Veb programiranje  
Izrada veb aplikacije u python programskom jeziku

NAPOMENA: u python-u 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 framework za kreiranje responsive html stranica
* FontAwesome - biblioteka ikonica

### Programski jezik python

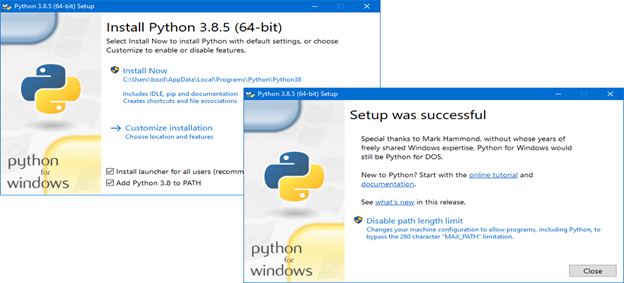
Programski jezik Python je objektno orijentisan, interpretativni programski jezik opšte namene. Autor programskog jezika Python je holandski programer Gvido van Rosum (Guido van Rossum). Jezik je nastao ranih devedesetih godina u slobodno vreme, kao jednonedeljni projekat razvoja modernog jezika koji 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. (aktuelna verzija)

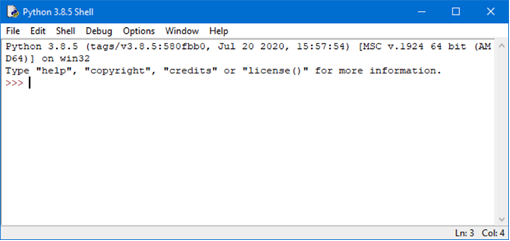
Programski jezik Python je relativno jednostavan. Međutim, oslanja se na veoma obimne i kvalitetne programske biblioteke. Standardna programska biblioteka Python programskog jezika (Python Standard Libary) obuhvata tipove i strukture podataka, ugrađene funkcije i obradu izuzetaka, kao i veliki broj međusobno 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.



### Osnovni elementi programskog jezika python

Program u Python programskom jeziku je niz naredbi, čiji je delimiter kraj linije teksta.

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 namenjenje osobama koje čitaju program, tako da su istvoremeno 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 deklarišu 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 pristupati 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 namenjenja za predstavljanje nizova i sekvenci objekata različitog tipa, ali se za razliku od liste, ne može 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 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 pokrente 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.

## 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 promt 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:

py -m pip install Flask

MariaDB:

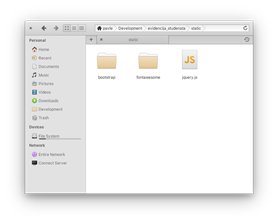
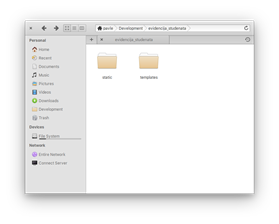
py -m pip install maraiadb

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.

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

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



**Napomene**

* Ako vec imate instaliran wamp nemojte instalirati xamp.
* Preimenovati folder za bootstrap i jquery file da se slazu sa putanjama u html fajlovima.

## 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 prestavljati polaznu tačku aplikacije. Sadržaće defiiniciju 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 mariadb

# 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)

U command promt-u se pozicionirajte unutar projekta. Komandom

python app.py

pokreće se aplikacija na lokalnom serveru, na adresi [http://127.0.0.1:5000](http://127.0.0.1:5000/), odnosno [http://localhost:5000](http://localhost:5000/)

**Napomene**

* Ako kopirate kod vodite racuna o tabulaciji, i prodjite kroz svaku liniju da biste je razumeli
* Umesto mysql, koristimo mariaDB konektor
* https://mariadb.com/resources/blog/how-to-connect-python-programs-to-mariadb/.
* Debug=True se koristi tokom izrade alikacije zbog olaksanja razvijanja, Vazno je ukolniti ovaj argument pri postavljanju aplikacije na internet iz sigurnosnih razloga.

## 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 paramater (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>

/predmeti  
/predmet\_novi

/predmet\_izmena/<id>

/predmet\_brisanje/<id>

/ocena\_nova/<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 aksije nad njima (izmena, brisanje)

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](mailto:<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-warning mx-1">

