Flask

prosty tutorial

Visual Studio Code

- https://code.visualstudio.com/
- rozszerzenie Python (Microsoft)

1. venv

środowisko wirtualne

```
$ mkdir myproject
$ cd myproject
$ python3 -m venv .venv
```

Aktywacja

```
$ . .venv/bin/activate
```

instalacja Flask

w środowisku wirtualnym (info po prawej stronie w terminalu)

```
(.venv) $ pip install Flask
```

prosta apka
hello.py

```
from flask import Flask

app = Flask(__name__)


@app.route('/')

def say_hello():

    return "<p>Hello!</p>"
```

hello.py

Uruchamiamy aplikację

$ flask --app hello run

lub

$ python -m flask --app hello run

w przeglądarce ---> http://127.0.0.1:5000

prosta apka
hello.py

```
from flask import Flask

app = Flask(__name__)


@app.route('/')

def say_hello():

    return f"<p>Hello {__name__}!</p>"
```

hello.py

Uruchamiamy aplikację

$ flask --app hello run --debug

lub

$ python -m flask --app hello run --debug

Routing (Trasowanie)

routing - metoda znajdowania ścieżek od źródeł do miejsc docelowych w sieci, wzdłuż której można przekazywać informacje

```
@app.route('/')
def index():
    return 'Index Page'


@app.route('/hello')
def hello():
    return 'Hello, World'
```

```
@app.route('/projects/')
def projects():
    return 'The project page'

@app.route('/about')
def about():
    return 'The about page'
```

Trasowanie dynamiczne

```
from markupsafe import escape

@app.route('/user/<username>')
def show_user_profile(username):
    return f'User {escape(username)}'

@app.route('/post/<int:post_id>')
def show_post(post_id):
    return f'Post {post_id}'

@app.route('/path/<path:subpath>')
def show_subpath(subpath):
    return f'Subpath {escape(subpath)}'
```

str - (domyślnie) akceptuje dowolny tekst oprócz ukośników

int - akceptuje dodatnie liczby całkowite

float - akceptuje dodatnie wartości zmiennoprzecinkowe

path - jak str, ale akceptuje również ukośniki

uuid - akceptuje ciągi znaków UUID

escape

zmienia znaczenie znaków tak, by tekst był bezpieczny w HTML / XML. Znaki, które mają specjalne znaczenie (np: <script>), są zastępowane przez rzeczywiste znaki. Ogranicza to ataki polegające na wstrzykiwaniu (injection attack), co oznacza, że niezaufane dane wprowadzane przez użytkownika mogą być bezpiecznie wyświetlane na stronie.

Funkcja **escape()** pomija tekst i zwraca *obiekt* złożony ze znaczników. Obiekt nie będzie już podlegał zmianie znaczenia, ale każdy tekst będzie po prostu tekstem.

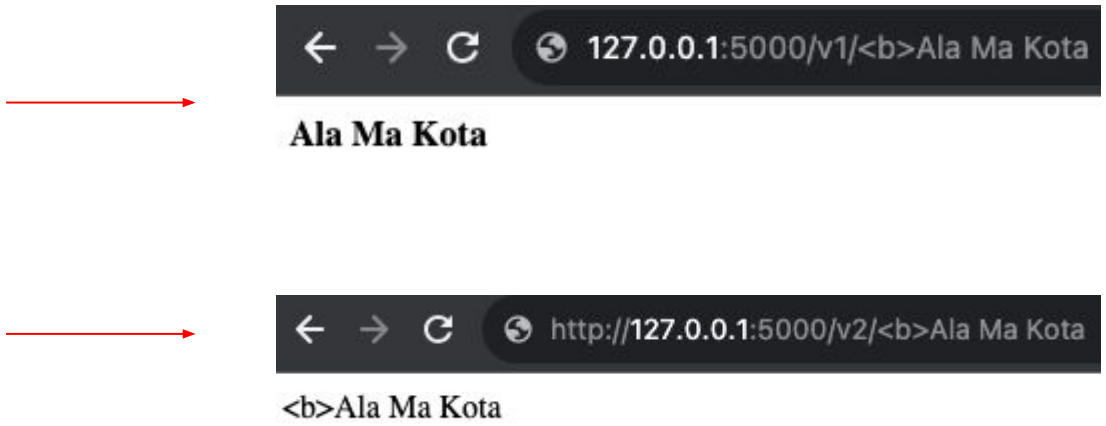
https://markupsafe.palletsprojects.com/

escape

przykłady

```
@app.route('/v1/<var>')
def show_user_profile1(var):
    return f'{var}'

@app.route('/v2/<var>')
def show_user_profile2(var):
    return f'{escape(var)}'
```



URL

aby zbudować URL dla konkretnego zasobu, musimy użyć funkcji url_for

```
@app.route('/login')
def login():
    msg = 'login'
    return msg

@app.route('/user/<username>')
def profile(username):
    return f'{username}\'s profile'


print(url_for('login'))
print(url_for('profile', username='John Doe'))
```

/login
/user/John%20Doe

Zadanie 1

Stwórz aplikację cwt (CodeWithTeam) która będzie miała 3 strony

- Startową - z prostym przywitaniem
- About - krótki opis projektu (2-3 zdania)
- Team - opis zespołu rozwijającego CWT

dodatkowo

- użyj tagów HTML (p, br…)
- dodaj nawigację pomiędzy stronami (url_for) - w tabeli lub w div-ach

pliki statyczne

dodajemy je w katalogu **static**, który tworzymy w głównym katalogu aplikacji; dobry przykład to pliki css, które będziemy dodawać do naszych stron

```
| app
      | static
            | css
                  | main.css
      | cwt.py
```

Zadanie 1A

- dodaj main.css do wszystkich stron
- zmień tabelkę nawigacyjną tak, by wpisy były UPPERCASE
- do tytułów paragrafów w sekcji About dodaj klasę “heading”; klasę tą opisz w main.css jako bold i zwiększ font o 2 punkty

szablony html - Jinja2

- już jest we Flask-u
- meta-programowanie
- render_template
- bazuje na katalogu templates

```
| app
    | cwt.py
    | static
        | css
            | main.css
    | templates
        | start.html
        | about.html
        | maciej.html
```

https://pl.wikipedia.org/wiki/Jinja2
https://jinja.palletsprojects.com/en/3.1.x/

Jinja2

Szablon zawiera zmienne i wyrażenia, które są zastępowane wartościami podczas renderowania szablonu; zawiera też znaczniki kontrolujące logikę szablonu.

- {% ... %} dla instrukcji: for, while…
- {{ ... }} dla wyrażeń: x, y + 3, url_for(‘start’)...
- {# ... #} dla komentarzy

Zadanie 2

Zmień plik aplikacji cwt.py tak by renderował poszczególne pliki HTML w katalogu templates.

base.html

szablon główny aplikacji

<head>
    zawsze identyczny z pominięciem tytułu
    {% block title %} {% endblock %}
<body>
    zawsze inne (poza nawigacją, nagłówkiem i stopką)
    {% block content %} {% endblock %}

<script>
    można dodać obsługę jakiegoś frameworku JS

base.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" href={{ url_for('static', filename='css/mainA.css') }}>
    <title>{% block title %} {% endblock %}</title>
</head>
<body>
<table>
    <tr>
        <td><a href={{ url_for('start') }}>start</a></td>
        <td><a href={{ url_for('about') }}>about</a></td>
        <td><a href={{ url_for('maciej') }}>maciej</a></td>
    </tr>
</table>
{% block content %} {% endblock %}
</body>
</html>
```

np: start.html

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

{% block title %}
Code With Team
{% endblock %}

{% block content %}
<h1>Witaj na stronie projektu Code With Team</h1>
<p>Zapraszamy!</p>
{% endblock %}
```

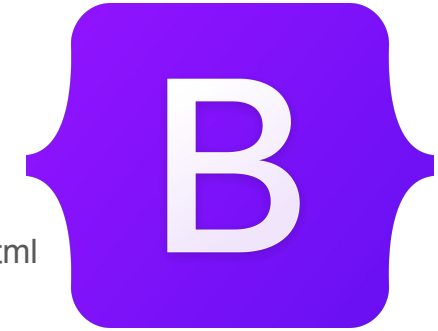
Zadanie 3

1. Wykorzystaj **base.html** do stworzenia struktury aplikacji.
2. Dodaj stronę **zadania.html**
   ma to być prosta strona z tabelą która ma 3 kolumny nagłówka (th)
   ZADANIE | DATA | AKCJA
   (pod nagłówkiem może być jedna pusta linia, tak by zobaczyć czy OK)
   AKCJE: Usuń oraz Zmień

   ZADANIE | DATA | AKCJA

   | | Usuń
   Zmień

bootstrap



aby nie wymyślać css, najlepiej posłużyć się jakimś znanym frameworkiem - dodać do base.html

- do <head> dodać

```
<!-- Latest compiled and minified CSS →
<link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap@3.4.1/dist/css/bootstrap.min.css"
integrity="sha384-HSMxcRTRxnN+Bdg0JdbxYKrThecOKuH5zCYotlSAcp1+c8xmyTe9GYg1l9a69psu"
crossorigin="anonymous">
```

- do body (na koniec)

```
<!-- Latest compiled and minified JavaScript →
<script src="https://cdn.jsdelivr.net/npm/bootstrap@3.4.1/dist/js/bootstrap.min.js"
integrity="sha384-aJ21OjlMXNL5UyIl/XNwTMqvzeRMZH2w8c5cRVpzpU8Y5bApTppSuUkhZXN0VxHd"
crossorigin="anonymous"></script>
```

https://getbootstrap.com/

bootstrap - tabele

- dodaj do tabeli
  <table class=”table”>

- jest wiele modyfikacji ad-hoc

```
<table class="table table-hover">
<table class="table table-hover table-dark">
<table class="table table-sm">
...
```

https://getbootstrap.com/docs/4.0/content/tables/

zadania

potrzebujemy

1. formularza - przechwytywanie zadań do tabelki (mamy)
2. bazy danych - do przechowywania zadań
   jest opór metod by taką DB stworzyć

formularz

ULR DO KTÓREGO PRZEJDZIEMY

```
<form class="form-inline" action="zadania" method="POST">
        <div class="form-group">
            <label for="zadanie">Zadanie</label>
            <input type="text" class="form-control" name="zadanie" id="zadanie"
placeholder="Nowe zadanie">
        </div>
        <button type="submit" class="btn btn-default">Dodaj</button>
    </form>
```

nazwa klucza

wszystkie “class” to odwołania do bootstrap

db: sqlite3

1. tworzymy strukturę DB: **schema.sql**

```
DROP TABLE IF EXISTS zadania;

CREATE TABLE zadania (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    czas TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    zadanie TEXT NOT NULL
);
```

tutaj będziemy zapisywać dane z formularza

db: sqlite3

2. budujemy plik PY by stworzyć DB

```
import sqlite3
connection = sqlite3.connect('database.db')

with open('schema.sql') as f:
        connection.executescript(f.read())

cur = connection.cursor()

# jeśli chcemy dodać coś od razy, by DB nie była pusta
# cur.execute("INSERT INTO zadania (zadanie) VALUES (?)", ('Wyrzuc smieci',))
# cur.execute("INSERT INTO zadania (zadanie) VALUES (?)", ('Ścisz TV',))

connection.commit()
connection.close()
```

zadania

```
def get_db_connection():
    conn = sqlite3.connect('database.db')
    conn.row_factory = sqlite3.Row
    return conn

@app.route('/zadania', methods=['POST', 'GET'])
def zadania():
    if request.method == 'POST':
            zrób coś jak ktoś użyje formularza
    else:
            po prostu wyświetl zadania
```

request.method = ‘POST’

```
if request.method == 'POST':
        zadanie = request.form['zadanie']
        try:
                conn = get_db_connection()
                conn.execute('INSERT INTO zadania (zadanie) VALUES (?)', (zadanie, ))
                conn.commit()
                conn.close()
                return redirect(url_for('zadania'))

         except:
                return 'DB error'
```

nie przekazujemy nic - po prostu wyświetlamy stronę

```
else:
    conn = get_db_connection()
    zadania = conn.execute('SELECT * FROM zadania').fetchall()
    conn.close()
    return render_template('zadania3.html', zadania=zadania)
```


przekazywanie zmiennej do szablonu

trzeba przebudować szablon, by pokazywał **zadania**

zadania.html

```
<tbody>
        {% for zad in zadania %}
        <tr>
                <td>{{ zad['zadanie'] }}</td>
                <td>{{ zad.czas }}</td>
                <td>
                        <a href="#">Usuń</a>
                        <br />
                        <a href="#">Zmień</a>
                </td>
        </tr>
        {% endfor %}
</tbody>
```

Zadanie 4

1. przebuduj DB tak, by miała jeszcze jedno pole **priorytet**
   priorytet przyjmuje 3 wartości: wysoki, normalny, niski
2. przebuduj zadania.html by obsługiwały wszystkie kolumny z DB (id, zadanie, data, priorytet + akcja)
3. w formularzu dodaj pole **priorytet**
4. *w przypadku gdy user poda inny **priorytet**, poinformuj go o 3 możliwych i nie update-uj DB

usuń zadanie z listy

1. dodać route do /delete/id (id jest INT)
   funkcja del_zadanie(id) tym razem będzie miała argument (id)

2. wywołać regułę SQL
   ```
   conn.execute('DELETE FROM zadania WHERE id = ?', (id, ))
   ```

3. wrócić returnem do zadań
   ```
   return redirect(url_for('zadania'))
   ```

Zadanie 5: zmień wpis w DB dla zadania

1. możemy zmienić
   a. zadanie (nazwę)
   b. priotytet

2. potrzebujemy formularza w *update.html*

3. start podobnie jak **usuń**, komenda SQL:

```
conn.execute('UPDATE zadania SET zadanie = ?, priorytet = ? WHERE id = ?',
             (zadanie, priorytet, id))
```

4. co zrobić, gdy user zmieni tylko jedno pole?

Zadanie 6

Jeżeli nie ma zadań, powinniśmy widzieć informację, że ich nie ma, a nie pustą tabelę…

```
{% if zadania|length < 1 %}
    TUTAJ INFO HTML ŻE NIE MA ZADAŃ
{% else %}
    TABELA ZADAŃ
{% endif %}
```

login / sesja

przemyśleć

- strukturę DB
  relację między tabelami USER i ZADANIA
- cookies
- admin?

login / sesja: DB

- stworzymy dodatkową tabelę USER, zawierającą id oraz nazwę użytkownika (bez hasła) - zalogować do systemu będzie się można, gdy ktoś poda nazwę użytkownika z tej tabeli
- w tabeli ZADANIA dodamy pole user, tak by zadania mogły być przypisane do usera i tylko on mógł je sobie dodawać i zmieniać
  *przemyśleć rolę admina

Zadanie 7

Stwórz takie pliki *schema.sql* i *init_db.py*, by odpowiadały powyższym założeniom. Dostosuj template zadania i plik aplikacji (zadania:POST). Zastosuj zmiany

zakładka login

1. template login.html (form)
2. sprawdzanie czy uname jest w tabeli user: jest -> OK, nie -> błąd
3. przechwytywanie zalogowanego usera do sesji (pliki cookies)
4. wyświetlanie login / logout jeżeli sesja nie ma / ma aktywnego usera

dodatkowo

- routig *login* w cwt.py
- update base.html

sesja (cookies)

dowolny STR lub render strony html

dodawanie wpisu w cookies

```
from flask import make_response

resp = make_response(‘zalogowano’)
resp.set_cookie('uname', user_name)
return resp
```

usuwanie wpisu w cookies

```
from flask import make_response

resp = make_response(‘wylogowano’)
resp.set_cookie('uname', ‘’)
return resp
```

Zadanie 8

Zmienić tak update zadań, by dodawało zadanie dla zalogowanego użytkownika. Jeżeli nie ma takiego (uname == ‘’), to usunąć możliwość dodawania zadania z informacją o potrzebie zalogowania.

Co dalej?

1. dodać możliwość tworzenia użytkowników
2. dodać obsługę haseł (reguły haseł, szyfrowanie, Flask-login)
3. scalić tworzenie db w głównym pliku aplikacji (komenda click)
   https://flask.palletsprojects.com/en/2.3.x/tutorial/database/
4. obsługa db przez Flask-SQLAlchemy
5.

użyteczne dodatki

- Flask-Login: zarządza sesją użytkownika i obsługuje logowanie i wylogowanie oraz zapamiętywanie zalogowanych użytkowników.
  https://flask-login.readthedocs.io/en/latest/

- Flask-SQLAlchemy: upraszcza korzystanie z Flask z SQLAlchemy, zestawem narzędzi Python SQL i Object Relational Mapper do interakcji z bazami danych SQL.
  https://flask-sqlalchemy.palletsprojects.com/en/3.0.x/

- Flask-Mail: pomaga w wysyłaniu wiadomości e-mail w aplikacji Flask.
  https://pythonhosted.org/Flask-Mail/

read

- https://flask.palletsprojects.com/en/2.3.x/
- https://www.geeksforgeeks.org/flask-tutorial/
- https://realpython.com/tutorials/flask/
- https://www.digitalocean.com/community/tutorials/how-to-make-a-web-application-using-flask-in-python-3
- https://www.tutorialspoint.com/flask/index.htm

- https://getbootstrap.com
- https://jinja.palletsprojects.com
- https://www.sqlalchemy.org