Универзитет у Београду
Математички факултет

ВЕБ ПРОГРАМИРАЊЕ У СРЕДЊОЈ ШКОЛИ

Семинарски рад

Студент: Давид Срећковић
Број индекса: 179/2022
Предмет: Методика наставе рачунарства и информатике
Професор: Сана Стојановић Ђурђевић

Београд, 2026.

Sadržaj

1 Увод

У претходном семинарском раду објашњен је појам шаблона и њихова улога у изради динамичких веб-страна. Шаблони омогућавају раздвајање логике апликације од њеног изгледа, што значајно олакшава одржавање и проширивање веб-сајтова. На тај начин створена је основа за разумевање савремених веб технологија и начина на који се подаци приказују кориснику.

Надовезујући се на тај рад, овај семинарски рад бави се библиотеком Flask, једним од најједноставнијих и најчешће коришћених веб оквира у програмском језику Python. Flask омогућава повезивање Python кода са HTML шаблонима, као и једноставну обраду HTTP захтева и креирање динамичких веб-страна.

Захваљујући својој једноставности и прегледној структури, Flask је веома погодан за примену у настави веб програмирања у средњој школи. У овом раду биће приказане основне карактеристике Flask библиотеке, начин дефинисања рута, рад са шаблонима и употреба статичких датотека, уз пример једноставне веб-апликације.

2 Прослеђивање података серверској апликацији

Веб-апликације стално примају податке од својих корисника. На пример, приликом логовања на систем корисник уноси и шаље своје корисничко име и лозинку, који се проверавају на серверу и, у зависности од резултата провере, кориснику се допушта или ускраћује приступ веб-апликацији. Слично томе, приликом претраге веб-сајта, параметри претраге се прослеђују серверској веб-апликацији, као што је случај код електронских веб-продавница, где корисник бира врсту производа и распон цена.

Уопштено, корисник у различитим ситуацијама уноси податке у формулар који се налази на веб-страни. Након попуњавања формулара, подаци се шаљу серверској апликацији, која их затим обрађује, на пример складишти у базу података.

2.1 GET параметри у претрази производа

У примеру претраге производа очекује се да се методом GET серверској скрипти проследе две вредности:

- `vrsta` – означава врсту производа који се претражују,
- `max_cena` – означава највишу дозвољену цену.

Пошто је могуће да се скрипта позове и без навођења параметара, у скрипти се користе подразумеване вредности. Метод `request.args.get` најчешће се позива са два

аргумента: први представља назив GET параметра, док други представља подразумевану вредност у случају да параметар није прослеђен. Уколико се подразумевана вредност не наведе, а параметар не постоји, функција враћа вредност `None`.

Датотека `app.py` има следећи садржај:

```
from flask import Flask, render_template, request

app = Flask(__name__)

@app.route("/pretraga")
def search():
    vrsta = request.args.get("vrsta", "svi proizvodi")
    max_cena = request.args.get("max_cena", float('inf'), type=float)
    return render_template("search.html",
                           vrsta=vrsta, max_cena=max_cena)
```

2.2 Подразумеване вредности и типови података

Ако параметар `vrsta` није прослеђен, његова вредност може бити постављена на подразумевану вредност, на пример `svi_proizvodi`. Уколико параметар `max_cena` није наведен, његова вредност може бити постављена на бесконачност, што се у програмском језику Python записује као `float("inf")`. На тај начин се обезбеђује да се не ограничава цена производа.

2.3 Шаблон и приказ података

Шаблон `templates/search.html` служи за приказ података који су прочитани из GET захтева. У реалној веб-апликацији, на основу ових параметара би се извршила претрага производа у бази података, а затим приказали одговарајући резултати.

Шаблон има следећи садржај:

```
<!DOCTYPE html>
<html>
   <head>
     <title>Pretraga proizvoda</title>
     <meta charset="utf-8" />
   </head>
   <body>
     <p>Vrsta proizvoda: {{ vrsta }}</p>
     <p>Maksimalna cena: {{ max_cena }}</p>
   </body>
</html>
```

2.4 Безбедност података

Не постоји гаранција да су сви подаци које веб-апликација прими безбедни и поуздани. Злонамерни корисник може проследити податке који садрже HTML или JavaScript код. Flask функција `render_template` аутоматски трансформише такве податке у безбедан облик, чиме се спречава њихово извршавање. Уколико се подаци не прослеђују овој функцији, препоручљиво је користити функцију `escape` из пакета `markupsafe`.

2.5 Вежба: Таблица множења

Као вежба, може се реализовати скрипта која приказује таблицу множења, при чему се димензија таблице задаје GET параметром `n`. Ако параметар није прослеђен, подразумева се вредност 10. Тако се посетом различитим URL адресама добијају таблице различитих димензија.

Датотека `app.py` има следећи садржај:

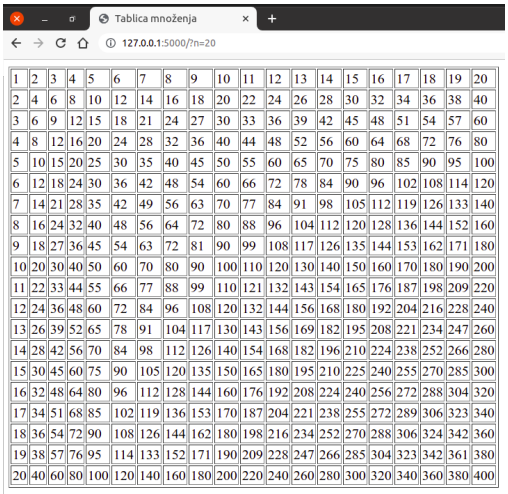
```
from flask import Flask, render_template, request

app = Flask(__name__)

@app.route("/")
def index():
    n = request.args.get("n", 10, int)
    return render_template("tablica.html", n=n)
```

```
<!DOCTYPE html>
<html>
   <head>
      <meta charset="utf-8" />
      <title>Tablica množenja</title>
   </head>
   <body>
     <table border="1">
     {% for i in range(1, n+1) %}
       <tr>
       {% for j in range(1, n+1) %}
       <td>{{ i*j }}</td>
       {% endfor %}
       </tr>
     {% endfor %}
     </table>
   </body>
</html>
```

2.6 Метода GET

Метода **GET** је један од основних начина слања података са клијента на сервер у оквиру протокола HTTP. Подаци се у овом случају шаљу као део URL адресе, након упитника (`?`), у облику парова `назив=вредност`. GET се најчешће користи за преузимање или претрагу података, јер не мења стање на серверу и дозвољава да се исти захтев поново пошаље без нежељених ефеката.

Пример URL адресе:



1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
2	4	6	8	10	12	14	16	18	20	22	24	26	28	30	32	34	36	38	40
3	6	9	12	15	18	21	24	27	30	33	36	39	42	45	48	51	54	57	60
4	8	12	16	20	24	28	32	36	40	44	48	52	56	60	64	68	72	76	80
5	10	15	20	25	30	35	40	45	50	55	60	65	70	75	80	85	90	95	100
6	12	18	24	30	36	42	48	54	60	66	72	78	84	90	96	102	108	114	120
7	14	21	28	35	42	49	56	63	70	77	84	91	98	105	112	119	126	133	140
8	16	24	32	40	48	56	64	72	80	88	96	104	112	120	128	136	144	152	160
9	18	27	36	45	54	63	72	81	90	99	108	117	126	135	144	153	162	171	180
10	20	30	40	50	60	70	80	90	100	110	120	130	140	150	160	170	180	190	200
11	22	33	44	55	66	77	88	99	110	121	132	143	154	165	176	187	198	209	220
12	24	36	48	60	72	84	96	108	120	132	144	156	168	180	192	204	216	228	240
13	26	39	52	65	78	91	104	117	130	143	156	169	182	195	208	221	234	247	260
14	28	42	56	70	84	98	112	126	140	154	168	182	196	210	224	238	252	266	280
15	30	45	60	75	90	105	120	135	150	165	180	195	210	225	240	255	270	285	300
16	32	48	64	80	96	112	128	144	160	176	192	208	224	240	256	272	288	304	320
17	34	51	68	85	102	119	136	153	170	187	204	221	238	255	272	289	306	323	340
18	36	54	72	90	108	126	144	162	180	198	216	234	252	270	288	306	324	342	360
19	38	57	76	95	114	133	152	171	190	209	228	247	266	285	304	323	342	361	380
20	40	60	80	100	120	140	160	180	200	220	240	260	280	300	320	340	360	380	400

Параметри се тумаче као ниске карактера. Уколико садрже специјалне карактере, они се кодирају, на пример размак се представља низом `%20`.

2.7 Приступ GET параметрима у Flask-у

Flask омогућава приступ GET параметрима коришћењем објекта `request`. Методом `request.args.get` могуће је прочитати вредност параметра и, по потреби, конвертовати је у одговарајући тип података.

2.8 Пренос података из формулара

Пренос података са клијента на сервер најчешће се реализује помоћу HTML формулара. Формулари садрже различите контроле за унос података, као што су поља за унос текста, падајуће листе, радио-дугмад и поља за потврду. Кликом на дугме типа `submit`, подаци се пакују у GET или POST захтев и шаљу серверу.

2.9 Пример: Калкулатор

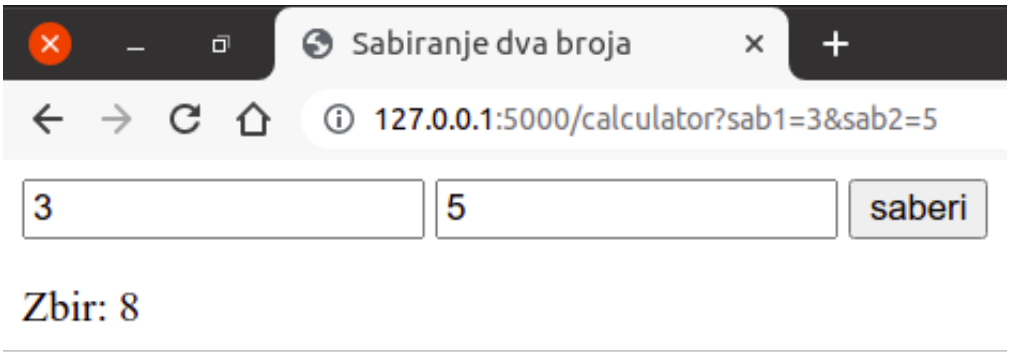
У овом примеру корисник уноси два броја у формулар, који се затим шаљу серверу методом GET. Сервер сабира унете вредности и враћа веб-страницу са приказом резултата.

Датотека `app.py` има следећи садржај:

```python
from flask import Flask, render_template, request

app = Flask(__name__)

@app.route("/calculator")
def calculator():
    sab1 = request.args.get("sab1", "")
    sab2 = request.args.get("sab2", "")
    try:
        zbir = int(sab1) + int(sab2)
    except:
        zbir = None
    return render_template("index.html", sab1=sab1, sab2=sab2, zbir=zbir)
```

Шаблон `templates/index.html` је дефинисан на следећи начин:

```html
<!DOCTYPE html>
<html>
   <head>
     <title>Sabiranje dva broja</title>
     <meta charset="utf-8" />
   </head>
   <body>
     <form>
       <input type="text" name="sab1" value="{{ sab1 }}" />
       <input type="text" name="sab2" value="{{ sab2 }}" />
       <input type="submit" value="saberi" />
     </form>
     {% if zbir != None %}
     <p>Zbir: {{ zbir }}</p>
     {% endif %}
   </body>
</html>
```



3 Метода POST

Метода **POST** се користи за слање података са клијента на сервер када је потребно променити стање на серверу или послати веће количине података. Подаци се шаљу у телу HTTP захтева, а не у URL-у, што омогућава већу безбедност и скривање осетљивих информација као што су лозинке или подаци формулара. POST се најчешће користи за слање података који требају бити обрађени или сачувани на серверу.

Генерална препорука за избор метода:

- **GET:** када желимо да узмемо податке са сервера без трајних промена.
- **POST:** када шаљемо податке на сервер и желимо да променимо стање на серверу (нпр. упис у базу).

HTML формулар који користи POST метод изгледа овако:

```
<form method="POST">
  ...
</form>
```

4 Путање са параметрима

Поред класичних GET параметара, који се у URL адреси наводе након упитника, Flask омогућава и коришћење **параметризованих путања**, где се поједини сегменти путање користе као параметри.

На пример, приказ одређеног производа помоћу класичних GET параметара може изгледати овако:

`http://www.moja-prodavnica.com/prikazi_proizvod?id=159`

Алтернативно, идентификатор производа може бити део саме путање:

`http://www.moja-prodavnica.com/prikazi_proizvod/159`

Овакав начин рутирања даје прегледније и читљивије URL адресе.

4.1 Параметризоване руте у Flask-у

Flask омогућава дефинисање рута које садрже промењиве сегменте. Ако се у путању наведе облик `/<ime>`, то значи да ће се вредност тог сегмента проследити функцији као параметар.

Пример једноставне Flask апликације:

```
from flask import Flask, render_template

app = Flask(__name__)

@app.route("/<ime>")
def pozdravi(ime):
    return render_template("index.html", ime=ime)
```

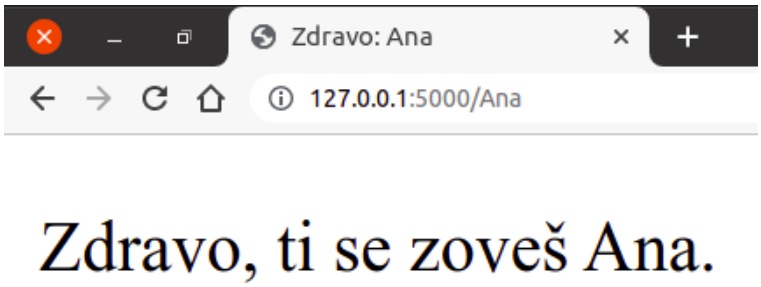
Ако се апликацији приступи преко URL адресе `http://127.0.0.1/Petar`, биће позвана функција `pozdravi("Petar")`, док ће приступом адреси `http://127.0.0.1/Ana` бити позвана функција `pozdravi("Ana")`. На тај начин се подаци извлаче директно из URL адресе и могу се даље користити у апликацији.

4.2 Приказ података у шаблону

Добијена вредност се може проследити HTML шаблону, који затим динамички приказује садржај странице. Шаблон може изгледати на следећи начин:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Zdravo: {{ ime }}</title>
    <meta charset="utf-8" />
  </head>
  <body>
    <p>Zdravo, ti se zoves {{ ime }}.</p>
  </body>
</html>
```

На слици је приказан изглед стране унутар прегледача веба.



Ако је рута дефинисана као `/pozdravi/<ime>`, поздравној страни се приступа преко URL адреса облика `http://127.0.0.1/pozdravi/Petar`.

4.3 Ограничење типа параметара

Flask омогућава и ограничавање типа података који се прихвата у путањи. На пример, ако желимо да параметар буде цео број, користи се тип `int`:

```
@app.route("/prikazi_proizvod/<int:id_proizvod>")
def prikazi_proizvod(id_proizvod):
    return f"ID proizvoda: {id_proizvod}"
```

Овој рути се може приступити преко URL адресе облика `http://www.moja-prodavnica.com/prikazi_proizvod/1234`.

4.4 Предности и ограничења

Параметризоване путање дају лепши и читљивији облик URL адреса у односу на класичне GET параметре. Међутим, овај приступ није практичан када постоји већи број параметара или када су неки од њих опциони.

Такође, уколико подаци долазе из HTML формулара, параметри се и даље читају помоћу објекта `request`, коришћењем метода `request.args.get(...)` или `request.form.get(...)`, у зависности од употребљене HTTP методе.

5 Колачићи

HTTP протокол је по својој природи **без стања**, што значи да се сваки пар захтев–одговор посматра као независан од претходних. Међутим, у савременим веб-апликацијама често постоји потреба да се одређени подаци памте током корисничке сесије. У ту сврху користе се различити механизми, од којих су најчешћи **колачићи** (eng. cookies) и **сесије** (eng. sessions).

Колачићи представљају мале количине података које сервер шаље клијенту, а клијент их чува и аутоматски шаље серверу у сваком наредном захтеву ка истом домену. Они се најчешће користе за чување привремених података, као што су:

- корисничко име улогованог корисника,
- садржај корпе у електронској продавници,
- подешавања сајта (језик, валута, тема приказа).

Ови подаци нису трајни и њихов губитак не представља озбиљан проблем, за разлику од података који се чувају у бази података (нпр. кориснички налози или објаве).

5.1 Пример употребе колачића у Flask-у

У следећем примеру корисник при првом приступу апликацији уноси своје име у формулар. Име се чува у колачићу, тако да се при сваком наредном приступу кориснику приказује поздравна порука. Корисник има могућност и да се „излогује“, чиме се колачић брише.

5.2 HTML шаблон

Шаблон `templates/index.html` приказује формулар или поздравну поруку, у зависности од тога да ли је прослеђена променљива `name`:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Kola i i</title>
    <meta charset="utf-8" />
  </head>
  <body>
      <h1>Welcome {{ name }}</h1>
      <a href="{{ url_for('resetcookie') }}">Logout</a>
      <form method="post" action="{{ url_for('setcookie') }}">
        <input type="text" name="name" />
        <input type="submit" value="login" />
      </form>
  </body>
</html>
```

5.3 Читање колачића

У главној путањи апликације проверава се да ли је колачић постављен. Ако јесте, његова вредност се чита и прослеђује шаблону:

```
@app.route("/")
def index():
    if "name" in request.cookies:
        name = request.cookies.get("name")
        return render_template("index.html", name=name)
    else:
        return render_template("index.html")
```

5.4 Постављање колачића

Колачић се поставља креирањем објекта одговора помоћу функције `make_response`, а затим позивом методе `set_cookie`. Након тога се корисник редиректује на почетну страну:

```
@app.route("/setcookie", methods=["POST"])
def setcookie():
    name = request.form["name"]
    response = make_response(redirect(url_for("index")))
    response.set_cookie("name", name)
    return response
```

Пошто се подаци шаљу формуларом, потребно је омогућити употребу HTTP методе POST.

5.5 Брисање колачића

Брисање колачића се врши тако што му се постави време истека једнако нули:

```
@app.route("/resetcookie")
def resetcookie():
    response = make_response(redirect(url_for("index")))
    response.set_cookie("name", "", expires=0)
    return response
```

5.6 Напомена о приватности

Колачићи се не користе искључиво за функционалност веб-апликација, већ и за праћење понашања корисника у сврху оглашавања. Због могућег нарушавања приватности, у Европској унији је прописано да веб-сајтови морају обавестити кориснике о употреби колачића и прибавити њихову сагласност.

6 Сесије

Колачићи омогућавају чување података на клијенту, унутар прегледача веба. Сесије, с друге стране, чувају податке на серверу и обично су ограниченог трајања. Током трајања сесије, сервер памти податке о кориснику (нпр. корисничко име), а након истека сесије или излоговања, приступ заштићеним деловима сајта се онемогућава.

Када се креира сесија, сервер шаље кориснику колачић сесије који је јединствен за сваку сесију. Наредни захтеви прегледача садрже овај колачић, па сервер може да идентификује сесију и приступљене податке. Колачићи сесије се бришу чим корисник затвори прегледач, а корисник не мора да брине о њиховом чувању.

HTML шаблон `templates/index.html` приказује формулар за унос имена или поздравну поруку ако је корисник улогован:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Sesije</title>
    <meta charset="utf-8" />
  </head>
  <body>
    {% if username %}
      <h1>Username: {{ username }}</h1>
      <a href="{{ url_for('logout') }}">Logout</a>
    {% else %}
      <form method="post" action="{{ url_for('login') }}">
        <input type="text" name="username" />
        <input type="submit" value="login" />
      </form>
  </body>
</html>
```

Кључ app.secret је неопходан за шифровање података у сесији и колачића који се шаљу на клијент. Вредност кључа треба бити тајна и може се генерисати у Python интерактивно помоћу:

```
import os
os.urandom(24)
```

Овај механизам омогућава да сервер сигурно прати улоговане кориснике без потребе да подаци буду видљиви на клијенту.

7 Коришћење база података из Flask апликација

На самом почетку нашег рада са базама података, рекли смо да је један од начина интеракције са системом за управљање базама података (СУБП) управо апликативни интерфејс.

Сваки сајт за пуштање музике, свака онлајн продавница, електронски дневник, свака апликација на мобилном уређају путем које се плаћају рачуни и тако даље, користи неку базу података. Да би то било могуће, неопходно је да постоји начин интеракције са СУБП који ће се уградити у апликацију која се креира. Програмери, када из својих програма приступају базама података, користе апликативни програмски интерфејс, АПИ (енгл. *application programming interface*).

Да би наши програми преко АПИ-ја могли да приступају СУБП-у, обично је потребно да у програм укључимо посебан модул, односно програмску библиотеку која имплементира АПИ за приступ бази.

Када је у питању програмски језик Python, већ у стандардној инсталацији програмског језика долази модул за SQLite који се зове `sqlite3`, и то са уграђеним СУБП за SQLite. То значи да, ако програмирате у програмском језику Python, није потребно ништа додатно да инсталирате да бисте користили SQLite. Модул `sqlite3` имплементира стандардну спецификацију АПИ-ја за приступ бази која се зове DB-API 2.0, што значи да када научите како у Python-у да користите SQLite, на сличан начин ћете моћи да користите и друге релационе базе.

Унутар Python кôда веб-апликације (код једноставних апликација он се налази у датотеци `app.py`) вршимо повезивање са базом података и читање података из базе, смештајући резултат упита у листу, која се затим прослеђује функцији `render_template`, при чему се у шаблону налази петља `for` која чита и приказује један по један податак из те листе.

7.1 Повезивање са базом и постављање упита

Као што смо већ нагласили, свака иоле сложенија веб-апликација укључује неколико различитих фајлова, па је логична конвенција да се све оне чувају унутар једног фолдера намењеног тој веб-апликацији. За први пример веб-апликације која се повезује на базу података направићемо фолдер под именом `01_database` и у њему креирати фајл `app.py` са програмским кôдом.

У исти фолдер стављамо и базу података. У овом примеру користимо базу података фиктивне компаније за продају музичких композиција `music.db`.

Да би се могло приступити SQLite бази података, потребно је прво повезати се са базом коришћењем функције `sqlite3.connect`, наводећи назив базе:

```
import os
import sqlite3
from flask import Flask

app = Flask(__name__)

conn = sqlite3.connect(os.path.join(app.root_path, 'music.db'))
# ...
conn.close()
```

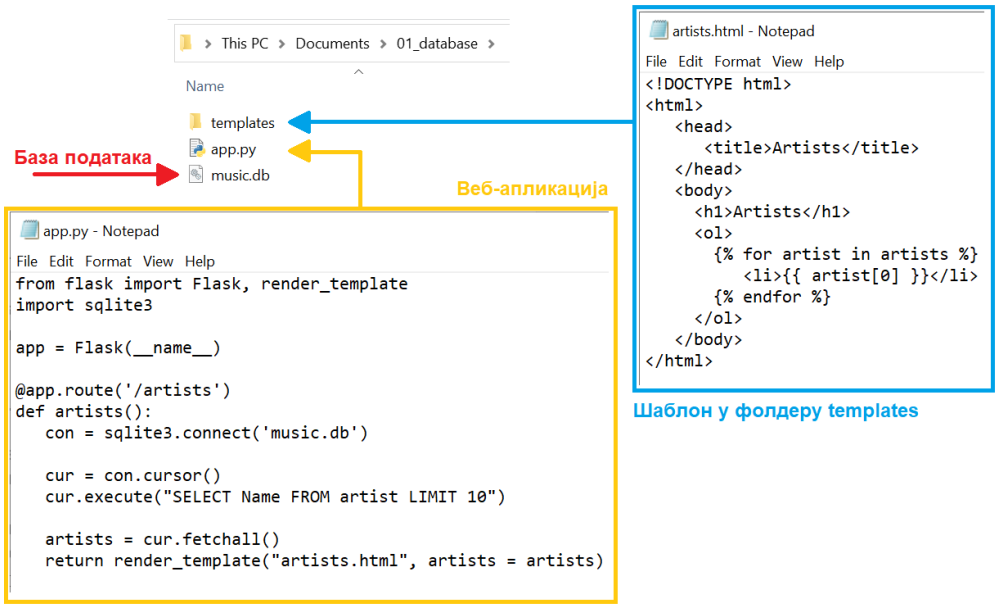
Шаблон `templates/artists.html` за приказ података може изгледати овако:

```
<!DOCTYPE html>
<html>
   <head>
      <title>Artists</title>
   </head>
   <body>
     <h1>Artists</h1>
     <ol>
       {% for artist in artists %}
          <li>{{ artist[0] }}</li>
       {% endfor %}
     </ol>
   </body>
</html>
```

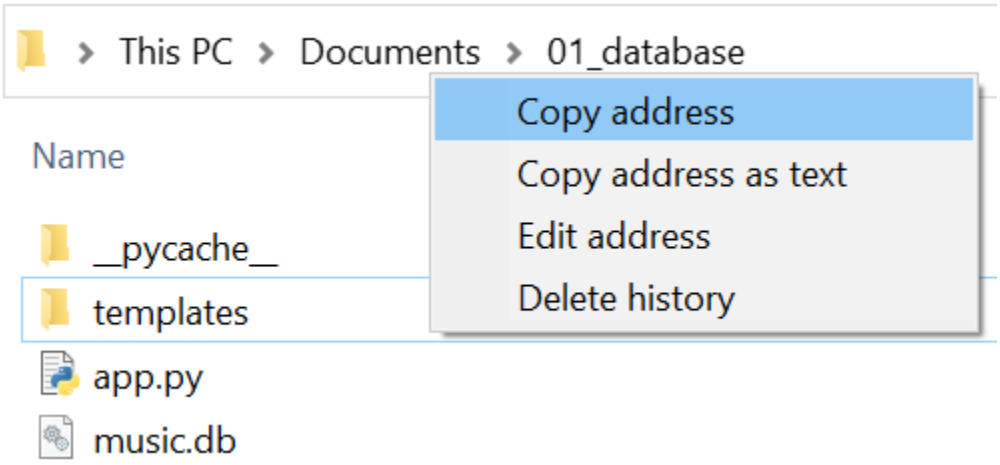
Комплетан пример app.py:

```
import sqlite3
from flask import Flask, render_template

app = Flask(__name__)

@app.route('/artists')
def artists():
    con = sqlite3.connect('music.db')
    cur = con.cursor()
    cur.execute("SELECT Name FROM artist LIMIT 10")
    artists = cur.fetchall()
    return render_template("artists.html", artists=artists)
```

Следећа слика илуструје фолдер у којем се налази наша веб-апликација.



Да бисмо имали тачну путању до фајла која нам је неопходна да покренемо програм, можемо да употребимо опцију Copy adress када урадимо десни клик мишем у прозору File Explorer.



На слици је приказан изглед апликације када се покрене http://127.0.0.1:5000/artists из прегледача веба:

7.2 Флексибилнија конекција и глобална променљива

Путању до базе можемо дефинисати преко `app.root_path`:

```
import os
import sqlite3
from flask import Flask
app = Flask(__name__)
conn = sqlite3.connect(os.path.join(app.root_path, 'music.db'))
conn.close()
```

Убудуће можемо користити глобалну променљиву `DATABASE`:

```
DATABASE = os.path.join(app.root_path, 'music.db')
```

Мало елегантније решење је да конекцију чувамо унутар објекта `flask.g`, на пример `g._db_conn`.

```
from flask import g

def get_db():
    if not "_db_conn" in g:
        g._db_conn = sqlite3.connect(DATABASE)
    return g._db_conn

@app.teardown_appcontext
def close_db(exception):
    if "_db_conn" in g:
        g._db_conn.close()
```

7.3 Помоћна функција за упите

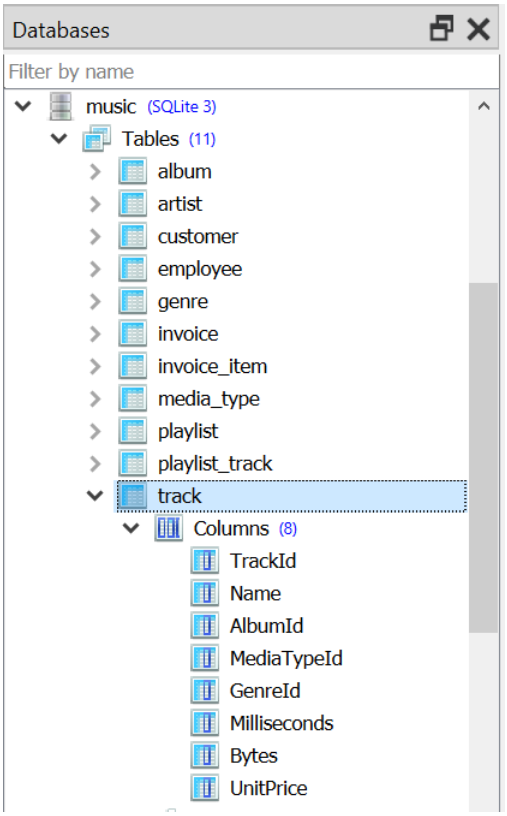
Функција `query_db` поједностављује рад са базом:

```
def query_db(query, args=(), one=False):
    cur = get_db().execute(query, args)
    rows = cur.fetchall()
    cur.close()
    if one:
        return rows[0] if rows else None
    else:
        return rows
@app.route("/artists")
def artists():
    artists = query_db("SELECT Name FROM Artist LIMIT 10")
    return render_template("artists.html", artists=artists)
```

Шаблон остаје исти као претходно.

7.4 Упознавање са базом

Приликом креирања веб-апликација које се повезују на базу података, важно је добро се упознати са том базом. Преузету SQLite базу `music.db` можемо отворити у програму SQLite Studio командом менија `Database → Add a database` и проверити списак табела и колона.



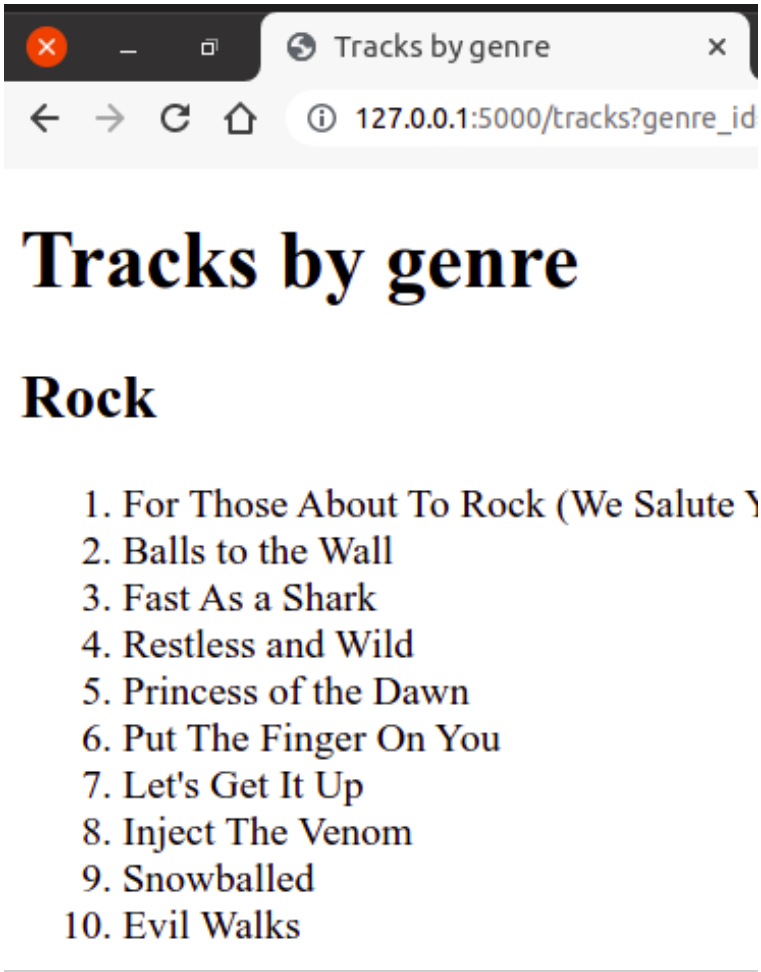
8 Пренос параметара упита

И у случају веб-апликација упити често зависе од неких параметара. На пример, желимо да прикажемо списак композиција одабраног жанра. Параметар је у том случају идентификатор жанра.

Претпоставимо да се идентификатор жанра прослеђује нашем серверском скрипту у склопу GET HTTP захтева. Страну ћемо отварати помоћу URL-ова облика `http://127.0.0.1:5000/tracks?genre_id=3`.

8.1 HTML шаблон за песме (tracks.html)

```
{% extends "index.html" %}
{% block content %}
    {% if error %}
        <h2>Error: {{ error_msg }}</h2>
    {% else %}
        <h2>{{ genre_name }}</h2>
        <ol>
            {% for track in tracks %}
                <li>{{ track["Name"] }}</li>
            {% endfor %}
        </ol>
    {% endif %}
{% endblock %}
```



8.2 HTML шаблон за жанрове (genres.html)

```
{% extends "index.html" %}
{% block content %}
    <ul>
        {% for genre in genres %}
            <li>
                <a href="{{ url_for('tracks_by_genre', genre_id=genre
                    .GenreId) }}">
                    {{ genre.Name }}
                </a>
            </li>
        {% endfor %}
    </ul>
{% endblock %}
```

8.3 Заједнички HTML шаблон (index.html)

```
<!DOCTYPE html>
<html>
    <head>
        <title>Tracks by genre</title>
    </head>
    <body>
        <h1>Tracks by genre</h1>
        {% block content %}
        {% endblock %}
    </body>
</html>
```

9 Формулари и упити ка бази података

Врло често се параметри упита ка бази података добијају на основу вредности које је корисник унео у HTML формулар. На пример, у оквиру текућег задатка, уместо листе линкова за одабир жанра, могуће је користити HTML формулар који садржи падајућу листу. На тај начин кориснику се омогућава једноставан и прегледан избор жељеног жанра.

Падајућа листа у језику HTML дефинише се помоћу елемената `select` и `option`. Пример једне такве листе приказан је у наставку:

```
<select name="genre_id">
  <option value="1">Rock</option>
  <option value="2">Jazz</option>
  <option value="3">Metal</option>
  ...
</select>
```

Поред падајуће листе, формулар треба да садржи и дугме типа `submit`, чијим притиском се шаље нови захтев серверу. Назив елемента `select`, односно вредност његовог атрибута `name`, постављен је на `genre_id`, док је вредност атрибута `value` сваког елемента `option` једнака идентификатору одговарајућег жанра.

На овај начин се у оквиру захтева шаље GET параметар `genre_id` чија је вредност идентификатор жанра који је корисник изабрао. Подсетимо се да су GET и POST параметри увек у облику парова `name=value`.

Према тренутној организацији апликације, захтев за приказ композиција не шаље се поново на путању `/genres`, већ на путању `/tracks`. Због тога је у оквиру HTML формулара потребно навести одговарајући атрибут `action`.

Имајући све наведено у виду, шаблон `genres.html` може се дефинисати на следећи начин. Уместо навођења апсолутне путање `/tracks`, користи се Flask функција `url_for`, којој се прослеђује назив функције `tracks_by_genre`. Овим приступом постиже се већа флексибилност апликације, јер ће она коректно функционисати и у случају промене путања.

```
{% extends "index.html" %}
{% block content %}
<form action="{{ url_for('tracks_by_genre') }}">
  <select name="genre_id">
    {% for genre in genres %}
    <option value="{{ genre.GenreId }}">{{ genre.Name }}</option>
    {% endfor %}
  </select>
  <input type="submit" value="Show" />
</form>
{% endblock %}
```

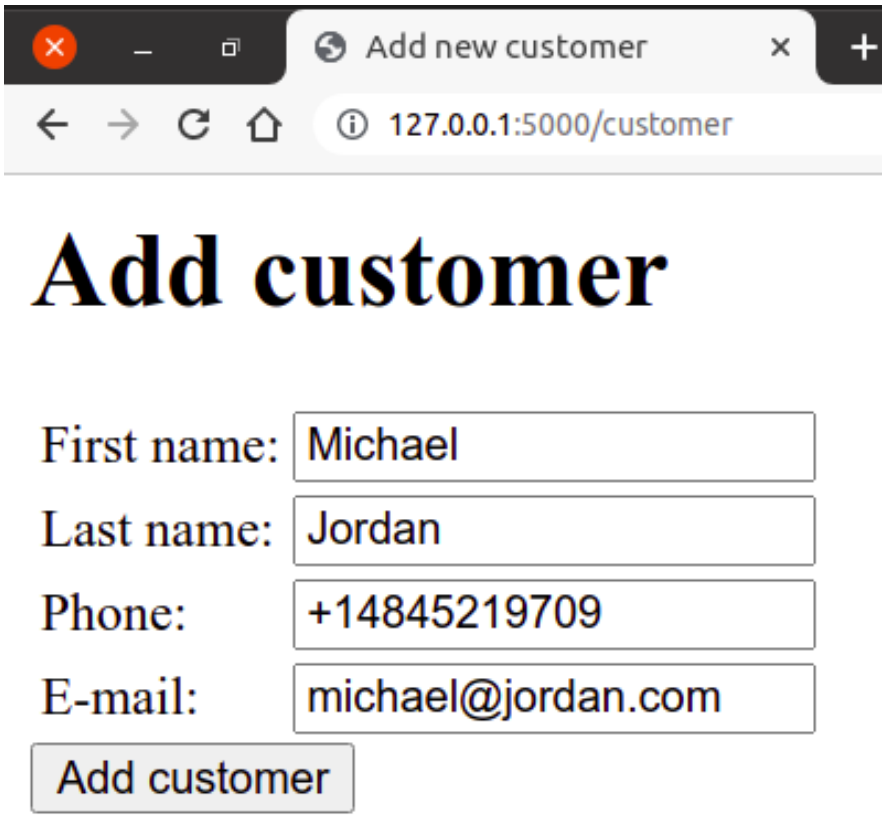
10 Упис података у базу

У претходним примерима приказано је читање података из базе података. У наставку ће бити приказан једноставан пример који илуструје како је могуће извршити упис података у базу из Flask апликације.

Прикажимо начин на који се може направити веб-страна која омогућава упис новог корисника у базу података. Претпоставићемо да се подаци о корисницима чувају у табели `customer`, као и да се за сваког корисника чувају следеће колоне: `CustomerId`, `ime`, `prezime`, `Phone` и `Email`. У бази података која је раније анализирана постојао је већи број колона, међутим, ради једноставности, у овом примеру нећемо се бавити осталим подацима који се углавном односе на адресу корисника.

Подаци који се уписују у базу најчешће се прикупљају коришћењем HTML формулара. Због тога је потребно креирати формулар у који корисник уноси своје личне податке, док се јединствени идентификатор корисника аутоматски додељује од стране базе података.

Изглед формулара за унос података о кориснику приказан је на следећој слици.



Након попуњавања формулара, притиском на дугме типа `submit` шаље се POST захтев серверу. Серверска апликација прихвата захтев, покушава да упише прослеђене

податке у базу података и, у зависности од исхода, приказује страницу са информацијом о томе да ли је упис био успешан или не. Након тога, кориснику се поново приказује формулар који омогућава упис новог корисника.

11 „Флешоване“ поруке

У претходној апликацији информација о томе да ли је упис података у базу био успешан прослеђивана је шаблону путем посебног параметра `msg`. Међутим, ситуације у којима је потребно кориснику приказати једнократне поруке, као што су обавештења о грешци или успешном извршавању неке операције, веома су честе у веб-апликацијама. Због тога библиотека Flask нуди посебан механизам за рад са таквим порукама, познат као механизам „флешованих“ порука.

Флешоване поруке (енгл. *flashed messages*) представљају кратке текстуалне поруке које се кориснику приказују једнократно, након чега нестају. Назив потиче од енглеске речи *flash*, која означава кратак бљесак. Поруке се постављају на серверској страни позивом функције `flash`, док се њихов приказ реализује унутар HTML шаблона.

Функција `flash` као аргумент прима текст поруке, а опционо и категорију поруке, као што су `error` или `ok`. Унутар шаблона, све тренутно активне флешоване поруке могу се преузети позивом функције `get_flashed_messages`. Када се користи опција `with_categories=True`, ова функција враћа листу уређених парова, где први елемент представља категорију поруке, а други њен текст.

При коришћењу наслеђивања шаблона, пожељно је да се приказ флешованих порука дефинише у основном шаблону апликације. На тај начин се обезбеђује да се све поруке приказују на истом месту, без обзира на то из које серверске функције потичу, док се појединачни шаблони ослобађају потребе да се баве њиховим приказом.

12 Валидација података унетих у формулар

У претходној верзији апликације корисник је у формулар могао да унесе произвољне податке, па чак и да формулар остави празним, а да се ти подаци ипак упишу у базу података. Овакво понашање представља лошу праксу, јер доводи до неконзистентних и некоректних података у систему. Због тога квалитетне веб-апликације увек проверавају елементарну исправност унетих података пре њихове даље обраде и уписа у базу.

Поступак провере података унетих у формулар назива се *валидација формулара*. Валидацију је могуће вршити на страни клијента, пре слања података серверу, као и на страни сервера, пре њихове обраде. Иако клијентска валидација смањује оптерећење

сервера и побољшава корисничко искуство, она никада не сме бити једини облик провере података.

12.1 Клијентска валидација уз помоћ HTML 5

Савремени HTML 5 стандард омогућава дефинисање основних правила коректности директно унутар формулара, без употребе програмског језика JavaScript. Најчешћи облици клијентске валидације обухватају:

- Обележавање обавезних поља атрибутом `required`, чиме се онемогућава слање формулара ако поље није попуњено.
- Ограничавање дужине унетог текста помоћу атрибута `minlength` и `maxlength`.
- Дефинисање формата дозвољених ниски употребом регуларних израза кроз атрибут `pattern`.

Регуларни изрази представљају формализам за опис формата ниски. На пример, израз `[a-z]+` описује реч која се састоји од једног или више малих слова, док израз `[0-9]*` описује број који се састоји од нула или више цифара. Симболи `^` и `$` означавају почетак и крај ниске, чиме се прецизира да цела вредност мора одговарати наведеном формату.

HTML 5 омогућава и специјализоване типове поља за унос података. За унос бројева користи се `input` поље типа `number`, уз могућност дефинисања минималне и максималне вредности атрибутима `min` и `max`. За унос електронске поште користи се тип `email`, док су за бројеве телефона и лозинке предвиђени типови `tel` и `password`.

HTML 5 валидација се често комбинује са CSS псеудокласама `:valid` и `:invalid`, које омогућавају визуелно истицање исправних и неисправних поља у формулару.

12.2 Клијентска валидација уз помоћ JavaScript-а

Иако HTML 5 нуди једноставан начин за основну валидацију, у случајевима када су услови коректности сложенији, неопходно је користити програмски језик JavaScript. JavaScript омогућава дефинисање произвољних правила провере и њихово прилагођавање конкретним потребама апликације.

Скрипт за валидацију се обично везује за догађај `submit` формулара. Унутар функције која обрађује тај догађај врши се провера да ли су поља попуњена и да ли унети подаци одговарају очекиваном формату. Уколико се утврди грешка, подразумевано понашање слања формулара се поништава, чиме се спречава слање некоректних података серверу.

12.3 Серверска валидација

Без обзира на то што је валидација извршена на страни клијента, сервер не сме да се ослони на ту проверу. Клијентску валидацију је могуће једноставно заобићи, на пример ручним слањем HTTP захтева. Због тога је серверска валидација увек неопходна.

Након пријема података из POST захтева, серверска апликација проверава да ли сва очекивана поља постоје и да ли садрже непразне и коректно форматиране вредности. Провере се најчешће односе на празна поља, формат мејл адресе или коректност бројева телефона.

У случају да се утврди грешка, сервер не врши упис података у базу, већ кориснику враћа одговарајућу поруку о грешци. Оваквим приступом обезбеђује се интегритет података и спречава унос неисправних информација у систем.

13 AJAX и асинхрона комуникација са сервером

До сада су веб-апликације у овом раду биле организоване на традиционалан начин: након пријема захтева, серверска апликација формира комплетну HTML страну и шаље је клијенту, који затим ту страну приказује у прегледачу веба. Том приликом се садржај тренутно отворене странице у потпуности замењује новом страницом.

Савремене веб-апликације често користе другачији приступ, у коме се део података преузима са сервера **без поновног учитавања целе странице**. Овакав начин рада остварује се помоћу технологије **AJAX** (енг. *Asynchronous JavaScript and XML*). Иако назив упућује на XML формат, у пракси се данас најчешће користи формат **JSON** (JavaScript Object Notation).

Основна идеја AJAX-а је да JavaScript код, који се извршава у прегледачу, на основу неке корисничке акције (нпр. уноса текста) шаље HTTP захтев серверу, прима податке и динамички их приказује унутар већ отворене странице.

Пример: динамичка претрага извођача

У овом примеру приказана је веб-апликација која омогућава претрагу извођача из базе података. Корисник уноси текст у поље за претрагу, а испод тог поља се приказује листа првих десет извођача чије име почиње унетим карактерима.

Апликација има две путање:

- корену путању `/`, која приказује HTML страну са пољем за претрагу,
- путању `/artists`, која прима текст за претрагу као GET параметар и враћа податке у JSON формату.

Основна страна апликације

Путања `/` враћа HTML шаблон који садржи поље за унос текста и празну листу у којој ће се приказивати резултати претраге:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Artists</title>
  </head>
  <body>
    <input type="text" id="Name" />
    <ul id="Artists"></ul>
  </body>
</html>
```

Серверска обрада AJAX захтева

Серверска функција на путањи `/artists` прима GET параметар `name`, врши упит ка бази података и враћа резултат у JSON формату:

```
@app.route("/artists")
def artists():
    name = request.args.get("name")
    artists = query_db(
        "SELECT Name FROM Artist WHERE Name LIKE ?",
        (name + "%",)
    )
    return jsonify(artists)
```

За претрагу се користи SQL оператор `LIKE`, при чему се специјални знак `%` додаје вредности параметра. Параметризовани упит обезбеђује заштиту од SQL инјекција, а функција `jsonify` претвара резултат у JSON формат погодан за употребу у JavaScript-у.

JavaScript и слање AJAX захтева

JavaScript код реагује на промену садржаја у пољу за унос текста и, када је унето најмање два карактера, шаље AJAX захтев серверу:

```
var inputName = document.getElementById("Name");

inputName.addEventListener("keyup", function() {
    if (inputName.value.length >= 2) {
        var xhttp = new XMLHttpRequest();

        xhttp.onreadystatechange = function() {
            if (this.readyState == 4 && this.status == 200) {
                var artists = JSON.parse(xhttp.responseText);
                var ul = document.getElementById("Artists");
                ul.innerHTML = "";

                artists.forEach(artist => {
                    var li = document.createElement("li");
                    li.innerHTML = artist[0];
                    ul.append(li);
                });
            }
        };

        const params = new URLSearchParams({name: inputName.value});
        const url = "{{ url_for('artists') }}?" + params.toString();

        xhttp.open("GET", url);
        xhttp.send();
    }
});
```

Након пријема успешног одговора (HTTP статус 200), подаци се претварају из JSON формата у JavaScript структуре података, а затим се динамички приказују у HTML листи. На овај начин се садржај странице ажурира без њеног поновног учитавања.

Предности AJAX приступа

Коришћењем AJAX технологије постиже се:

- бржи и интерактивнији рад веб-апликације,
- мањи проток података између клијента и сервера,
- боље корисничко искуство, јер се страна не освежава у целости.

Због ових особина, AJAX представља један од основних механизама савремених веб-апликација и важан је део наставе веб програмирања.

14 Закључак

У овом семинарском раду приказани су основни елементи развоја веб-апликација на серверској и клијентској страни, са посебним освртом на њихову примену у настави рачунарства и информатике у средњој школи. Кроз рад су обрађене теме као што су HTTP захтеви, рад са формуларима, колачићима и сесијама, повезивање са базом података, као и основи асинхроне комуникације помоћу AJAX технологије.

Приликом избора примера и начина излагања водило се рачуна о томе да садржај буде прилагођен ученицима, односно да се нови појмови уводе постепено и кроз конкретне и разумљиве ситуације. На тај начин ученици могу да стекну јасну представу о томе како функционише веб-апликација, која је улога сервера, а која клијента, као и како се подаци размењују између њих. Посебно је значајно што ученици кроз практичне примере могу да повежу теоријска знања са реалним применама које су им већ познате из свакодневне употребе веба.

Flask оквир се у том контексту показује као веома погодан за наставу, јер омогућава једноставну имплементацију веб-апликација без сложене конфигурације. Ученици се могу усмерити на разумевање основних концепата веб програмирања, уместо на техничке детаље који би им у почетној фази могли представљати препреку. Поред тога, кроз рад са базама података и параметризованим упитима могу се увести и основни појмови безбедности, што је важан део савременог информатичког образовања.

Садржаји приказани у овом раду могу да послуже као добра основа за реализацију наставних часова или вежби, али и као полазна тачка за даље проширивање градива. У настави је могуће постепено уводити напредније концепте, као што су кориснички налози, управљање сесијама или сложеније клијентско-серверске интеракције. На тај начин ученици не стичу само техничка знања, већ развијају и логичко размишљање и разумевање начина на који функционишу савремене информационе технологије.

Literatura

[1] Petlja.org — *Веб програмирање за четврти разред специјализованих ИТ одељења*. Доступно на: https://petlja.org/sr-Latn-RS/kurs/18657/21/6558 (приступљено: јануар 2026).

[2] Veb programiranje — математички факултет, Универзитет у Београду. Опис курса, теме и препоручена литература. Доступно на: https://www.vebp.matf.bg.ac.rs/ (приступљено: јануар 2026).

[3] TutorialsPoint — *Flask Tutorial*. Dostupno na: https://www.tutorialspoint.com/flask/index.htm (приступљено: јануар 2026).

[4] Д. Вуковић, С. Матковић, М. Ђуришић, *Рачунарство и информатика за четврти разред гимназије*, Завод за уџбенике, Београд, 2021.