<i class="fas fa-trash"></i>

</a>

</td>

</tr>

</tbody>

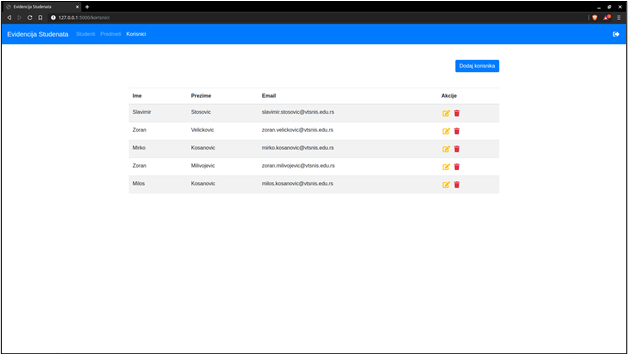
</table>

</div>

</div>

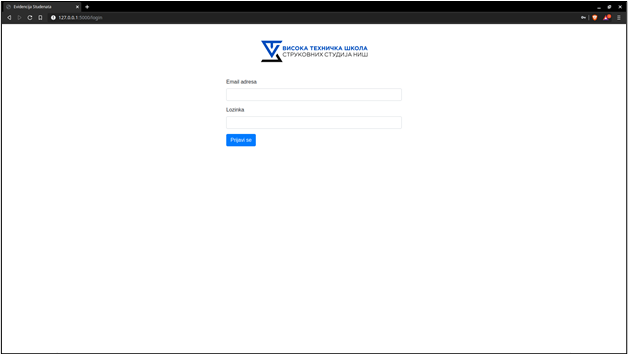
Ovde je definisano dugme koje vodi na stranicu za kreiranje korisnika i tabela sa korisnicima. Trenutno tabela sadrži samo jedan red kao primer. Potrebno je kasnije uz pomoć jinja-e u tabeli ispisati informacije o svim korisnicima iz baze podataka.

Kreirana stranica bi trebalo da izgleda ovako:



*korisnici.html*

Po ovom principu je potrebno napraviti i ostale stranice.



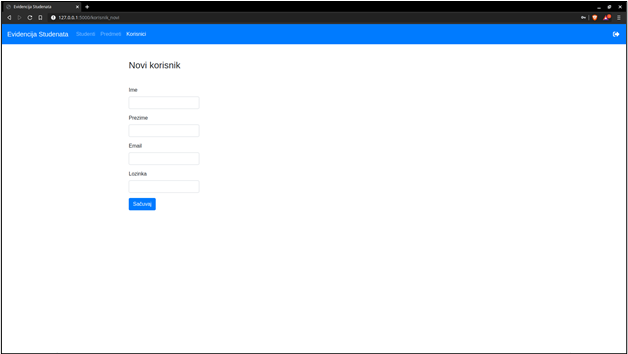
*Login.html*

*HTML kod za dodavanje slike:*

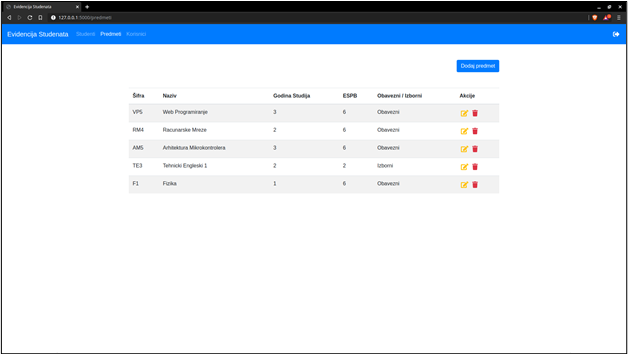
<div class="row justify-content-center mb-5 mt-5">

<img src="{{url\_for('static', filename='slike/Logo.png')}}" alt="VTS logo"/>

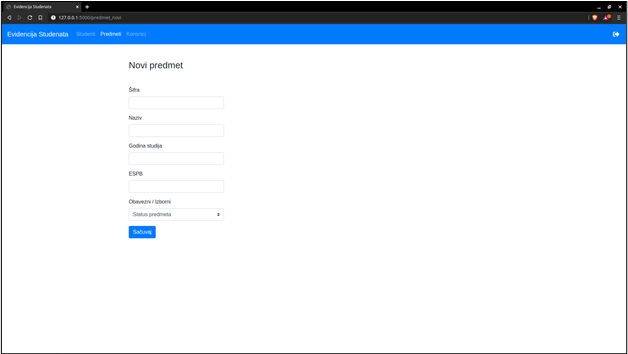
</div>



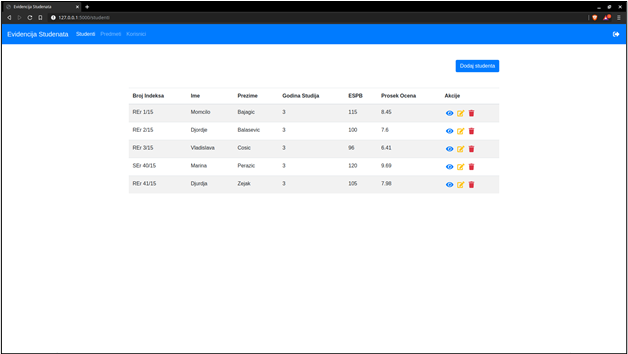
*korisnik\_novi.html*



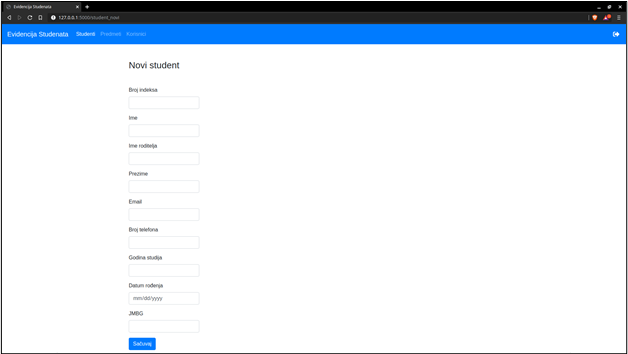
*predmeti.html*



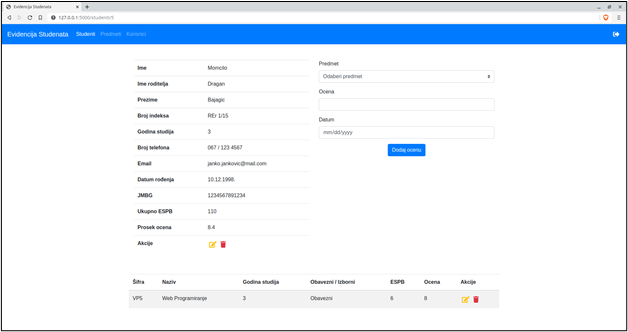
*predmet\_novi.html*



*studenti.html*



*student\_novi.html*



*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:

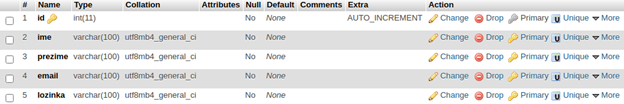
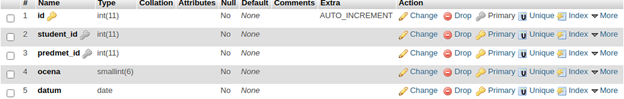


Tabela ocene:



*\*polja student\_id i predmet\_id su strani ključevi iz tabela studenti i predmeti*

Tabela predmeti:

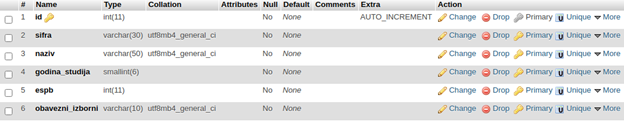
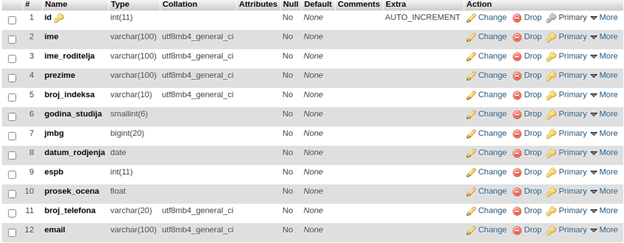


Tabela studenti:



## Konekcija sa bazom

Na vrhu fajla “app.py” potrebno je kreirati promenljivu koja predstavlja konekciju sa sql bazom podataka pomoću funkcije connect objekta mariadb

konekcija = mariadb.connect(

user="root",

password="",

host="localhost",

port=3306,

database="evidencija\_studenata"

)

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"))

Ruta “/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

hesovana\_lozinka = generate\_password\_hash(forma["lozinka"])

vrednosti =(

forma["ime"],

forma["prezime"],

forma["email"],

hesovana\_lozinka,

)

upit = """ INSERT INTO

korisnici(ime,prezime,email,lozinka)

VALUES (%s, %s, %s, %s)

"""

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="email" class="form-control" id="email" name="email" value={{ korisnik.email }}>

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.

forma = request.form

hashovana\_lozinka = generate\_password\_hash(forma["lozinka"])

vrednosti = (

forma["ime"],

forma["prezime"],

forma["email"],

hashovana\_lozinka,

id,

)

upit = """UPDATE korisnici SET

ime=%s,

prezime=%s,

email=%s,

lozinka=%s

WHERE id=%s

"""

kursor.execute(upit, vrednosti)

konekcija.commit()

return redirect(url\_for("korisnici"))

## 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”.

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 aplikaciju, 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 onemugićiti pristup profesoru. To se može učinit 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>.

### 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

### 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](mailto: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 salje mejl i koja na osnovu tih podataka salje 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 korisnku. Na taj način se se korisnik obaveštava da je za njega kreiran korisnički nalog u aplikaciji.

send\_email(forma["ime"], forma["prezime"], forma["email"], forma["lozinka"])