

Flask

Partea I. Bazele Flask

Ce ne așteaptă?

1. Mediul virtual
2. Prima aplicație Flask
3. Bazele rutării
4. Rutarea dinamică
5. Debug Mode

1. Mediul virtual

De ce mediu virtual?

- Imaginați-vă situația în care creați o aplicație web ce utilizează unele biblioteci Python externe
- Când una din biblioteci este actualizată la o versiune mai nouă, se introduc noi caracteristici, dar se fac și unele modificări prin ștergerea unor caracteristici existente.
- Ce se întâmplă cu aplicația web elaborată?
- Din fericire, putem utiliza un mediu virtual care ne permite controlul dependențelor!
- Fiecare proiect asupra căruia se lucrează are propriul său mediu virtual
- Acesta este motivul utilizării mediilor virtuale.

Crearea mediului

- Anaconda vine cu “manager” încorporat pentru controlul mediului virtual.
- