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

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.

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)
```

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.

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:

```html
<!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>
```

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`.

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:

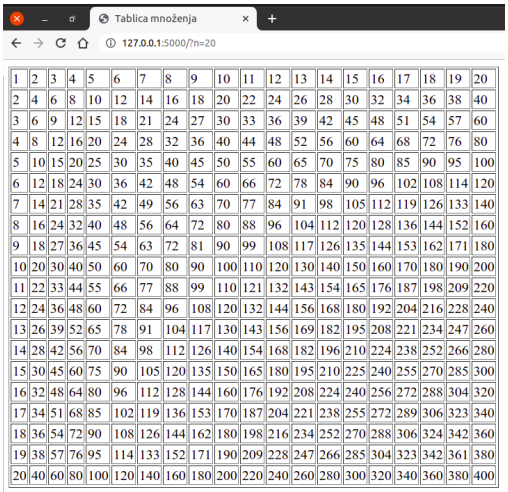
```python
from flask import Flask, render_template, request

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>
```

Метода GET

Метода GET подразумева да су подаци које клијент шаље серверу саставни део URL адресе. Подаци се наводе након упитника (`?`) у облику парова `naziv=vrednost`, који су раздвојени амперсандом (`&`).

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.

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.

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.

Пример: Калкулатор

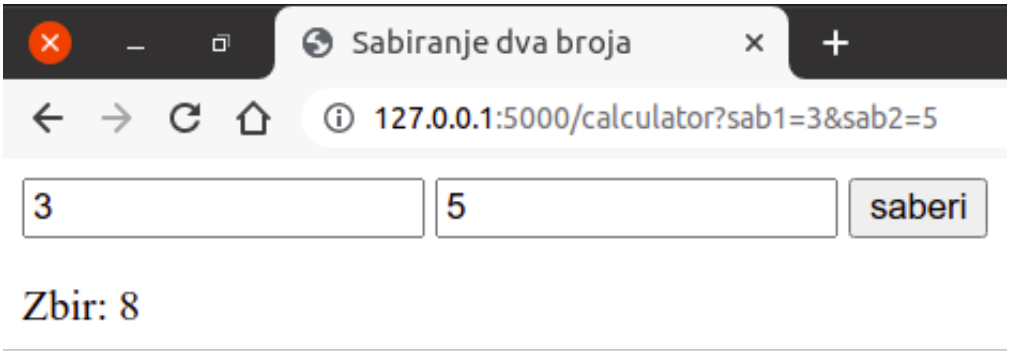
У овом примеру корисник уноси два броја у формулар, који се затим шаљу серверу методом GET. Сервер сабира унете вредности и враћа веб-страницу са приказом резултата.

Датотека `app.py` има следећи садржај:

```
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:

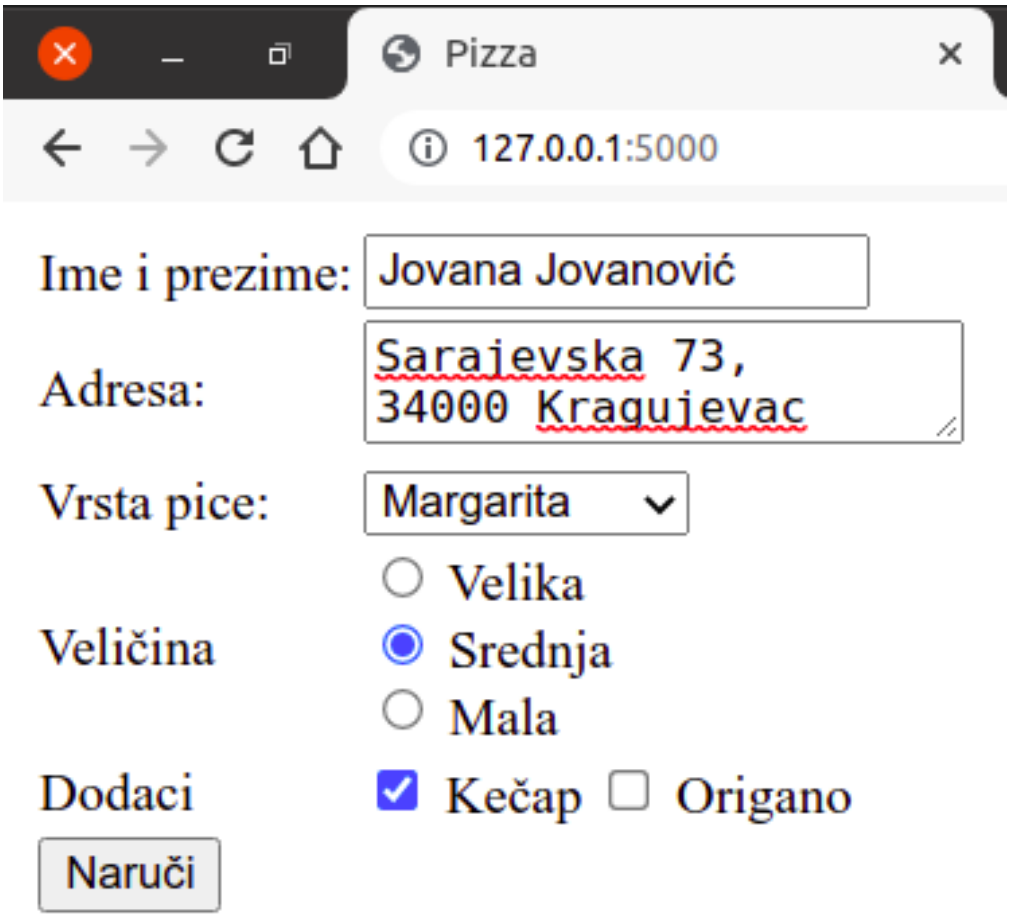
```html
<!DOCTYPE html>
<html>
   <head>
     <title>Sabiranje dva broja</title>
     <meta charset="utf-8" />
   </head>
   <body>
     <form>
       <input type="text" name="sab1" value="{{ sab1 }}" />
       <input type="text" name="sab2" value="{{ sab2 }}" />
       <input type="submit" value="saberi"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



Пример: Наручивање пице

Још један пример употребе GET захтева је формулар за наручивање пице. Корисник попуњава формулар, а унети подаци се шаљу серверу и приказују у облику наруџбенице.

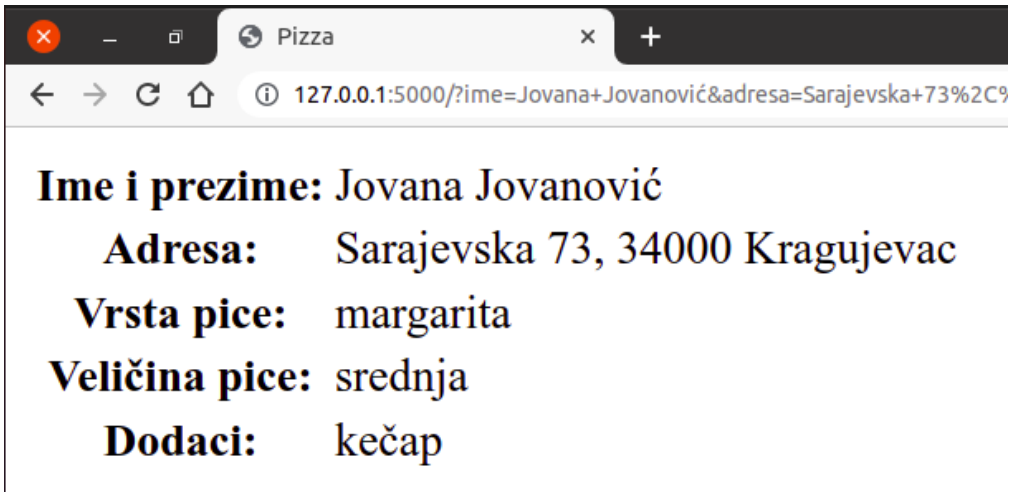
Изглед формулара:



Шаблон `templates/narudzbenica.html` има следећи садржај:

```html
<!DOCTYPE html>
<html>
  <head>
    <title>Pizza</title>
  </head>
  <body>
    <form>
      <table>
        <tr>
          <td><label for="ime">Ime i prezime: </label></td>
          <td><input type="text" id="ime" name="ime" /></td>
        </tr>
        <tr>
          <td><label for="adresa">Adresa: </label></td>
          <td><textarea id="adresa" name="adresa"></textarea></td>
        </tr>
        <tr>
          <td><label for="vrsta">Vrsta pice: </label></td>
          <td>
            <select id="vrsta" name="vrsta">
              <option value="kapricoza">Kapričoza</option>
              <option value="margarita">Margarita</option>
              <option value="vegetarijana">Vegetarijana</option>
            </select>
          </td>
        </tr>
        <tr>
          <td>Veličina</td>
          <td>
            <input type="radio" name="velicina" id="velika" value="velika"
                   checked />
            <label for="velika">Velika</label> <br/>
            <input type="radio" name="velicina" id="srednja" value="srednja" />
            <label for="srednja">Srednja</label> <br/>
            <input type="radio" name="velicina" id="mala" value="mala" />
            <label for="mala">Mala</label>
          </td>
        </tr>
        <tr>
          <td>Dodaci</td>
          <td>
            <input type="checkbox" name="kecap" value="kečap" id="kecap" />
            <label for="kecap">Kečap</label>
            <input type="checkbox" name="origano" value="origano" id="origano" />
            <label for="origano">Origano</label>
          </td>
        </tr>
        <tr>
          <td colspan="2">
            <input type="submit" name="naruci" value="Naručи" />
          </td>
        </tr>
      </table>
    </form>
  </body>
</html>
```

Сви подаци се примају у датотеци `app.py` путем GET параметара и прослеђују функцији `render_template`, која их табеларно приказује.



3 Метода POST

За разлику од методе GET, која користи садржај URL-а за слање података, метода POST шаље податке у **телу HTTP захтева**. То омогућава слање веће количине података, а подаци нису видљиви у URL-у, што је посебно важно за поверљиве информације, као

што су лозинке.

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.

Параметризоване руте у Flask-у

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

Пример једноставне Flask апликације:

```
1 from flask import Flask, render_template
2
3 app = Flask(__name__)
4
5 @app.route("/<ime>")
6 def pozdravi(ime):
7     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.

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 zove  {{ ime }}.</p>
  </body>
</html>
```

На слици је приказан изглед стране унутар прегледача веба.

Zdravo, ti se zoveš Ana.

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

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>")
```

```
2 def prikazi_proizvod(id_proizvod):
3     return f"ID proizvoda: {id_proizvod}"
```

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

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** (cookies) и **сесије** (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).

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.

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>
    {% if name %}
      <h1>Welcome {{ name }}</h1>
      <a href="{{ url_for('resetcookie') }}">Logout</a>
    {% else %}
      <form method="post" action="{{ url_for('setcookie') }}">
        <input type="text" name="name" />
        <input type="submit" value="login" />
      </form>
    {% endif %}
  </body>
</html>
```

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")
```

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.

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
```

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 њиховој управи.

Прикажимо пример Flask апликације која памти корисничко име у сесији. Корисник уноси своје име у формулар и, док је улогован, види своје име и опцију за излоговање.

Listing 1: Пример Flask апликације са сесијама

```
from flask import Flask, render_template, request, redirect, url_for,
    session

app = Flask(__name__)
app.secret_key = b'Lz\xbe\xaa\xd3\xce\xcd\xcb>\xdcLq\n%\x11h\x13\x9e\
   x1f\xf2\x1f\xcc\xddg'

@app.route("/")
def index():
    if "username" in session:
        username = session["username"]
        return render_template("index.html", username=username)
    else:
        return render_template("index.html")

@app.route("/login", methods=["POST", "GET"])
def login():
    if request.method == "POST":
        session["username"] = request.form["username"]
    return redirect(url_for("index"))

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

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

Listing 2: HTML шаблон за сесије

```
<!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>
    {% endif %}
  </body>
</html>
```

Кључ app.secret$_k eyjjjj.jPython$:

```
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.

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()
```

Приказаћемо називе првих 10 извођача. За то је потребно креирати објекат курсора:

```
cur = conn.cursor()
```

```
cur.execute("SELECT Name FROM artist LIMIT 10")
artists = cur.fetchall()
```

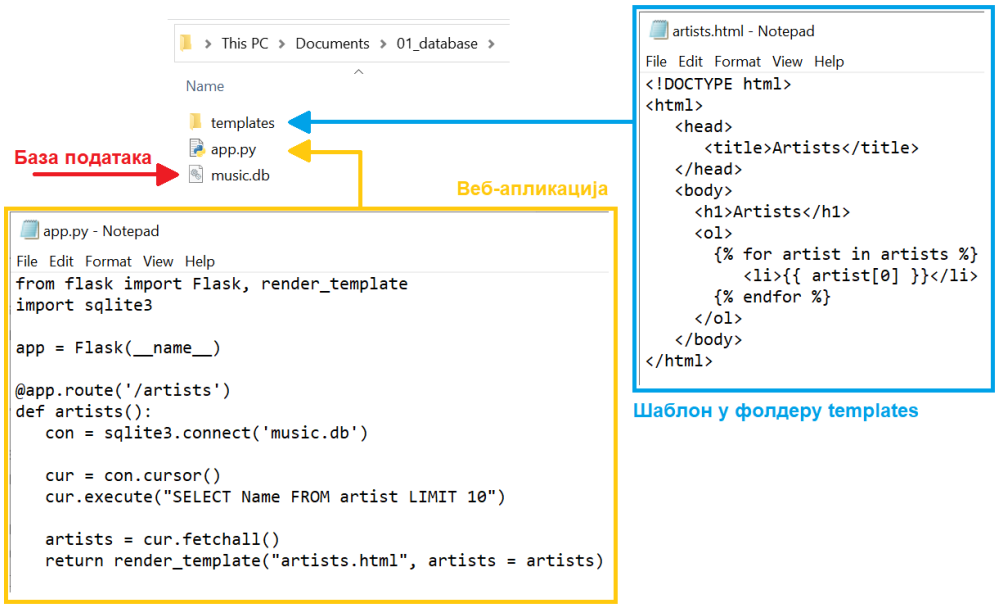
Шаблон templates/artists.html за приказ података може изгледати овако:

```
<!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>
```

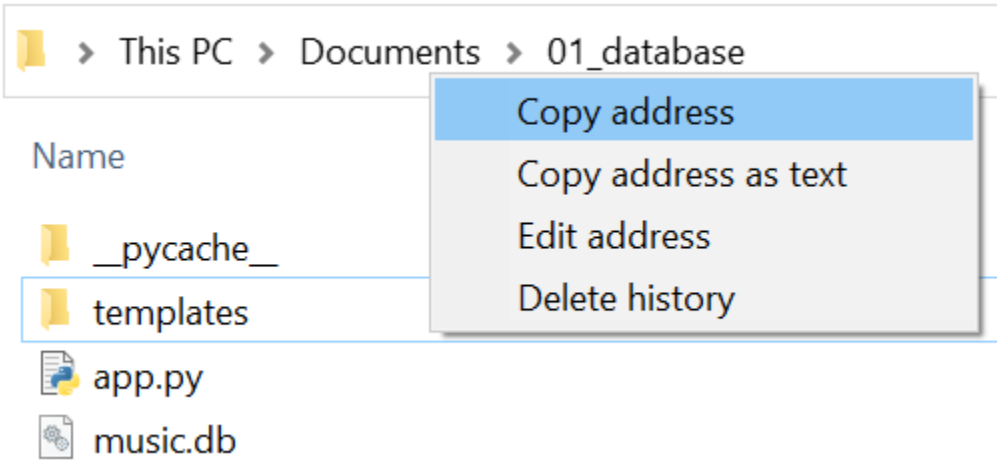
Комплетан пример app.py:

```
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:

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()
```

Курсор за упите добијамо као:

```
cur = get_db().cursor()
artists = cur.execute("SELECT Name FROM Artist LIMIT 10").fetchall()
```

Помоћна функција за упите

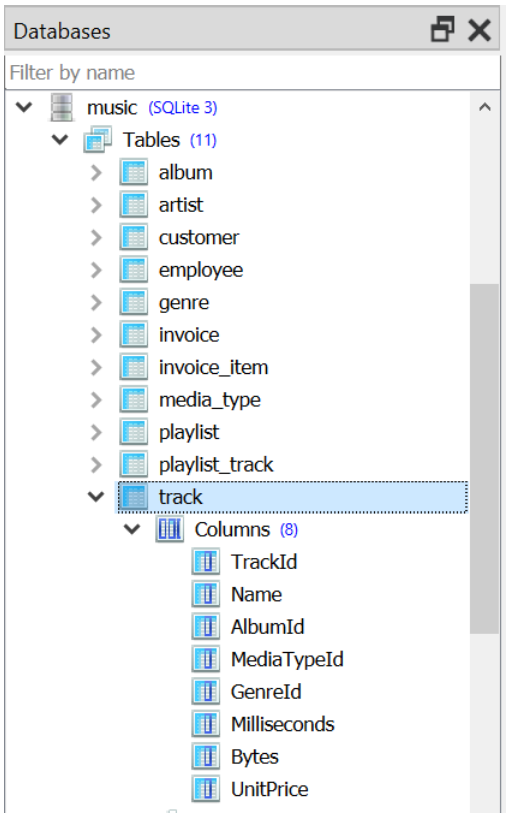
Функција query_db поједностављује рад са базом:

```
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.

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.



[a4paper,12pt]article [utf8]inputenc [T1]fontenc [serbian]babel listings xcolor

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://127.0.0.1:5

Python код за приказ песама по жанру

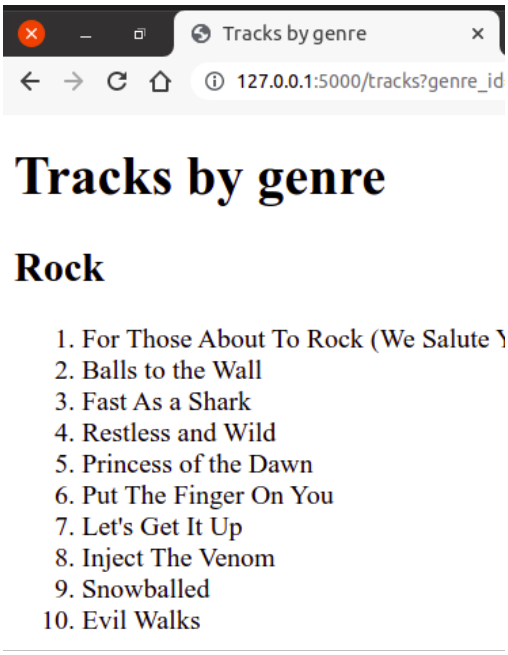
```
1 from flask import Flask, request, render_template
2 from your_db_module import query_db  # pretpostavimo da postoji ova
     funkcija
3
4 app = Flask(__name__)
5
6 @app.route("/tracks")
```

```
def tracks_by_genre():
    # proveravamo da li se me u prosle enim GET argumentima nalazi genre_id
    if "genre_id" in request.args:
        #  itamo  identifikator  anra  iz GET argumenata
        genre_id = request.args.get("genre_id")

        #  itamo  iz baze naziv  anra
        genre_name = query_db(
            "SELECT Name FROM Genre WHERE GenreId=?",
            (genre_id,),
            one=True  # vra a samo jedan red
        )

        # proveravamo da li je naziv  anra  uspe no pro itan
        if genre_name is None:
            return render_template(
                "tracks.html", error=True, error_msg="wrong genre id supplied"
            )

        #  itamo  iz baze spisak od najvi e 10 kompozicija tog anra
        tracks = query_db(
            "SELECT Name FROM Track WHERE GenreId=? LIMIT 10",
            (genre_id,)
        )

        # formiramo i vra amo veb-stranu
        return render_template(
            "tracks.html", genre_name=genre_name["Name"], tracks=tracks
        )
    else:
        # formiramo i vra amo veb-stranu koja ukazuje na gre ku
        return render_template(
            "tracks.html", error=True, error_msg="no genre id supplied"
```

```
        )

@app.route("/genres")
def genres():
    #   itamo   sve   anrove   iz baze
    genres = query_db("SELECT GenreId, Name FROM Genre")
    return render_template("genres.html", genres=genres)

if __name__ == "__main__":
    app.run(debug=True)
```

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 %}
```



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 %}
```

Заеднички 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

Јако често се параметри упита добијају на основу вредности које је корисник унео у формулар. На пример, у текућем задатку уместо листе линкова, одабир жанра можемо урадити коришћењем HTML формулара. Жанрове је могуће бирати из падајуће листе. Подсетимо се, падајућу листу у језику HTML можемо дефинисати на следећи начин:

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

Уз ову падајућу листу формулар треба да садржи и дугме submit чијим притиском проузрокујемо слање новог захтева. Назив (атрибут name) елемента select је genre$_id$, option

```
{% 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 %}
```

Апликација може имати само једну страницу која приказује и листу жанрова и одабране песме. У том случају, функција која обрађује захтев на путањи /tracks изгледа овако:

```
@app.route("/tracks")
def tracks_by_genre():
    # u itavamo sve  anrove  iz baze
    genres = query_db("SELECT GenreId, Name FROM genre")

    # proveravamo da li se me u GET argumentima nalazi genre_id
    if not "genre_id" in request.args:
        # prikazujemo samo formular za izbor  anra
        return render_template("index.html", genres=genres)

    #  itamo  identifikator  anra  koji je dat kao GET parametar
    genre_id = request.args.get("genre_id", type=int)
    #  itamo  iz baze naziv  anra
    genre_name = query_db("SELECT Name FROM Genre WHERE GenreId=?", (
        genre_id,), True)
    # prijavljujemo gre ku ako je identifikator  anra  pogre an
    if genre_name is None:
        return render_template("index.html", genres=genres,
                               error=True, error_msg="wrong genre id
                                  supplied")

    #  itamo  iz baze spisak od najvi e 10 kompozicija tog  anra
    tracks = query_db("SELECT Name FROM Track WHERE GenreId=? LIMIT
        10", (genre_id,))

    # formiramo i vra amo veb-stranu koja sadr i i formular i
        spisak pesama
    return render_template("index.html", genres=genres,
                           genre_id=genre_id, genre_name=genre_name["
                              Name"],
                           tracks=tracks)
```

Шаблон index.html приказује формулар и, ако је дат идентификатор жанра, списак песама или поруку о грешци:

```
<!DOCTYPE html>
```

```
<html>
   <head>
      <title>Tracks by genre</title>
   </head>
   <body>
     <h1>Tracks by genre</h1>

     <form>
       <select name="genre_id">
         {% for genre in genres: %}
         <option value="{{ genre.GenreId }}"
                 {% if genre_id == genre.GenreId %}selected{% endif
                    %}>
           {{ genre.Name }}
         </option>
         {% endfor %}
       </select>
       <input type="submit" value="Show" />
     </form>

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

Важно је напоменути да када се одабере жанр и притисне дугме submit, формулар се поново учитава са истим избором, што се постиже проверавањем вредности genre_id и додавањем атрибута selected у HTML-у.

10 Пренос параметара упита и избор жанра из листе

У веб-апликацијама често желимо да корисник бира параметре и на основу тога добије резултате из базе. У овом примеру приказујемо како се прослеђује идентификатор жанра и како се приказују композиције тог жанра. Такође, користимо падајућу листу за избор жанра.

10.1 Python код за Flask апликацију

```
from flask import Flask, request, render_template
import sqlite3

app = Flask(__name__)

# Funkcija za konekciju sa bazom
def get_db():
    conn = sqlite3.connect("your_database.db")
    conn.row_factory = sqlite3.Row
    return conn

# Funkcija za parametrizovane upite
def query_db(query, args=(), one=False):
    db = get_db()
    cur = db.execute(query, args)
    rv = cur.fetchall()
    cur.close()
    return (rv[0] if rv else None) if one else rv

# ------------------------
# 10.1 - Prikaz pesama po zanru
# ------------------------
@app.route("/tracks")
def tracks_by_genre():
    # Ucitavamo sve zanrove iz baze za formu
    genres = query_db("SELECT GenreId, Name FROM Genre")

    # Proveravamo da li je prosledjen parametar genre_id
    if "genre_id" not in request.args:
        # Ako nije, prikazujemo samo formu za izbor zanra
```

```
31          return render_template("index.html", genres=genres)
32
33      # Citamo prosledjeni identifikator zanra
34      genre_id = request.args.get("genre_id", type=int)
35
36      # Citamo naziv zanra iz baze
37      genre_name = query_db("SELECT Name FROM Genre WHERE GenreId=?", (
            genre_id,), one=True)
38
39      if genre_name is None:
40          # Pogresan ID  anra
41          return render_template("index.html", genres=genres,
42                                 error=True, error_msg="wrong genre id
                                       supplied")
43
44      # Citamo spisak do 10 kompozicija tog zanra
45      tracks = query_db("SELECT Name FROM Track WHERE GenreId=? LIMIT
            10", (genre_id,))
46
47      # Prikazujemo formu i listu pesama
48      return render_template("index.html", genres=genres,
49                             genre_id=genre_id, genre_name=genre_name["
                                   Name"],
50                             tracks=tracks)
51
52  # ------------------------
53  # 10.2 - Upis novog korisnika u bazu
54  # ------------------------
55  @app.route("/customer", methods=["GET", "POST"])
56  def customer():
57      if request.method == 'GET':
58          return render_template("customer.html")
59      else:
60          try:
61              db_conn = get_db()
62              cursor = db_conn.cursor()
63              query = ("INSERT INTO Customer "
64                       "(FirstName, LastName, Phone, Email) "
```

```
65                      "VALUES (?, ?, ?, ?)")
66
67              first_name = request.form['first-name']
68              last_name = request.form['last-name']
69              phone = request.form['phone']
70              email = request.form['e-mail']
71
72              cursor.execute(query, (first_name, last_name, phone,
                    email))
73              db_conn.commit()
74
75              return render_template("customer.html",
76                                     msg="Customer was successfully
                                        added")
77          except:
78              return render_template("customer.html",
79                                     msg="Error adding new customer")
```

11 „Флешоване“ поруке

Поруку о томе да ли је упис у базу успео смо у претходној апликацији вршили тако што смо увели посебан параметар msg који смо прослеђивали шаблону. Пошто је ситуација у којој кориснику треба да буду приказане неке једнократне поруке прилично честа, библиотека Flask нуди посебан механизам за то. У питању су такозване „флешоване“ поруке (енг. *flashed messages* - „флеш“ означава бљесак који се на тренутак појави и брзо нестане). Python функција која одговара на HTTP захтев може да постави нову поруку позивом функције flash. Са друге стране, унутар шаблона се листа свих порука може добити помоћу get_flashed_messages(). Свака порука је обична ниска карактера, при чему се порукама могу додатно придружити и категорије које корисник дефинише (попут "error", "ok" и слично), тако што се функцији flash проследе две ниске (прва је текст поруке, а друга је категорија). Тада је поруке унутар шаблона потребно прихватити позивом get_flashed_messages(with_categories=True), која онда уместо листе порука враћа листу уређених парова где је први елемент категорија, а други текст поруке. Када се користи наслеђивање шаблона, у главном костуру стране се може предвидети неки простор на ком ће се приказивати све флешоване поруке, без обзира на то из које функције оне долазе. На тај начин се шаблони

који приказују резултат рада појединачних функција могу ослободити потребе да се баве приказивањем једноструких порука корисницима. Прилагодимо претходну апликацију тако да користи флешоване поруке.

11.1 Костур HTML стране

Направићемо костур HTML странице у шаблону templates/index.html и на почетку тела ћемо предвидети приказивање свих флешованих порука:

```
<!DOCTYPE html>
<html>
   <head>
     <title>Add new customer</title>
     <style>
       .msg { border: 1px solid black; }
       .error { background-color: #fcc; }
       .ok  { background-color: #cfc; }
     </style>
   </head>
   <body>
     {% with messages = get_flashed_messages(with_categories=true) %}
     {% if messages %}
        <h1>Info</h1>
        {% for category, message in messages %}
        <div class="{{ category }} msg">{{ message }}</div>
        {% endfor %}
     {% endif %}
     {% endwith %}

     {% block content %}
     {% endblock %}
   </body>
</html>
```

Приметимо да смо употребили наредбу:

```
{% with promenljiva = funkcija(...) %}
   ...
{% endwith %}
```

Њом постижемо да се вредност функције складишти у наведену променљиву и користи унутар тела те наредбе (у нашем примеру смо на тај начин формирали променљиву која садржи листу свих флешованих порука).

11.2 Шаблон за формулар за унос података

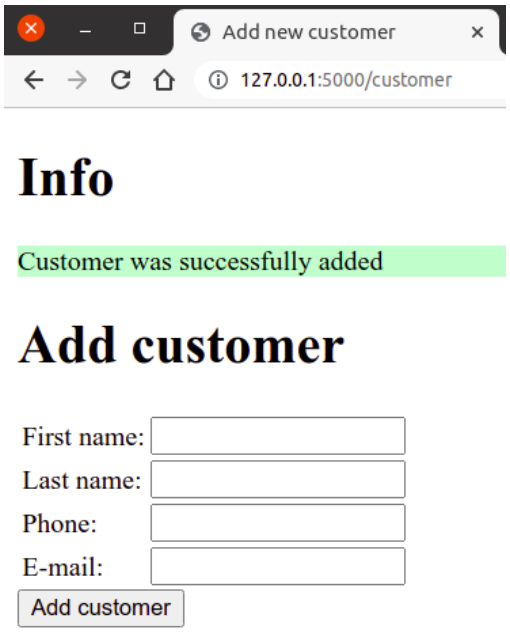
У датотеци templates/customer.html дефинишемо централни блок у ком се само приказује формулар за унос података:

```
{% extends "index.html" %}
{% block content %}
     <h1>Add customer</h1>

     <form method="POST">
       <table>
         <tr>
           <td><label for="first-name">First name:</label></td>
           <td><input type="text" id="first-name" name="first-name" /
              ></td>
         </tr>
         <tr>
           <td><label for="last-name">Last name:</label></td>
           <td><input type="text" id="last-name" name="last-name" /><
              /td>
         </tr>
         <tr>
           <td><label for="phone">Phone:</label></td>
           <td><input type="text" id="phone" name="phone" /></td>
         </tr>
         <tr>
           <td><label for="e-mail">E-mail:</label></td>
           <td><input type="text" id="e-mail" name="e-mail" /></td>
         </tr>
       </table>
       <input type="submit" value="Add customer" />
     </form>
{% endblock %}
```

11.3 Python апликација са флешованим порукама

На крају, у самој апликацији флешоване поруке постављамо коришћењем функције flash (наравно, она мора бити увезена у склопу директиве import):

```
@app.route("/customer", methods=["GET", "POST"])
def customer():
    if request.method == 'GET':
        return render_template("customer.html")
    else:
        try:
            db_conn = get_db()
            cursor = db_conn.cursor()
            query = "INSERT INTO Customer ('FirstName', 'LastName', '
               Phone', 'Email') VALUES (?, ?, ?, ?)"
            first_name = request.form['first-name']
            last_name = request.form['last-name']
            phone = request.form['phone']
            email = request.form['e-mail']
            cursor.execute(query, (first_name, last_name, phone,
               email))
            db_conn.commit()
            flash("Customer was successfully added", "ok")
            return render_template("customer.html")
        except:
            flash("Error adding new customer", "error")
            return render_template("customer.html")
```



12 Валидација података унетих у формулар

У претходној верзији апликације корисник је у формулар могао да упише било какве податке (па чак и да формулар остави празним) и ти подаци би завршили у бази података. Наравно, ово је веома лоша пракса и квалитетне веб-апликације проверавају елементарну коректност унетих података пре него што их упишу у базу. Поступак провере података унетих у формулар назива се валидација формулара.

12.1 Клијентска валидација уз помоћ HTML 5

HTML5 омогућава да се правила коректности формулара задају директно у формулару. На пример:

- Атрибут required осигурава да поље није празно.
- Атрибут pattern одређује регуларни израз који ниска мора да задовољи.
- Атрибути minlength и maxlength ограничавају дужину унетог текста.
- За бројеве се користи <input type="number"min="...», за мејл <input type="email», за телефон <input type="tel».

Пример HTML формулара са валидацијом:

```
<!DOCTYPE html>
<html>
   <head>
     <title>Add new customer</title>
     <style>
       .msg { border: 1px solid black; }
       input:invalid {
           background-color: #f55;
       }
     </style>
   </head>
   <body>
     <h1>Add customer</h1>
     <form method="POST" id="customerForm">
       <table>
         <tr>
           <td><label for="first-name">First name:</label></td>
           <td><input type="text" id="first-name" name="first-name"
              required /></td>
         </tr>
         <tr>
           <td><label for="last-name">Last name:</label></td>
           <td><input type="text" id="last-name" name="last-name"
              required /></td>
         </tr>
         <tr>
           <td><label for="phone">Phone:</label></td>
           <td><input type="tel" id="phone" name="phone" required /><
              /td>
         </tr>
         <tr>
           <td><label for="e-mail">E-mail:</label></td>
           <td><input type="email" id="e-mail" name="e-mail" required
               /></td>
         </tr>
       </table>
       <input type="submit" value="Add customer" />
     </form>
```

```
    </body>
</html>
```

12.2 Клијентска валидација уз помоћ JavaScript-а

```
<script>
document.forms.customerForm.addEventListener("submit", validateForm);

function validateForm(event) {
    var form = document.forms.customerForm;
    var firstName = form.elements["first-name"];
    var lastName = form.elements["last-name"];
    var phone = form.elements["phone"];
    var eMail = form.elements["e-mail"];

    function validateNonEmpty(field) {
        field.classList.remove("invalid");
        if (field.value.length == 0) {
            field.classList.add("invalid");
            return false;
        }
        return true;
    }

    function validateEMail(field) {
        field.classList.remove("invalid");
        const re = /^[a-z0-9+_.-]+@[a-z0-9.-]+$/;
        if (!re.test(String(field.value).toLowerCase())) {
            field.classList.add("invalid");
            return false;
        }
        return true;
    }

    var OK = true;
    OK = OK && validateNonEmpty(firstName);
    OK = OK && validateNonEmpty(lastName);
    OK = OK && validateNonEmpty(phone);
```

```
    OK = OK && validateNonEmpty(eMail) && validateEMail(eMail);

    if (!OK) event.preventDefault();
}
</script>
```

12.3 Серверска валидација у Python-у (Flask)

```
@app.route("/customer", methods=["GET", "POST"])
def customer():
    if request.method == 'GET':
        return render_template("customer.html")
    else:
        first_name = request.form['first-name']
        last_name = request.form['last-name']
        phone = request.form['phone']
        email = request.form['e-mail']

        error = None
        if not first_name or not first_name.strip():
            error = "First name is missing"
        elif not last_name or not last_name.strip():
            error = "Last name is missing"
        elif not phone or not phone.strip():
            error = "Phone is missing"
        elif not email or not re.match(r"[a-z0-9+_\.-]+@[a-z0-9\.-]+"
           , email):
            error = "E-mail is not valid"

        if error:
            return render_template("customer.html", msg=error)

        try:
            db_conn = get_db()
            cursor = db_conn.cursor()
            query = ("INSERT INTO Customer "
                     "(FirstName, LastName, Phone, Email) "
                     "VALUES (?, ?, ?, ?)")
```

```
30              cursor.execute(query, (first_name, last_name, phone,
                    email))
31              db_conn.commit()
32              return render_template("customer.html",
33                                     msg="Customer was successfully
                                        added")
34          except:
35              return render_template("customer.html",
36                                     msg="Error adding new customer")
```

На овај начин је обезбеђена и клијентска и серверска валидација података, чиме се спречава упис некоректних података у базу.

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 (преузето: 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/ (преузето: јануар 2026).