i unazad, počevši od poslednjeg elementa stringa koji ima indeks -1.

```
ime = "Janko"
prvi_element = ime[0]
poslednji_element = ime[-1]
```

Liste su sekvence ili nizovi podataka, koji mogu biti različitog tipa i kojima se može pristupiti pomoću indeksa, tj. podatka o njihovom redosledu u nizu.

```
lista = [1,2,3]
lista_2 = ['A', 'B', 'C']
```

Kao i kod stringova, i kod listi se svakom elementu može pristupiti na osnovu indeksa.

```
lista = [1,2,3]
drugi_element = lista[1]
```

Ugnježdene liste predstavljaju liste čiji su elementi takođe strukture tipa liste pa se u memoriji računara pomoću njih mogu predstaviti složeniji odnosi između podataka, kao na primer hijerarhije. Definisane liste lista i lista\_2 mogu se uključiti u strukturu ugnježdene liste.

```
lista = [1, 2, 3]
lista_2 = ['a', 'b', 'c']
ugnjezdena_lista = [lista, lista_2]
#ugnjezdena_lista ima vrednost [[1, 2, 3], ['a', 'b', 'c']]
```

Struktura n-torke takođe je namenjena za predstavljanje nizova i sekvenci objekata različitog tipa, ali se za razliku od liste, ne može se menjati. Zbog toga se za predstavljanje n-torki koriste druge oznake. Na primer, prethodna hijerarhijska struktura može se predstaviti i pomoću nepromenljive strukture n-torke.

```
ntorka = ((1,2,3), ('a', 'b', 'c'))
```

Rečnik je struktura kod koje se elementima, koji takođe mogu biti bilo kog tipa, ne pristupa prema numeričkom indeksu, već pomoću nenumeričke informacije koja se naziva ključ.

```
student = {  
    "ime": "Mirko",  
    "prezime": "Mirkovic",  
    "broj_indeksa": "REr 5/20"  
}
```

## Upravljanje tokom programa (grananje i ponavljanje)

Osnovne naredbe za upravljanje tokom izvršavanja programa u programskom jeziku Python su naredbe grananja (selekcije) i naredbe ponavljanja (iteracije).

Naredba selekcije služi za izbor delova programa koji se izvršavaju samo pod određenim uslovima.

```
if dan_je_suncan:  
    print("Kisobran nije potreban.")
```

Naredbe koje se uslovno izvršavaju uvučene su za isti broj mesta u odnosu na početak naredbe `if`. Standardno se koristi uvlačenje (indentancija) za 4 mesta.

Najjednostavnija naredba ponavljanja je naredba `for`, koja izvršava skup naredbi zadani broj puta, po jednom za svaku vrednost iz zadatog skupa vrednosti.

```
skup_vrednosti = [1,2,3]  
for element in skup_vrednosti:  
    print(element)
```

Skup vrednosti se može generisati i pomoću neke funkcije kao što je `range(1, 101)`, koja proizvodi niz celih brojeva u rasponu od 1 do 100.

```
for element in range (1, 101):  
    print(element)
```

## Interaktivni ulaz-izlaz

Osnovne naredbe za interaktivni ulaz i izlaz podataka su funkcija `input()`, koja služi za čitanje podataka unesenih putem tastature i funkcija `print()`, koja vrši ispis podataka na ekran računara.

Program pomoću funkcije `input()` postavlja pitanje korisniku, koje se prikazuje na ekranu računara, a kursor se postavlja na sledeće slobodno polje radi unosa i čitanja odgovora. Kada korisnik putem tastature unese odgovor i potvrdi tasterom Enter, uneti tekst se vraća kao rezultat izvršavanja funkcije i može se zapamtiti u nekoj promenljivoj.

```
ime = input("Unesite Vase ime: ")
```

## Lokalni server (Localhost)

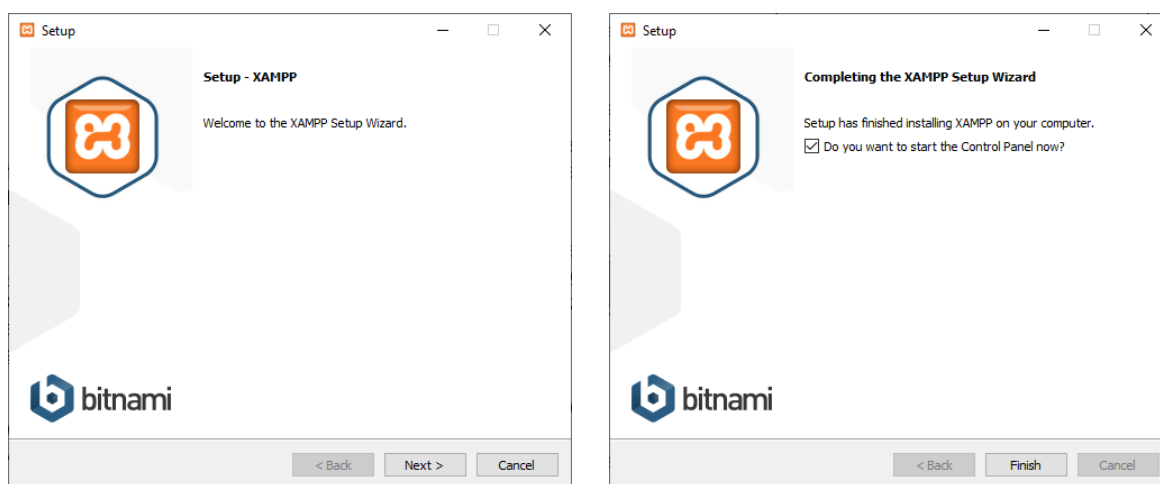
Lokalni server engl. **localhost** u najjednostavnijoj definiciji je web server koji se nalazi na lokalnom sistemu kao što je vaš računar, što znači da nije dostupan na internetu. Lokalni server daje programeru mogućnost da testira svoj proizvod u kontrolisanom okruženju bez plaćanja zakupa hosting prostora na serveru hosting kompanije. Pošto se datoteke čuvaju lokalno na vašem računaru, pregledač ima sve što je potrebno za prikaz web stranice koja je

vidljiva samo vama. Lokalni server je sigurniji i pruža idealno okruženje za testiranje, daleko od znatiželjnih očiju ili hakera. Nije vam potreban internet da biste mogli da koristite svoj lokalni server bilo gde, u bilo koje vreme, samo vam treba računar.

Da biste na računaru mogli da pokrenete i testirate veb aplikaciju, potreban je Apache (web server), MySQL (sistem za upravljanje bazama otvorenog koda), php i phpmyadmin, a instalacijom XAMPP-a se dobija sve navedeno. Na web adresi nalazi se opcija za preuzimanje XAMPP izaberite verziju koja odgovara vašem operativnom sistemu:

<https://www.apachefriends.org/download.html>

Tokom postupka instalacije može se pojaviti upozorenje poput „*ako ste sigurni da želite da instalirate softver i instalacioni program, kliknite da biste nastavili instalaciju*“. Čarobnjak za postavljanje XAMPP vodiće vas kroz instalaciju.



Početak i završetak instalacije

## Flask radni okvir

Flask je mikro radni okvir za izradu veb aplikacija (*micro web framework*), baziran na programskom jeziku Python, koji daje skup opštih osnovnih tehnologija na kojima može da se gradi veb aplikacija.

Radni okvir (*framework*) je najjednostavnije rečeno, biblioteka ili zbirka biblioteka koja ima za cilj rešavanje dela generičkog problema umesto potpunog, specifičnog problema. Prilikom izrade veb aplikacija postoje neki problemi koje uvek treba rešavati, kao što je umetanje dinamičkih podataka u HTML, ili interakcija sa krajnjim korisnikom.

To što je nazvan malim, odnosno mikro radnim okvirom, ne znači da može da uradi manje od ostalih veb radnih okvira. Dizajniran je kao proširivi radni okvir i pruža solidno jezgro sa osnovnim uslugama, dok proširenja (ekstenzije) pružaju sve ostalo. Programer ima moć da odabere ekstenzije koje najbolje odgovaraju njegovom projektu, ili da čak napiše svoje ekstenzije ako to želi. Ovo je u suprotnosti sa većinom radnih okvira, gde su sve odluke donete unapred, i teško je ili je nemoguće promeniti ih.

U Flasku ne postoji matična podrška za pristup bazama podataka, validaciju veb obrazaca (formi), autentifikaciju korisnika ili druge zadatke na visokom nivou. Ove i mnoge druge ključne

usluge koje su potrebne većini veb aplikacija, dostupne su kroz proširenja (ekstenzije) koja se integrišu sa osnovnim paketima.

Prva verzija Flask-a pojavila se 16. aprila 2010. godine, a njegov autor je austrijski programer Armin Ronašer (Armin Ronacher).

## Inicijalizacija projekta i instalacija potrebnih biblioteka

Potrebno je kreirati direktorijum koji će sadržati projekat (npr. "evidencija\_studenata"). Zatim otvoriti comand prompt i u pozicionirati se unutar kreiranog direktorijuma:

```
cd C:\Users\putanja\do\direktorijuma
```

Pozicionirati se unutar direktorijuma i instalirati biblioteke Flask i Python drajver za komunikaciju sa MySQL serverom mysql-connector-python

Flask:

```
python -m pip install Flask
```

mysql-connector-python:

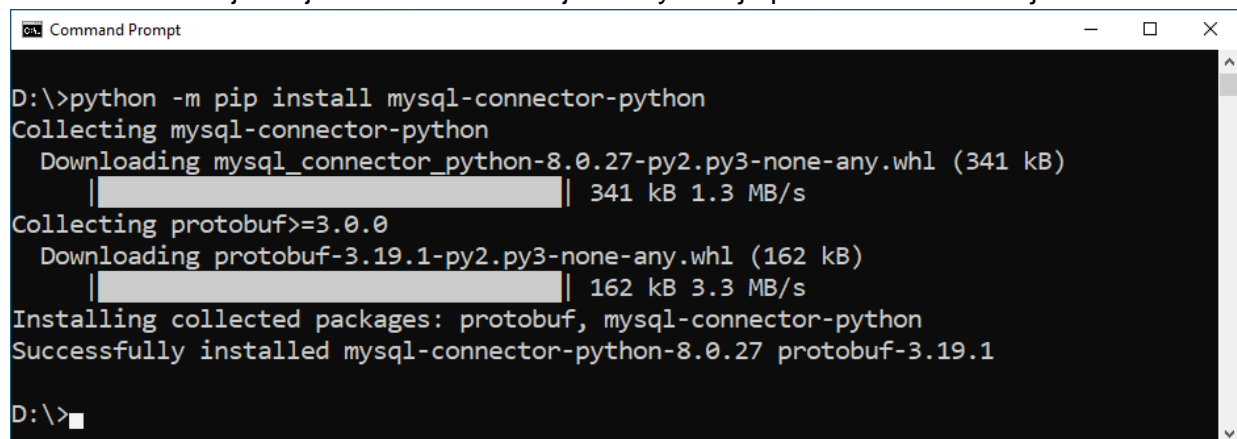
```
python -m pip install mysql-connector-python
```

maria db

```
python -m pip install mariadb
```

XAMPP od verzije 5.5.30 i 5.6.14, ne koristi MySQL vec koristi MariaDB, ipak sintaksa i mysql paket koji uvozimo su kompatibilnii sa ovom bazom.

Rezultat instalacije drajvera za komunikaciju sa MySQL je prikazan na sledećoj slici:



```
Command Prompt
D:\>python -m pip install mysql-connector-python
Collecting mysql-connector-python
  Downloading mysql_connector_python-8.0.27-py2.py3-none-any.whl (341 kB)
    | 341 kB 1.3 MB/s
Collecting protobuf<=3.0.0
  Downloading protobuf-3.19.1-py2.py3-none-any.whl (162 kB)
    | 162 kB 3.3 MB/s
Installing collected packages: protobuf, mysql-connector-python
Successfully installed mysql-connector-python-8.0.27 protobuf-3.19.1
D:\>
```

NAPOMENA: ovim postupkom se flask i mysql-connector-python instaliraju globalno, dok postoji i opcija zasebnog instaliranja po projektu.

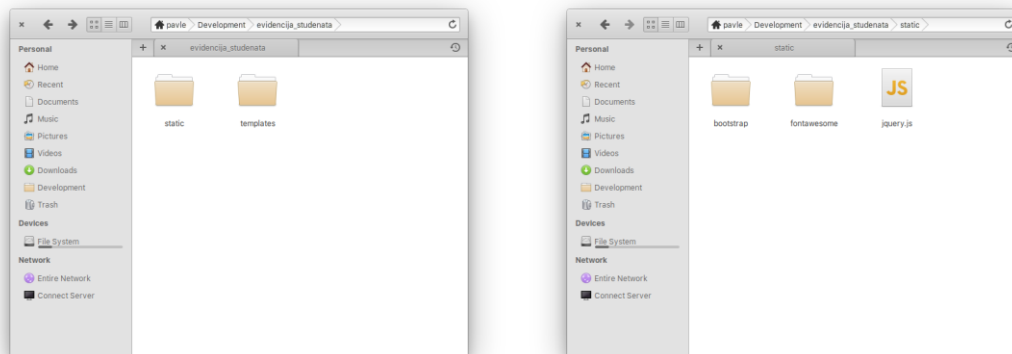
Unutar projekta treba kreirati 2 foldera: "static" i "templates". Folder "static" će sadržati sve statičke fajlove, kao što su Bootstrap biblioteka, FontAwesome biblioteka, jQuery skripta, slike koje vab aplikacija koristi, css stilovi i slično. Folder "templates" će sadržati sve šablone (html fajlove) napisane korišćenjem Jinja biblioteke.



Sa sajta <https://getbootstrap.com/> preuzeti poslednju verziju Bootstrap biblioteke. Otpakovati kompresovan fajl i dobijeni folder smestiti u folder “static” unutar projekta pod nazivom bootstrap.

Sa sajta <https://jquery.com/download/> preuzeti biblioteku jQuery (od koje zavisi Bootstrap) i fajl jquery.js smestiti u folder “static”, pod nazivom jquery.js.

Sa sajta <https://fontawesome.com/download> preuzeti FontAwesome biblioteku, raspakovati .zip fajl i dobijeni folder smestiti u folder “static” pod nazivom fontawesome.



*Folder projekta i folder static*

## Projektovanje zadatka

Prvi korak u izradi zadatka je projektovanje. Potrebno je definisati zadatak i korake kojima se dolazi do željenog proizvoda.

Zadatak je izrada veb aplikacije u kojoj profesori vode evidenciju o studentima, predmetima i ocenama. Od funkcionalnosti je potrebno da ima login, logout, kreiranje, iščitavanje, izmenu i brisanje studenata, predmeta, ocena i profesora (u daljem tekstu korisnici).

Potrebno je definisati osnovni izgled veb aplikacije i raspored elemenata na njemu, odnosno napraviti wireframe.

Na sledećem linku je wireframe prototip aplikacije

<https://xd.adobe.com/view/a5707705-2bb5-4657-a80c-799cbd047b52-f023/>

## app.py

Unutar projekta, pored foldera “static” i “templates”, kreirati fajl “app.py”. Ovaj fajl će predstavljati polaznu tačku aplikacije. Sadržaće definiciju aplikacije i sve njene funkcionalnosti. U njemu će biti definisane sve rute koje sajt koristi, kao i logika koja se odvija na njima.

```
# uvoz potrebnih biblioteka na vrhu fajla
from flask import Flask, render_template, url_for, request, redirect, session
import mysql.connector
```

```
# deklaracija Flask aplikacije ispod "import-a"
app = Flask(__name__)

# Logika aplikacije
# je smeštena
# između deklaracije
# i pokretanja aplikacije

# pokretanje aplikacije na dnu fajla
app.run(debug=True)
```

Kod koji je naveden u zagradama `debug=True`, kao u nastavku:

```
app.run(debug=True)
```

omogućava da se aplikacija izvršava u režimu za ispravljanje grešaka (*debugging mode*), koji između ostalog, automatski ponovo pokreće veb aplikaciju kad god Flask primeti da se kod promenio (obično kod izmene koda i snimanja izmena). Prilikom postavljanja konačne verzije aplikacije na server i puštanja u produkciju, ovaj deo svakako treba isključiti (obrisati).

U command prompt-u se pozicionirajte unutar projekta. Komandom

```
python app.py
```

pokreće se aplikacija na lokalnom serveru, na adresi <http://127.0.0.1:5000>, odnosno <http://localhost:5000>

## Definisanje rute

HTTP podržava 2 metode za prenos podataka, GET i POST. Ruta definisana u aplikaciji može podržavati jednu od njih ili obe. Ruta se definiše na sledeći način:

```
@app.route("/putanja/do/resursa", methods=["GET", "POST"])
def ime_funkcije_na_ovoj_ruti():
    # definicija funkcije
```

Ako je potrebno funkciji proslediti neki parametar (najčešće id), to se može uraditi na sledeći način:

```
@app.route("/resurs/<id>", methods=["GET"])
def ime_funkcije(id):
    # definicija funkcije
```

Dakle, sastoji se od 2 dela, definicije rute i deklaracije i definicije funkcije koja se izvršava na toj ruti.

Aplikacija je potrebno da ima sledeće rute sa odgovarajućim funkcijama:

```
/login
/logout
/korisnici
/korisnik_novi
/korisnik_izmena/<id>
/korisnik_brisanje/<id>
```

```
/studenti
/student/<id>
/student_novi
/student_izmena/<id>
/student_brisanje/<id>
<student_id> kroz ovu rutu racunamo i novi prosek studenta, tako da je dodata jos jedna
promenljivu
/predmeti
/predmet_novi
/predmet_izmena/<id>
/predmet_brisanje/<id>
/ocena_nova/<id>
/ocena_brisanje/<id>
/ocena_izmena/<id>
/ocena_brisanje/<id>/
```

## Kreiranje šablona (template)

Za početak, u folderu "templates" potrebno je kreirati fajl "base.html" koji će služiti kao omotač (wrapper) za šablone (template). U njemu se definiše html dokument, naslov (title) aplikacije i u njemu se uvoze sve potrebne frontend biblioteke, stilovi i skripte.

U <head> elementu se uvoze stilovi (Bootstrap, FontAwesome, vaši stilovi). Na dnu <body> elementa se uvoze skripte (jQuery, Bootstrap, FontAwesome, vaše skripte).

Uz pomoć jinja sintakse unutar elementa <body> definišemo jedan blok koji će služiti kao placeholder za sve šablone.

base.html treba da izgleda ovako:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">

  <link rel="stylesheet" href="/static/bootstrap/css/bootstrap.min.css">
  <link rel="stylesheet" href="/static/fontawesome/css/all.min.css">
  <link rel="stylesheet" href="/static/style.css">

  <title>Evidencija studenata</title>
</head>

<body>

  {% block content %} {% endblock %}

  <script src="/static/jquery.js"></script>
  <script src="/static/bootstrap/js/bootstrap.min.js"></script>
```

```
<script src="/static/fontawesome/js/all.min.js"></script>
</body>
</html>
```

Kreirati fajl "korisnici.html" u folderu "templates". U ovom fajlu biće definisan izgled stranice na kojoj se prikazuje tabela sa korisnicima.

Na vrhu fajla je potrebno naglasiti da on nasleđuje fajl "base.html" kako bi biblioteke uvezene u tom fajlu bile dostupne.

```
{% extends 'base.html' %}
```

U nastavku je potrebno naglasiti da čitav sadržaj ove stranice treba smestiti u blok "content", i to na sledeći način:

```
{% block content %}
<!--sadržaj stranice-->
{% endblock %}
```

Trenutno "korisnici.html" izgleda ovako:

```
{% extends 'base.html' %}
{% block content %}

<!--sadržaj stranice-->
{% endblock %}
```

U "app.py" potrebno je definisati rutu "/korisnici" i u njenoj funkciji korisniku vratiti (iscrtati, renderovati) stranicu "korisnici". Za to nam je potrebna funkcija `render_template()` iz Flask biblioteke.

```
@app.route("/korisnici", methods=[ 'GET' ])
def korisnici():
    return render_template("korisnici.html")
```

U browser-u možemo na adresi <http://127.0.0.1:5000/korisnici> videti novokreirani šablon.

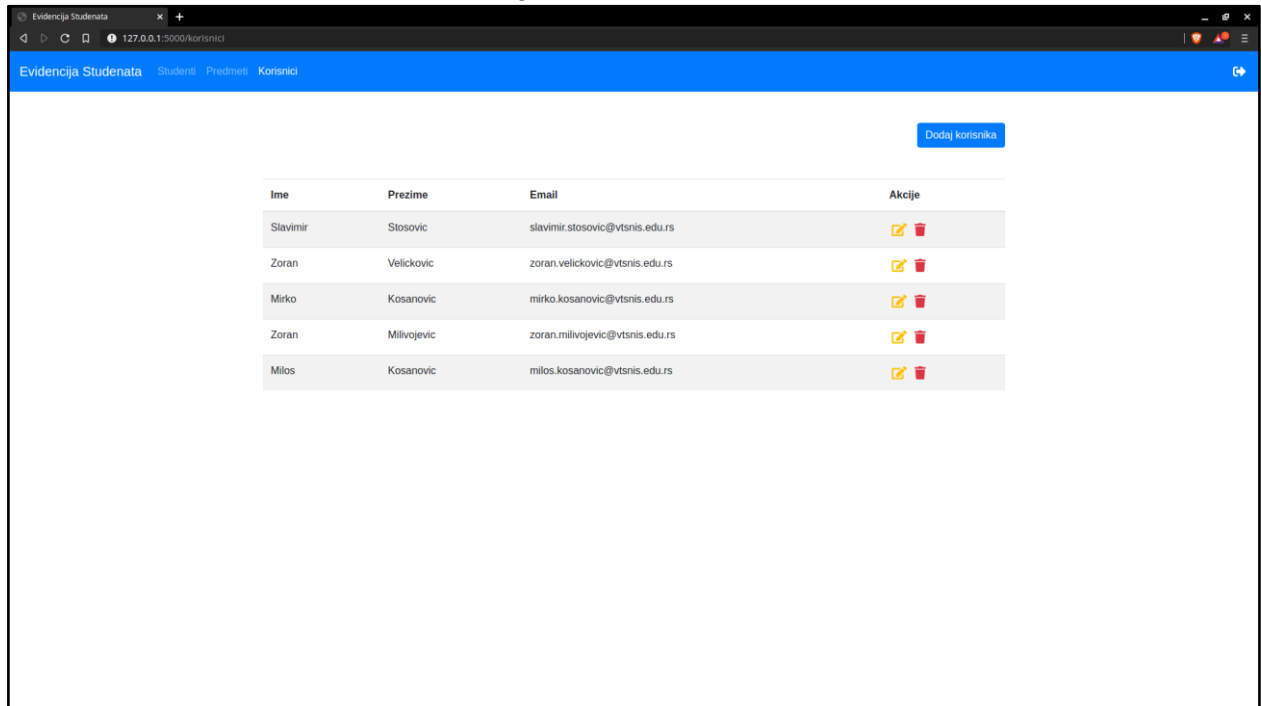
Sada je potrebno napraviti izgled stranice "korisnici". Za to se mogu koristiti Bootstrap komponente. Iz Bootstrap dokumentacije

(<https://getbootstrap.com/docs/4.5/components/alerts/>) se mogu preuzeti gotove komponente i modifikovati za potrebe aplikacije.

Jedna stranica kao što je "korisnici" potrebno je da sadrži sledeće elemente:

- Navigaciju na vrhu sa imenom aplikacije, linkovima do ostalih stranica i linkom za odjavljivanje korisnika.
- Dugme za dodavanje novog korisnika
- Tabelu korisnika sa njihovim informacijama i linkovima za izvršenje akcije nad njima (izmena, brisanje)

Kreirana stranica bi trebalo ovako da izgleda



*korisnici.html*

Primer modifikovane komponente Navbar:

```
<nav class="navbar navbar-expand-lg navbar-dark bg-primary">
  <a class="navbar-brand" href="#">Evidencija studenata</a>
  <button class="navbar-toggler" type="button" data-toggle="collapse"
data-target="#navbarSupportedContent"
    aria-controls="navbarSupportedContent" aria-expanded="false" aria-
label="Toggle navigation">
    <span class="navbar-toggler-icon"></span>
  </button>

  <div class="collapse navbar-collapse" id="navbarSupportedContent">
    <ul class="navbar-nav mr-auto">
      <li class="nav-item">
        <a class="nav-link" href={{ url_for('studenti')}}>Studenti</a>
      </li>
      <li class="nav-item">
        <a class="nav-link" href={{ url_for('predmeti')}}>Predmeti</a>
      </li>
      <li class="nav-item active">
        <a class="nav-link" href={{ url_for('korisnici')}}>Korisnici</a>
      </li>
    </ul>
    <form class="form-inline">
```

```
        <a href={{ url_for('logout') }} class="btn btn-primary">
            <i class="fas fa-sign-out-alt"></i>
        </a>
    </form>
</div>
</nav>
```

```
{{ url_for('studenti') }}
```

Ovo je deo jinja sintakse. Funkcija `url_for()` prihvata više argumenata. Prvi je uvek ime funkcije koju želimo da pozovemo iz “app.py”, ostalo su argumenti koje ta funkcija prihvata i opcioni su.

Tako može da postoji:

```
{{ url_for('korisnik_izmena', id=5) }}
```

Ovo je ekvivalentno unosu adrese u “http://127.0.0.1:5000/korisnik\_izmena/5” pretraživaču.

```
<i class="fas fa-sign-out-alt"></i>
```

Ovo je primenjena ikonica iz biblioteke FontAwesome.

CSS klase koje se koriste u ovom primeru su deo Bootstrap biblioteke.

Ovaj deo koda se može primeniti na svim šablonima na kojima treba da se pojavi navigacioni meni na vrhu. Jedina razlika može biti u tome kom se `<li>` elementu dodeljuje klasa “active” koja označava link trenutno aktivne stranice.

Ostatak šablona:

```
<div class="container">

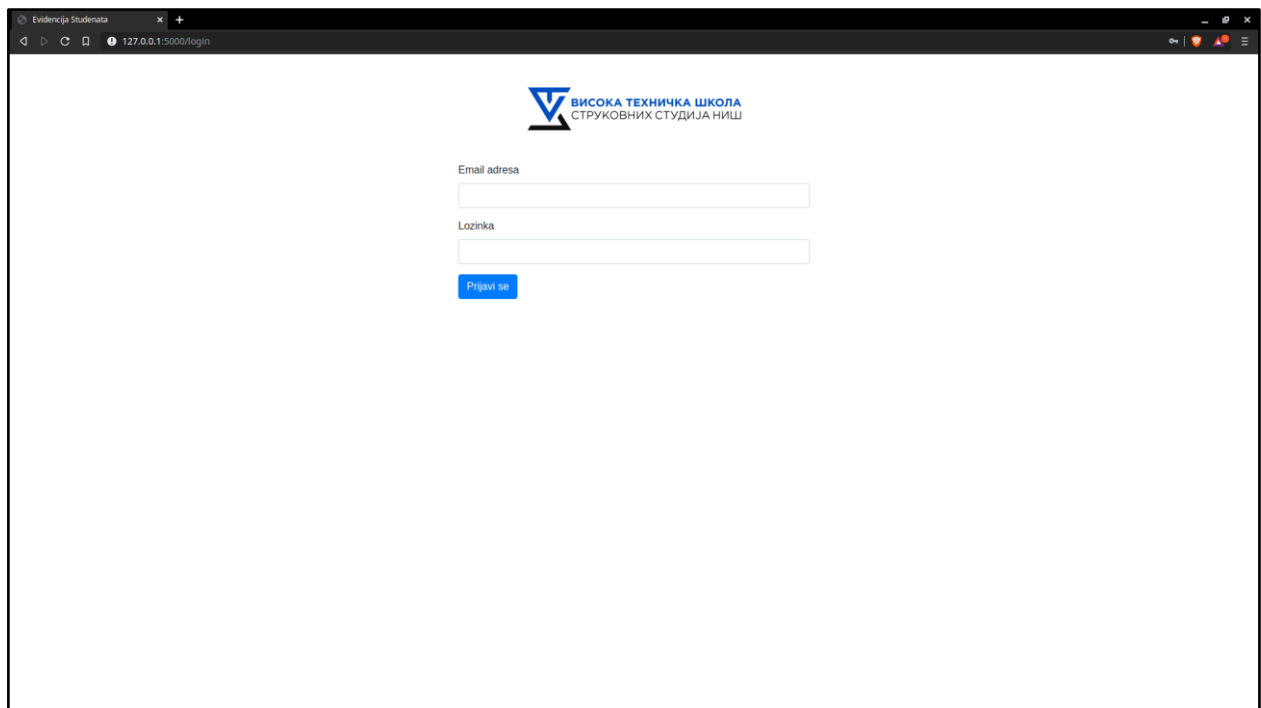
    <div class="row my-5 justify-content-end">
        <a href={{ url_for('korisnik_novi') }} role="button" class="btn
btn-primary">Dodaj korisnika</a>
    </div>

    <div class="row">
        <table class="table">
            <thead>
                <tr>
                    <th scope="col">Ime</th>
                    <th scope="col">Prezime</th>
                    <th scope="col">Email</th>
                    <th scope="col">Akcije</th>
                </tr>
            </thead>
            <tbody>
                <tr>
                    <td>Petar</td>
                    <td>Petrovic</td>
```

```
        <td>petar.petrovic@vtsnis.edu.rs</td>
        <td class="row">
            <a href={{ url_for("korisnik_izmena", id=11) }}
role="button" class="text-warning mx-1">
        <i class="fas fa-edit"></i>
        </a>
            <a href={{ url_for("korisnik_brisanje", id=11) }}
role="button" class="text-danger mx-1">
        <i class="fas fa-trash"></i>
        </a>
        </td>
    </tr>
</tbody>
</table>
</div>
</div>
```

Ovde je definisano dugme koje vodi na stranicu za kreiranje korisnika i tabela sa korisnicima. Trenutno tabela sadrži samo jedan red kao primer. Potrebno je kasnije uz pomoć jinja-e u tabeli ispisati informacije o svim korisnicima iz baze podataka.

Po ovom principu je potrebno napraviti i ostale stranice.



login.html

Evidencija Studenata

Novi korisnik

Ime

Prezime

Email











Lozinka

Sačuvaj

*korisnik\_novi.html*

Evidencija Studenata

Dodaj predmet

Šifra	Naziv	Godina Studija	ESPB	Obavezni / Izborni	Akcije
VP5	Web Programiranje	3	6	Obavezni	 
RM4	Racunarske Mreze	2	6	Obavezni	 
AM5	Arhitektura Mikrokontrolera	3	6	Obavezni	 
TE3	Tehnici Engleski 1	2	2	Izborni	 
F1	Fizika	1	6	Obavezni	 
















*predmeti.html*



The screenshot shows a web browser window with the address bar displaying '127.0.0.1:5000/predmet\_novi'. The page has a blue header with the navigation menu: 'Evidencija Studenata', 'Studenti', 'Predmeti', and 'Korisnici'. The main content area is titled 'Novi predmet' and contains a form with the following fields: 'Šifra' (text input), 'Naziv' (text input), 'Godina studija' (text input), 'ESPB' (text input), 'Obavezni / Izborni' (dropdown menu with 'Status predmeta' selected), and a 'Sačuvaj' (Save) button.

*predmet\_novi.html*

The screenshot shows a web browser window with the address bar displaying '127.0.0.1:5000/studenti'. The page has a blue header with the navigation menu: 'Evidencija Studenata', 'Studenti', 'Predmeti', and 'Korisnici'. The main content area features a 'Dodaj studenta' (Add student) button in the top right corner and a table listing students.



Broj Indeksa	Ime	Prezime	Godina Studija	ESPB	Prosek Ocena	Akcije
REr 1/15	Momcilo	Bajagic	3	115	8.45	  
REr 2/15	Djordje	Balasevic	3	100	7.6	  
REr 3/15	Vladislava	Cosic	3	96	6.41	  
SEr 40/15	Marina	Perazic	3	120	9.69	  
REr 41/15	Djordja	Zejak	3	105	7.98	  

*studenti.html*

The screenshot shows a web browser window with the address bar displaying '127.0.0.1:5000/student\_novi'. The page has a blue header with the navigation menu: 'Evidencija Studenata', 'Studenti', 'Predmeti', and 'Korisnici'. The main content area is titled 'Novi student' and contains a vertical form with the following fields: 'Broj indeksa', 'Ime', 'Ime roditelja', 'Prezime', 'Email', 'Broj telefona', 'Godina studija', 'Datum rođenja' (with a date picker showing 'mm/dd/yyyy'), and 'JMBG'. A blue 'Sačuvaj' button is located at the bottom of the form.



*student\_novi.html*

The screenshot shows the 'student.html' page. It features a blue header with the same navigation menu. The main content area is divided into two sections. On the left, a table displays the student's details:

Ime	Momcilo
Ime roditelja	Dragan
Prezime	Bajagic
Broj indeksa	REr 1/15
Godina studija	3
Broj telefona	067 / 123 4567
Email	janko.jankovic@mail.com
Datum rođenja	10.12.1998.
JMBG	1234567891234
Ukupno ESPB	110
Prosek ocena	8.4
Akcije	 

On the right, there is a form for adding a grade, with fields for 'Predmet' (a dropdown menu showing 'Odaberi predmet'), 'Ocena', and 'Datum' (a date picker showing 'mm/dd/yyyy'). A blue 'Dodaj ocenu' button is at the bottom of this form.

At the bottom of the page, there is a table showing a list of courses:

Šifra	Naziv	Godina studija	Obavezni / Izborni	ESPB	Ocena	Akcije
VP5	Web Programiranje	3	Obavezni	6	8	 

*student.html*

Stranice za kreiranje i izmenu nekog resursa su izgledom identične. Razlika je u tome što će stranice za izmenu biti popunjene već postojećim podacima o korisniku, predmetu ili studentu. Tako stranice "korisnik\_novi" i "korisnik\_izmena" mogu trenutno da imaju isti kod.

## Kreiranje baze podataka

Da bi aplikacija imala sve potrebne informacije potrebno je kreirati 4 tabele u sql bazi podataka. U phpmyadmin-u, na adresi <http://localhost/phpmyadmin> potrebno je kreirati bazu sa imenom "evidencija\_studenata". U bazi kreirati 4 tabele: "korisnici", "ocene", "predmeti", "studenti".

Tabela korisnici:

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
<input type="checkbox"/> 1	id	int(11)			No	None		AUTO_INCREMENT	Change  Drop  More
<input type="checkbox"/> 2	ime	varchar(100)	utf8mb4_general_ci		No	None			Change  Drop  More
<input type="checkbox"/> 3	prezime	varchar(100)	utf8mb4_general_ci		No	None			Change  Drop  More
<input type="checkbox"/> 4	email	varchar(100)	utf8mb4_general_ci		No	None			Change  Drop  More
<input type="checkbox"/> 5	lozinka	varchar(100)	utf8mb4_general_ci		No	None			Change  Drop  More
<input type="checkbox"/> 6	rola	varchar(50)	utf8mb4_general_ci		No	None			Change  Drop  More

Tabela ocene:

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
<input type="checkbox"/> 1	id	int(11)			No	None		AUTO_INCREMENT	Change  Drop  Primary  Unique  Index  More
<input type="checkbox"/> 2	student_id	int(11)			No	None			Change  Drop  Primary  Unique  Index  More
<input type="checkbox"/> 3	predmet_id	int(11)			No	None			Change  Drop  Primary  Unique  Index  More
<input type="checkbox"/> 4	ocena	smallint(6)			No	None			Change  Drop  Primary  Unique  Index  More
<input type="checkbox"/> 5	datum	date			No	None			Change  Drop  Primary  Unique  Index  More

Polja student\_id i predmet\_id su strani ključevi iz tabela studenti i predmeti, a veza se ostvaruje u delu Relation view na način koji je prikazan na sledećoj slici:

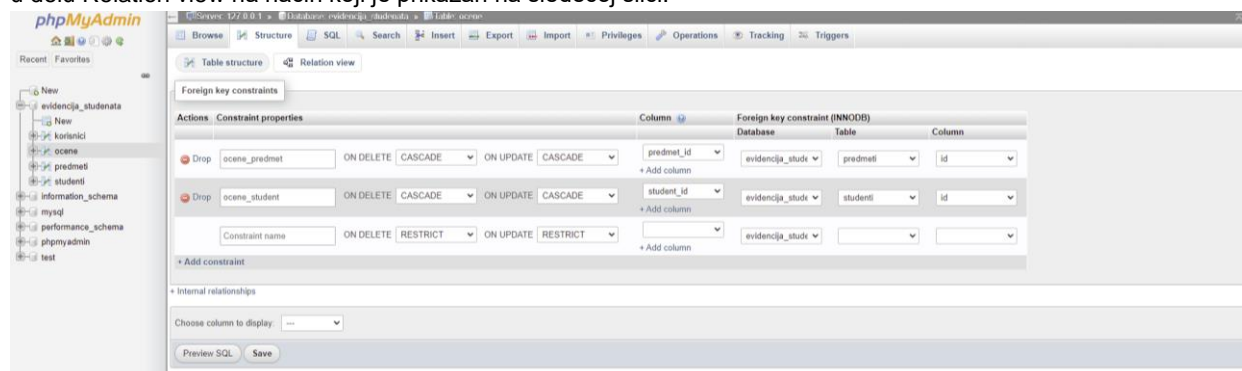


Tabela predmeti:

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
<input type="checkbox"/> 1	id	int(11)			No	None		AUTO_INCREMENT	Change  Drop  Primary  Unique  More
<input type="checkbox"/> 2	sifra	varchar(30)	utf8mb4_general_ci		No	None			Change  Drop  Primary  Unique  More
<input type="checkbox"/> 3	naziv	varchar(50)	utf8mb4_general_ci		No	None			Change  Drop  Primary  Unique  More
<input type="checkbox"/> 4	godina_studija	smallint(6)			No	None			Change  Drop  Primary  Unique  More
<input type="checkbox"/> 5	espb	int(11)			No	None			Change  Drop  Primary  Unique  More
<input type="checkbox"/> 6	obavezni_izborni	varchar(10)	utf8mb4_general_ci		No	None			Change  Drop  Primary  Unique  More

Tabela studenti:

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
<input type="checkbox"/>	1	id			No	None		AUTO_INCREMENT	Change  Drop  Primary  More
<input type="checkbox"/>	2	ime	utf8mb4_general_ci		No	None			Change  Drop  Primary  More
<input type="checkbox"/>	3	ime_roditelja	utf8mb4_general_ci		No	None			Change  Drop  Primary  More
<input type="checkbox"/>	4	prezime	utf8mb4_general_ci		No	None			Change  Drop  Primary  More
<input type="checkbox"/>	5	broj_indeksa	utf8mb4_general_ci		No	None			Change  Drop  Primary  More
<input type="checkbox"/>	6	godina_studija			No	None			Change  Drop  Primary  More
<input type="checkbox"/>	7	jmbg			No	None			Change  Drop  Primary  More
<input type="checkbox"/>	8	datum_rodjenja			No	None			Change  Drop  Primary  More
<input type="checkbox"/>	9	espb			No	None			Change  Drop  Primary  More
<input type="checkbox"/>	10	prosek_ocena			No	None			Change  Drop  Primary  More
<input type="checkbox"/>	11	broj_telefona	utf8mb4_general_ci		No	None			Change  Drop  Primary  More
<input type="checkbox"/>	12	email	utf8mb4_general_ci		No	None			Change  Drop  Primary  More

## Konekcija sa bazom

Na vrhu fajla "app.py" potrebno je kreirati promenljivu koja predstavlja konekciju sa sql bazom podataka pomoću funkcije connect objekta mysql.connector.

```
konekcija = mysql.connector.connect(
    passwd="root", # lozinka za bazu
    user="phpmyadmin", # korisničko ime
    database="evidencija_studenata", # ime baze
    port=3306, # port na kojem je mysql server
    auth_plugin='mysql_native_password' # ako se koristi mysql 8.x
)
```

Takođe, treba kreirati kursor koji služi za izvršenje upita nad bazom.

```
kursor = konekcija.cursor(dictionary=True)
```

## CRUD

CRUD (create, read, update, delete) je pojam koji označava osnovne operacije nad resursima u bazi podataka. Potrebno je napraviti ove funkcionalnosti za sve entitete u aplikaciji (studenti, predmeti, korisnici, ocene) da bi korisnik imao potpunu kontrolu nad bazom podataka.

## Kreiranje korisnika (Create)

U fajlu "korisnik\_novi.html" potrebno je kreirati formu sa metodom POST i akcijom koja vodi na rutu (funkciju) za kreiranje korisnika.

```
<form action={{ url_for('korisnik_novi') }} method="POST">
# sadržaj
# forme
```

```
</form>
```

Na osnovu tabele u bazi može se videti da forma treba da ima sledeća polja za unos podataka:

ime, prezime, email i lozinka. Polja za unos treba da imaju odgovarajuće vrednosti atributa "name". Na primer:

```
<input type="email" class="form-control" name="email">
```

Na kraju forme potrebno je dugme za prosleđivanje forme backend-u.

```
<button type="submit" class="btn btn-primary">Sačuvaj</button>
```

Sada je potrebno obraditi prosleđene podatke i uneti ih u bazu. Ta logika je deo fajla "app.py".

Prvo treba kreirati rutu "/korisnik\_novi" koja podržava 2 metode, GET i POST.

```
@app.route("/korisnik_novi", methods=["GET", "POST"])
def korisnik_novi():
```

GET metoda služi da se korisniku prikaže forma, a metoda POST služi za obradu podataka nakon prosleđivanja forme. Za ispitivanje kojom metodom je korisnik došao do rute koristi se objekat request iz Flask biblioteke.

```
if request.method == "GET":
    # algoritam za GET metodu
elif request.method == "POST":
    # algoritam za POST metodu
```

Za GET metodu dovoljno je vratiti template "korisnik\_novi.html".

```
return render_template("korisnik_novi.html")
```

Za POST metodu potrebno je obraditi podatke i uneti ih u bazu i to u sledećim koracima:

- Pristupiti podacima iz forme
- Hash-ovati lozinku korisnika radi bolje sigurnosti
- Napisati sql upit za unos podataka
- "Spojiti" podatke i upit
- Izvršiti upit

Podacima se može pristupiti preko objekta request i njegovog ključa form. Za hash-ovanje lozinke koristi se funkcija generate\_password\_hash iz biblioteke werkzeug.security koja se instalira zajedno sa Flask-om (nije potrebna dodatna instalacija). Zatim je potrebno formatirati dobijene vrednosti u obliku tuple tipa podatka.

```
# na vrhu fajla, uvoz biblioteke
from werkzeug.security import generate_password_hash, check_password_hash
```

```
# unutar funkcije korisnik_novi(), unutar grane za obradu POST metode
forma = request.form
hesovana_lozinka = generate_password_hash(forma["lozinka"])
```

```
vrednosti = (  
    forma["ime"],  
    forma["prezime"],  
    forma["email"],  
    hesovana_lozinka,  
)
```

SQL upit izgleda ovako

```
upit = """ INSERT INTO  
    korisnici(ime,prezime,email,lozinka)  
    VALUES (%s, %s, %s, %s)  
    """
```

gde su %s oznake da se na tom mestu očekuje string vrednost. Potrebno je voditi računa o rasporedu vrednosti u promenljivoj vrednosti i sql upitu, treba da budu identični.

Funkcijom execute() objekta kursor se izvršava upit nad bazom. A zatim je potrebno sačuvati izmene u bazi funkcijom commit() objekta konekcija.

```
kursor.execute(upit, vrednosti)  
konekcija.commit()
```

Na kraju je potrebno korisniku prikazati novokreiranog korisnika u tabeli na stranici "korisnici". Za to se koristi funkcija redirect() iz Flask biblioteke.

```
return redirect(url_for("korisnici"))
```

Prikaz celokupnog sadržaja rute "/korisnik\_novi"

```
@app.route("/korisnik_novi", methods=["GET", "POST"])  
def korisnik_novi():  
    if request.method == "GET":  
        return render_template("korisnik_novi.html")  
    elif request.method == "POST":  
        forma = request.form  
        upit = """ INSERT INTO  
            korisnici(ime,prezime,email,lozinka)  
            VALUES (%s, %s, %s, %s)  
            """  
        hesovana_lozinka = generate_password_hash(forma["lozinka"])  
        vrednosti = (  
            forma["ime"],  
            forma["prezime"],  
            forma["email"],  
            hesovana_lozinka,  
        )  
        kursor.execute(upit, vrednosti)  
        konekcija.commit()  
        return redirect(url_for("korisnici"))
```

## Iščitavanje korisnika (Read)

Kreirati rutu `/korisnici` sa jednom metodom, GET, i odgovarajuću funkciju `korisnici()`. U funkciji kreirati upit koji selektuje sve korisnike iz baze, izvršiti upit i prikazati na korisnike na stranici. Nakon izvršenja upita potrebno je pozvati funkciju `fetchall()` objekta `kursor` kako bi se dobili korisnici u obliku liste.

```
@app.route("/korisnici")
def korisnici():
    upit = "SELECT * FROM korisnici"
    kursor.execute(upit)
    korisnici = kursor.fetchall()
    return render_template("korisnici.html", korisnici=korisnici)
```

Šablonu `"korisnici"` proslediti listu korisnika kao drugi argument funkcije `render_template()`.

U šablonu `"korisnici"` u `<tbody>` elementu pomoću for petlje (jinja sintaksa) ispisati informacije svih korisnika i odgovarajuće akcije nad njima.

```
{% for korisnik in korisnici %}
    <tr>
        <td>{{ korisnik.ime }}</td>
        <td>{{ korisnik.prezime }}</td>
        <td>{{ korisnik.email }}</td>
        <td class="row">
            <a href={{ url_for("korisnik_izmena", id=korisnik.id) }}
              role="button" class="icon-btn orange mx-1">
                <i class="fas fa-edit"></i>
            </a>
            <form action={{ url_for("korisnik_brisanje", id=korisnik.id) }}
              method="POST" class="mx-1">
                <button class="icon-btn red">
                    <i class="fas fa-trash"></i>
                </button>
            </form>
        </td>
    </tr>
{% endfor %}
```

U šablonu je dobijena lista objekata `"korisnici"`. Svaki `"korisnik"` je objekat a njegovim vrednostima ključeva se može pristupiti kao `ime_objekta.ime_ključa`

Za prikazivanje stranice za izmenu korisnika se koristi `<a>` element koji vodi na funkciju `korisnik_izmena()`, kojoj se prosleđuje id korisnika.

Za brisanje korisnika se koristi forma čija akcija vodi na funkciju `korisnik_brisanje()` i takođe se prosleđuje id korisnika.

## Izmena korisnika (Update)

Kreirati rutu “/korisnik\_izmena/<id>” sa 2 metode, GET i POST.

```
@app.route("/korisnik_izmena/<id>", methods=["GET", "POST"])
def korisnik_izmena(id):
    # definicija funkcije
```

Za GET metodu je potrebno pronaći korisnika u bazi i prikazati njegove informacije u šablonu. Napisati upit, izvršiti ga nad bazom, parsirati objekat “korisnik” i prikazati ga na stranici. Za parsiranje objekta “korisnik” koristi se funkcija fetchone() objekta kursor, nakon izvršenja upita.

```
upit = "SELECT * FROM korisnici WHERE id=%s"
vrednost = (id,)
kursor.execute(upit, vrednost)
korisnik = kursor.fetchone()
return render_template("korisnik_izmena.html", korisnik=korisnik)
```

Potrebno je modifikovati šablon “korisnik\_izmena.html” kako bi prikazivao informacije korisnika kao vrednosti input polja. Na primer:

```
<input type="text" class="form-control" id="ime" name="ime"
value={{ korisnik.ime }} required>

<input type="text" class="form-control" id="prezime" name="prezime"
value={{ korisnik.prezime }} required>
```

Forma za izmenu korisnika treba da ima akciju

```
action={{ url_for('korisnik_izmena', id=korisnik.id) }}
```

i metodu method="POST" kako bi se prosledile nove informacije o korisniku.

Za metodu POST funkcije korisnik\_izmena potrebno je obraditi prosleđene podatke, napisati upit za izmenu podataka u bazi, izvršiti upit i sačuvati izmene u bazi.

```
upit = """UPDATE korisnici SET
    ime=%s,
    prezime=%s,
    email=%s,
    rola=%s,
    lozinka=%s
WHERE id=%s
"""

forma = request.form
vrednosti = (
    forma["ime"],
    forma["prezime"],
    forma["email"],
    forma["rola"],
```



```
        forma["lozinka"],
        id,
    )
    kursor.execute(upit, vrednosti)
    konekcija.commit()
    return redirect(url_for("korisnici"))
```

Kako bi u padajućoj listi bila selektovana stavka koja odgovara roli korisnika čiju izmenu želimo, potrebno je izmeniti option tagove na sledeći način:

```
<select id="rola" name="rola" class="form-control">
    <option value="" {% if korisnik.rola==" " %} selected {% endif %} >--Izaberi rolu--</option>
    <option value="administrator" {% if korisnik.rola=="administrator" %} selected {% endif %} >Administrator</option>
    <option value="profesor" {% if korisnik.rola=="profesor" %} selected {% endif %} >Profesor</option>
</select>
```

## Brisanje korisnika (Delete)

Kreirati rutu `"/korisnik_brisanje/<id>"` sa metodom POST, napisati upit za brisanje korisnika sa prosleđenim id-jem, izvršiti upit i sačuvati promenu u bazi.

```
@app.route("/korisnik_brisanje/<id>", methods=["POST"])
def korisnik_brisanje(id):
    upit = """
        DELETE FROM korisnici WHERE id=%s
    """
    vrednost = (id,)
    kursor.execute(upit, vrednost)
    konekcija.commit()
    return redirect(url_for("korisnici"))
```

Napistai CRUD operacije za preostale entitete u aplikaciji (studenti, predmeti, ocene).

Prilikom kreiranja, izmene i brisanja ocene potrebno je izmeniti prosečnu ocenu i ESPB odgovarajućeg studenta.

Primer dodavanja ocene:

```
@app.route("/ocena_nova/<id>", methods=["POST"])
def ocena_nova(id):

    # Dodavanje ocene u tabelu ocene
    upit = """
        INSERT INTO ocene(student_id, predmet_id, ocena, datum)
        VALUES(%s, %s, %s, %s)
```

```
"""
    forma = request.form
    vrednosti = (id, forma['predmet_id'], forma['ocena'],
forma['datum'])
    kursor.execute(upit, vrednosti)
    konekcija.commit()

    # Računanje proseka ocena
    upit = "SELECT AVG(ocena) AS rezultat FROM ocene WHERE
student_id=%s"
    vrednost = (id,)
    kursor.execute(upit, vrednost)
    prosek_ocena = kursor.fetchone()

    # Računanje ukupno espb
    upit = "SELECT SUM(espb) AS rezultat FROM predmeti WHERE id IN
(SELECT predmet_id FROM ocene WHERE student_id=%s)"
    vrednost = (id,)
    kursor.execute(upit, vrednost)
    espb = kursor.fetchone()

    # Izmena tabele student
    upit = "UPDATE studenti SET espb=%s, prosek_ocena=%s WHERE id=%s"
    vrednosti = (espb['rezultat'], prosek_ocena['rezultat'], id)
    kursor.execute(upit, vrednosti)
    konekcija.commit()
    return redirect(url_for('student', id=id))
"""
```

## Login i Logout

Zbog sigurnosti aplikacije potrebno je implementirati funkcionalnosti prijave i odjave na početnoj stranici. Za to se koristi sesija (session) koja predstavlja interval u kome je korisnik prijavljen. Podaci o sesiji se čuvaju na serveru u privremenom folderu. Prilikom prijavljivanja korisnika podaci se beleže u sesiju, a prilikom odjave se brišu iz sesije. Koristi se objekat session iz biblioteke Flask.

Flask sesija je enkriptovana i potrebno je kreirati tajni ključ aplikacije pri vrhu "app.py", a nakon dela gde se vrši deklaracija Flask aplikacije:

```
app = Flask(__name__)
app.secret_key = "tajni_kljuc_aplikacije"
```

Kreirati rutu "/login" sa 2 metode, GET i POST. Za GET metodu dovoljno je iscrtati "login.html" šablon sa formom čija akcija vodi na "/login" rutu metodom POST i prosleđuje joj email i lozinku.

Za POST metodu potrebno je parsirati podatke iz forme, pronaći korisnika u bazi sa prosleđenim emailom i uporediti njegovu lozinku sa unešenom lozinkom. Kako je lozinka hash-ovana koristi se funkcija check\_password\_hash za upoređivanje. Ako se unešeni email i lozinka podudaraju sa onima iz baze, korisnik se upisuje u sesiju, a zatim redirect-uje na željenu stranicu. U suprotnom se vraća na stranicu "login".

```
forma = request.form
upit = "SELECT * FROM korisnici WHERE email=%s"
vrednost = (forma["email"],)
kursor.execute(upit, vrednost)
korisnik = kursor.fetchone()
if check_password_hash(korisnik["lozinka"], forma["lozinka"]):
    # upisivanje korisnika u sesiju
    session["ulogovani_korisnik"] = str(korisnik)
    return redirect(url_for("studenti"))
else:
    return render_template("login.html")
```

Kreirati rutu "/logout" na kojoj se korisnik briše iz sesije.

```
@app.route("/logout")
def logout():
    session.pop("ulogovani_korisnik", None)
    return redirect(url_for("login"))
```

Kreirati globalnu funkciju ulogovan() koja proverava da li je korisnik prijavljen, odnosno da li je upisan u sesiju.

```
def ulogovan():
    if "ulogovani_korisnik" in session:
```

```
        return True
    else:
        return False
```

Na svim rutama, osim login i logout, treba proveriti da li je korisnik ulogovan. Ako jeste, dozvoljeno je da se izvrši zahtevana akcija, u suprotnom posetilac sajta se redirect-uje na stranicu "login".

Primer:

```
@app.route("/student_novi", methods=["GET", "POST"])
def student_novi():
    if ulogovan():
        if request.method == 'GET':
            # obrada GET metode
        elif request.method == 'POST':
            # obrada POST metode
    else:
        return redirect(url_for('login'))
```

## Dodatne funkcionalnosti

### Korisničke role

Potrebno je kreirati odvojene korisničke role administratora i profesora. Administrator bi trebalo da ima potpunu kontrolu nad aplikacijom, odnosno ima pristup svim entitetima u aplikaciji. Profesor bi trebalo da može da pristupi samo sekciji "Studenti", da vidi listu studenata i da dodeljuje ocene studentima, ali ne i da menja njihove podatke.

U bazi podataka u tabeli "korisnici" dodati kolonu "rola". U njoj se upisuje jedna od vrednosti "administrator" ili "profesor".

Na stranicama za kreiranje i izmenu profesora potrebno je dodati polje za unos korisničke role.

```
<div class="form-group">
    <label for="rola">Rola</label>
    <select id="rola" name="rola" class="form-control">
        <option value="administrator">Administrator</option>
        <option value="profesor">Profesor</option>
    </select>
</div>
```

Adaptirati funkcije korisnik\_novi() i korisnik\_izmena() tako da se polje rola čuva u bazi podataka.

Prilikom kreiranja Login funkcionalnosti napravljeno je da se podaci o korisniku čuvaju u sesiji. To se može iskoristiti za proveru korisničke role trenutno ulogovanog korisnika.

Kreirati funkciju `rola()` čija je povratna vrednost rola ulogovanog korisnika.

```
def rola():
    if ulogovan():
        return ast.literal_eval(session["ulogovani_korisnik"]).pop("rola")
```

`ast` je standardna python biblioteka i ne zahteva dodatnu instalaciju. U ovom slučaju se koristi njena funkcija `literal_eval()` za pretvaranje stringa u dictionary.

Na svim rutama, osim na rutama za prikazivanje studenata i upravljanje njihovim ocenama, potrebno je onemogućiti pristup profesoru. To se može učiniti proverom da li je ulogovani korisnik profesor i ukoliko jeste `redirect`-ovati ga na rutu `/studenti`.

```
@app.route("/student_novi", methods=["GET", "POST"])
def student_novi():
    if rola() == "profesor":
        return redirect(url_for("studenti"))
    if ulogovan():
        # logika ukoliko je administrator ulogovan
```

Na ruti `/studenti` treba sakriti linkove do liste predmeta i liste korisnika ukoliko je ulogovani korisnik profesor. To se može uraditi prosleđivanjem korisničke role šablonu `"studenti.html"` i zatim uz pomoć jinja sintakse sakriti linkove.

```
# u app.py, u funkciji studenti()
return render_template(
    "studenti.html",
    studenti=studenti,
    rola=rola()
)
```

```
# u šablonu studenti.html
{% if (rola == 'administrator') %}
<li class="nav-item">
    <a class="nav-link" href={{ url_for('predmeti') }}>Predmeti</a>
</li>
<li class="nav-item">
    <a class="nav-link" href={{ url_for('korisnici') }}>Korisnici</a>
</li>
{% endif %}
```

Na isti način sakriti akcije nad studentima (izmena i brisanje). Ovu logiku je potrebno primeniti i na ruti `/student/<id>`.

## Dijalog modal za brisanje

Kako ne bi došlo do slučajnog brisanja reda iz baze podataka, poželjno je tražiti potvrdu korisnika da zaista želi da obriše taj red. To se može uraditi uz pomoć modala. U ovom primeru modal će se primeniti u tabeli `studenti`.

Uz svakog studenta, odnosno red u tabeli, dodaje se modal sa unikatnim id-jem koga pokreće (trigger-uje) crveno dugme iz kolone “Akcije”.

```
<div class="modal fade" id="deleteModal{{student.id}}" tabindex="-1" aria-
hidden="true">
<div class="modal-dialog modal-dialog-centered">
<div class="modal-content">
<div class="modal-body">
    Da li ste sigurni da želite da obrišete studenta {{ student.ime }}
    {{ student.prezime }}?
</div>
<div class="modal-footer">
    <button type="button" class="btn btn-secondary" data-
dismiss="modal">Odustani</button>
<a href={{ url_for('student_brisanje', id=student.id) }} role="button"
class="btn btn-danger">Obriši</a>
</div>
</div>
</div>
</div>
```

id modala je deleteModal{{student.id}}, te tako će id-jevi biti deleteModal1, deleteModal2...

Modal se sastoji od pitanja “Sigurno želite da obrišete studenta *Ime Prezime?*”, dugmeta koje služi za odustajanje od brisanja i zatvara modal i linka koji vodi na funkciju student\_brisanje() i prosledjuje joj id studenta.

Potrebno je izmeniti crveno dugme iz kolone “Akcije” kako bi pokretalo modal sa određenim id-jem.

```
<button class="icon-btn red mx-1" data-toggle="modal" data-
target="#deleteModal{{student.id}}">
    <i class="fas fa-trash"></i>
</button>
```

U kodu, lista studenata sada izgleda ovako

```
{% for student in studenti %}
<tr>
    <td>{{ student.broj_indeksa }}</td>
    <td>{{ student.ime }}</td>
    <td>{{ student.prezime }}</td>
    <td>{{ student.godina_studija }}</td>
    <td>{{ student.espb }}</td>
    <td>{{ student.prosek_ocena }}</td>
    <td>
        <a href={{ url_for('student', id=student.id) }} role="button"
class="icon-btn blue mx-1"><i>
```

```

        class="far fa-eye"></i></a>
        {% if (rola == 'administrator') %}
        <a href={{ url_for('student_izmena', id=student.id) }} role="button"
        class="icon-btn orange mx-1"><i class="fas fa-edit"></i></a>
        <button class="icon-btn red mx-1" data-toggle="modal" data-
target="#deleteModal{{student.id}}">
        <i class="fas fa-trash"></i>
        </button>
        {% endif %}
    </td>
</tr>
<!-- Modal -->
<div class="modal fade" id="deleteModal{{student.id}}" tabindex="-1" aria-
hidden="true">
    <div class="modal-dialog modal-dialog-centered">
        <div class="modal-content">
            <div class="modal-body">
                Sigurno želite da obrišete studenta {{ student.ime }} {{
student.prezime }}?
            </div>
            <div class="modal-footer">
                <button type="button" class="btn btn-secondary" data-
dismiss="modal">Odustani</button>
                <a href={{ url_for('student_brisanje', id=student.id) }}
role="button"
                class="btn btn-danger">Obriši</a>
            </div>
        </div>
    </div>
</div>
{% endfor %}

```

Ovo treba primeniti svuda gde postoji funkcionalnost brisanja (studenti, predmeti, korisnici, ocene).

Na stranici sa detaljima o studentu postoje funkcionalnosti brisanja studenta i brisanja ocena, tako treba voditi računa o davanju id-ja modalima kako ne bi došlo do podudaranja. Na primer, modal za brisanje studenta može imati id "deleteStudent", a modal za brisanje ocene "deleteOcena{{ocena.id}}".

## Slika studenta

Na stranici "student\_novi" treba dodati polje za dodavanje slike studenta. Takođe, na stranici "student\_izmena" treba dodati polje za izmenu, odnosno dodavanje, slike. Na stranici "student" treba dodati pregled slike studenta.

```

<!-- student_novi -->
<div class="form-group">
    <label for="slika">

```

```

</label>
<label for="slika" class="btn btn-primary">Odaberi sliku</label>
<input type="file" accept="image/*" class="custom-file-input" hidden
id="slika" name="slika">
</div>
```

Oznaka `<input>` je sakrivena i ima id "slika". Na to polje se postavlja fokus preko labele koja ujedno služi za prikaz slike.

Slika `placeholder.png` se prikazuje pre upload-a slike studenta.

```
<!-- student_izmena -->
<div class="form-group">
  <label for="slika">
    
  </label>
  <label for="slika" class="btn btn-primary">Odaberi sliku</label>
  <input type="file" accept="image/*" class="custom-file-input" hidden
id="slika" name="slika">
</div>
```

U šablonu "student\_izmena" if blok služi da se prikaže slika studenta ako postoji, u suprotnom se prikazuje `placeholder.png`.

Prilikom upload-a slike potrebno je promeniti `src` atribut oznake `<img id="slika_pregled" />`. Za tu funkcionalnost potrebno je koristiti JavaScript.

U fajlu "base.html", na dnu, ispod ostalih skripti dodati `<script>` oznaku i unutar nje skriptu za prikazivanje slike prilikom upload-a.

```
<script>
  $("#slika").on("change", function (e) {
    var reader = new FileReader();
    const file = e.target.files[0];
    reader.onload = function (e) {
      $("#slika_pregled").attr("src", e.target.result);
    };
    reader.readAsDataURL(file);
  });
</script>
```

U fajlu "style.css" dodati stil za prikazivanje slike.



```
#slika_pregled {  
  height: 150px;  
  width: 150px;  
  border-radius: 150px;  
  background-color: #999999;  
  object-fit: cover;  
}
```

U šablonu "student.html", u tabeli sa podacima o studentu, dodati sliku studenta.

```
<tr>  
  <td colspan="2" align="center">  
      
  </td>  
</tr>
```

Sledeće, potrebno je u app.py napraviti logiku za čuvanje slike. Prvo, u strukturi projekta, u folderu "static" kreirati folder "uploads". To je folder na serveru u kome će se čuvati slike studenata.

Pri vrhu "app.py" potrebno je konfigurisati aplikaciju da upload-ovane slike smešta u novokreirani folder.

```
# deklaracija upload direktorijuma  
UPLOAD_FOLDER = "static/uploads/"  
app.config["UPLOAD_FOLDER"] = UPLOAD_FOLDER  
app.config["MAX_CONTENT_LENGTH"] = 16 * 1024 * 1024
```

Na rutama "/student\_novi" i "/student\_izmena" napisati algoritam za čuvanje slika. Prvo se ispituje da li je slika poslata, a zatim se čuva u folderu "uploads". Takođe je potrebno u bazi podataka za svakog studenta čuvati ime njegove slike. U bazi podataka, u tabeli studenti, dodati kolonu "slika" tipa VARCHAR(255).

```
naziv_slike = ""  
if "slika" in request.files:  
    file = request.files["slika"]  
    if file.filename:  
        naziv_slike = forma["jmbg"] + file.filename  
        file.save(os.path.join(app.config["UPLOAD_FOLDER"], naziv_slike))
```

os je standardna python biblioteka koja ovde služi za pristup file system-u servera, odnosno u ovom slučaju, čuvanje slike.

Ime slike treba da bude unikatno i zato je korišćen JMBG studenta na koji je nadovezano ime fajla koji se upload-uje.

## Paginacija

Rastom aplikacije broj evidentiranih studenata se može višestruko uvećati. Zbog bržeg učitavanja i preglednosti stranice potrebno je napraviti paginaciju. Ovu funkcionalnost treba napraviti na stranicama "Studenti", "Predmeti" i "Korisnici".

U šablonu "studenti.html", ispod tabele sa studentima, dodati Bootstrap komponentu za paginaciju.

```
<nav>
    <ul class="pagination justify-content-center mt-3">
        <li class="page-item {% if strana=='1' %} disabled {% endif %}">
            <a class="page-link" href="{% prethodna_strana
}}">Prethodna</a>
        </li>
        <li class="page-item active"><a class="page-link" href="#">{{
strana }}</a></li>
        <li class="page-item">
            <a class="page-link" href="{% sledeca_strana %}">Sledeca</a>
        </li>
    </ul>
</nav>
```

Promenljive strana, prethodna\_strana i sledeca\_strana će biti prosledjene šablonu u funkciji render\_template().

Paginacija će se realizovati u SQL upitu na sledeći način:

```
upit = """
    SELECT * FROM studenti
    ...
    LIMIT 10 OFFSET %s
    """
```

LIMIT 10 označava da će se iz baze podataka iščitati 10 redova, dok OFFSET označava od kog reda se počinje. Vrednost OFFSET-a se računa na osnovu toga koju stranu korisnik želi da vidi.

```
offset = int(strana) * 10 - 10
```

U funkciji studenti() stranici će se pristupiti na sledeći način:

```
strana = request.args.get("page", "1")
```

args predstavlja listu parametara koji se šalju kroz url (na primer: "/studenti?page=2"). Vrednost "1" predstavlja vrednost parametra "page" ukoliko se taj parametar ne pošalje kroz url (default vrednost).

Potrebno je formirati link do prethodne i do sledeće stranice i dodeliti dobijene stringove promenljivama prethodna\_strana i sledeca\_strana.

```
prethodna_strana = ""
```

```
sledeca_strana = "/studenti?page=2"
if "=" in request.full_path:
    split_path = request.full_path.split("=")
    del split_path[-1]
    sledeca_strana = "=".join(split_path) + "=" + str(int(strana) + 1)
    prethodna_strana = "=".join(split_path) + "=" + str(int(strana) - 1)
```

Povratna vrednost funkcije `studenti()` sada izgleda ovako:

```
return render_template(
    "studenti.html",
    studenti=studenti,
    rola=rola(),
    strana=strana,
    sledeca_strana=sledeca_strana,
    prethodna_strana=prethodna_strana,
)
```

## Sortiranje

Potrebno je kreirati funkcionalnost sortiranja redova u tabelama na osnovu vrednosti kolona u bazi podataka.

`<thead>` oznaku je potrebno ugnjezditi u oznaku `<form>` čija je metoda "GET" i akcija `"url_for('studenti')"`.

Zatim modifikovati naslove kolona u tabeli tako da se sastoje od sakrivenog input polja čija je vrednost naziv kolone po kojoj treba sortirati redove i labele za to input polje.

```
<label for="order_by_broj_indeksa" class="header-btn">
    Broj indeksa
</label>
<input style="display: none;" id="order_by_broj_indeksa"
    value="broj_indeksa" class="header-btn"
    type="submit" name="order_by"
/>
```

Zbog boljeg korisničkog iskustva poželjno je dodati ikonice pored labele kao naznaku da se sortira po toj koloni.

```
{% if order_type == 'asc' and args.order_by == 'broj_indeksa' %}
<i class="fas fa-sort-up"></i>
{% elif order_type == 'desc' and args.order_by == 'broj_indeksa' %}
<i class="fas fa-sort-down"></i>
{% else %}
<i class="fas fa-sort light-icon"></i>
{% endif %}
```

Na kraju forme potrebno je dodati sledeća sakrivena input polja:

```
<input name="prethodni_order_by" value="{{ args.order_by }}" hidden />
```

```
<input name="order_type" value="{{ order_type }}" hidden />
<input name="page" value="{{ strana }}" hidden />
```

U funkciji `studenti()` potrebno je detektovati poslate vrednosti putem url-a. Zbog lakšeg pristupa parametrima `request.args` lista se može pretvoriti u dictionary.

```
args = request.args.to_dict()
```

`order_by` je promenljiva u kojoj se čuva naziv kolone po kojoj se vrši sortiranje i njena inicijalna vrednost je "id".

`order_type` je promenljiva u kojoj se čuva tip sortiranja: "ASC" (uzlazno) ili "DESC" (silazno) i inicijalna vrednost je "ASC".

```
order_by = "id"
order_type = "asc"
```

Ukoliko korisnik 2 puta zaredom odabere da sortira po određenoj koloni, prvi put bi trebalo da sortiranje bude uzlazno, a drugi put silazno. Detekcija ove funkcionalnosti se ostvaruje parametrom `prethodni_order_by` koji se prosleđuje u funkciju `studenti()`.

```
if "order_by" in args:
    order_by = args["order_by"].lower()
    if (
        "prethodni_order_by" in args
        and args["prethodni_order_by"] == args["order_by"]
    ):
        if args["order_type"] == "asc":
            order_type = "desc"
```

Zatim je potrebno izmeniti SQL upit tako da se podaci čitaju po određenom redosledu.

```
upit = """
        SELECT * FROM studenti
        ...
        ORDER BY %s %s
        LIMIT 10 OFFSET %s
    """
```

Vrednosti koje se prosleđuju upitu su:

```
vrednosti = (
    order_by,
    order_type,
    offset,
)
```

Povratna vrednost funkcije `studenti()` sada izgleda ovako:

```
return render_template(
    "studenti.html",
    studenti=studenti,
    rola=rola(),
```

```
        strana=strana,  
        sledeca_strana=sledeca_strana,  
        prethodna_strana=prethodna_strana,  
        order_type=order_type  
    )
```

## Pretraga i filtriranje

U šablonu "studenti.html", u tabeli, u formi, dodati još jednu oznaku <thead> iznad postojeće. Ona će sadržati red input (ili select) polja koja služe za pretragu entiteta na osnovu unetog pojma.

```
<thead>  
<tr>  
  
<th scope="col">  
    <div class="input-group input-group-sm">  
        <input value="{{ args.broj_indeksa }}" name="broj_indeksa"  
            placeholder="Broj indeksa" type="text" class="form-control">  
    </div>  
</th>  
  
<th scope="col">  
    <div class="input-group input-group-sm">  
        <select name="godina_studija" class="form-control">  
            <option value="">Sve</option>  
            <option {%if args.godina_studija == '1'%}selected{% endif %}  
                value="1">1</option>  
            <option {%if args.godina_studija == '2'%}selected{% endif %}  
                value="2">2</option>  
            <option {%if args.godina_studija == '3'%}selected{% endif %}  
                value="3">3</option>  
        </select>  
    </div>  
</th>  
  
<th scope="col">  
    <div class="inline">  
        <div class="input-group input-group-sm mr-3">  
            <input value="{{ args.prosek_ocena_od }}" name="prosek_ocena_od"  
                placeholder="Od" type="text" class="form-control">  
        </div>  
        <div class="input-group input-group-sm">  
            <input value="{{ args.prosek_ocena_do }}" name="prosek_ocena_do"  
                placeholder="Do" type="text" class="form-control">  
        </div>  
    </div>  
</th>
```

```
<th scope="col">
    <div class="inline-btns">
        <button type="submit" class="btn btn-primary btn-sm mr-3">Pretraži</button>
        <a role="button" class="btn btn-secondary btn-sm" href="{url_for('studenti') }}" >Poništi</button>
    </div>
</th>
</tr>
</thead>
```

args je dictionary prosleđen iz funkcije `render_template()` iz funkcije `studenti()` i sadrži prethodno poslate vrednosti za pretragu i filtriranje.

U funkciji `studenti()` inicijalne vrednosti za filtriranje će biti `"%"` (wildcard u SQL-u) ukoliko je upitanju tip string. Tako na primer:

```
s_ime = "%"
```

Za brojne tipove kao što je broj espb inicijalne vrednosti mogu biti:

```
s_espb_od = "0"
s_espb_do = "240"
```

Potrebno je proveriti da li su vrednosti za pretragu poslate i zatim ih izmeniti

```
if "ime" in args:
    s_ime = "%" + args["ime"] + "%"
```

Promenljiva vrednosti treba da izgleda ovako:

```
vrednosti = (
    s_ime,
    s_prezime,
    s_broj_indeksa,
    s_godina_studija,
    s_espb_od,
    s_espb_do,
    s_prosek_ocena_od,
    s_prosek_ocena_do,
    order_by,
    order_type,
    offset,
)
```

A SQL upit ovako:

```
upit = """
        SELECT * FROM studenti
        WHERE ime LIKE "%s" AND
```

```
prezime LIKE "%s" AND
broj_indeksa LIKE "%s" AND
godina_studija LIKE "%s" AND
espb >= %s AND espb <= %s AND
prosek_ocena >= %s AND prosek_ocena <= %s
ORDER BY %s %s
LIMIT 10 OFFSET %s
"""
```

Modifikovana povratna vrednost funkcije studenti()

```
return render_template(
    "studenti.html",
    studenti=studenti,
    rola=rola(),
    strana=strana,
    sledeca_strana=sledeca_strana,
    prethodna_strana=prethodna_strana,
    order_type=order_type,
    args=args,
)
```

## Slanje mejlova

Za slanje mejlova potrebno je instalirati biblioteku Flask-Mail.

```
pip install Flask-Mail
```

Na vrhu app.py potrebno je uvesti biblioteku, a zatim konfigurisati aplikaciju za slanje mejlova.

```
from flask_mail import Mail, Message

app.config["MAIL_SERVER"] = "smtp.gmail.com"
app.config["MAIL_PORT"] = 465
app.config["MAIL_USERNAME"] = "evidencija.atvss@gmail.com"
app.config["MAIL_PASSWORD"] = "atvss123loz"
app.config["MAIL_USE_TLS"] = False
app.config["MAIL_USE_SSL"] = True
mail = Mail(app)
```

Definisati funkciju send\_mail kojoj se prosleđuju podaci o korisniku kome se šalje mejl i koja na osnovu tih podataka šalje mejl.

```
def send_email(ime, prezime, email, lozinka):
    msg = Message(
        subject="Korisnički nalog",
        sender="ATVSS Evidencija studenata",
        recipients=[email],
    )
```

```
msg.html = render_template("email.html", ime=ime, prezime=prezime,
lozinka=lozinka)
mail.send(msg)
return "Sent"
```

Ovde je za telo poruke stavljen šablon "email.html" kome se prosleđuju ime, prezime i lozinka korisnika kome se salje mejl.

Šablon "email.html" izgleda ovako:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0"
/>
  <title>Evidencija studenata</title>
</head>
<body>
  <h2>Poštovani {{ime}} {{prezime}},</h2>
  <p>
    Za Vas je kreiran korisnički nalog na sajtu
    <a href="http://localhost:5000/">ATVSS Evidencija studenata</a>
  </p>
  <p>Vaša lozinka je: <code style="font-size:
18px">{{lozinka}}</code></p>
</body>
</html>
```

U funkciji korisnik\_novi(), nakon kreiranja korisnika, potrebno je pozvati funkciju send\_mail() sa podacima o novokreiranom korisniku. Na taj način se korisnik obaveštava da je za njega kreiran korisnički nalog u aplikaciji.

```
send_email(forma["ime"], forma["prezime"], forma["email"],
forma["lozinka"])
```

## Generisanje .csv fajlova

.csv(comma seperated values) je format fajla pogodan za čuvanje podataka na lokalnom računaru. Otvara ga veliki broj softvera i lako ga prevesti u .xlsx format.

U ovoj aplikaciji može služiti za izvoz (export) podataka o studentima, korisnicima i predmetima.

U šablonima sa tabelama, pored linka "Dodaj entitet", dodati link za export.

```
<a href={{ url_for('export', tip="studenti") }} role="button" class="btn
btn-success mr-3">Export</a>
```



Ovde je uzet primer za export studenata. Kod korisnika i predmeta, jedina razlika je vrednost parametra tip funkcije `url_for()`.

U `app.py` definisati novu rutu `"/export"` i njenu funkciju koja za parametar prihvata promenljivu `tip`.

Algoritam ove funkcije će biti:

- na osnovu tipa očitati potrebne podatke iz baze podataka
- kreirati fajl
- popuniti fajl iščitanim podacima
- poslati fajl korisniku na download

Izgleda ovako:

```
@app.route("/export/<tip>")
def export(tip):
    switch = {
        "studenti": "SELECT * FROM studenti",
        "korisnici": "SELECT * FROM korisnici",
        "predmeti": "SELECT * FROM predmeti",
    }
    upit = switch.get(tip)

    kursor.execute(upit)
    rezultat = kursor.fetchall()

    output = io.StringIO()
    writer = csv.writer(output)

    for row in rezultat:
        red = [ ]
        for value in row.values():
            red.append(str(value))
        writer.writerow(red)

    output.seek(0)

    return Response(
        output,
        mimetype="text/csv",
        headers={"Content-Disposition": "attachment;filename=" + tip +
            ".csv"},
    )
```

Biblioteke `io` i `csv` su standardne python biblioteke i nije potrebna instalacija, samo import na vrhu `app.py`.

`Response` klasa je deo Flask biblioteke i potrebno je uvesti je na vrhu `app.py`.

## Dodatni zadaci za studente

1. Kreirati korisničku rolu za studenta koji ima pristup samo svojim podacima. Svoje lične podatke može da pregleda i da menja, dok ocene može samo da pregleda. Prilikom prijave u aplikaciju student se redirect-uje na stranicu sa svojim profilom.
2. Omogućiti studentu da export-uje svoje ocene u .csv formatu.
3. Na stranici studenta, u tabeli sa ocenama, napraviti funkcionalnosti pretrage, filtriranja i sortiranja.

## Napredni zadaci za studente

Napravljena je aplikacija sa osnovnim funkcionalnostima za vođenje evidencije o studentima. Aplikaciju je moguće proširiti dodatnim funkcionalnostima radi boljeg korisničkog iskustva, kao što su:

- Slanje mejla sa verifikacijom prilikom kreiranja korisnika
- Resetovanje lozinke ukoliko je korisnik zaboravio istu
- Generisanje pdf fajlova sa podacima
- Captcha zaštita

## Reference

<https://www.tutorialspoint.com/python/index.htm>

<https://pythonbasics.org/>

<https://www.w3schools.com/python/default.asp>

<https://flask.palletsprojects.com/en/1.1.x/>

<https://jinja.palletsprojects.com/en/2.11.x/>

<https://getbootstrap.com/>

<https://fontawesome.com/>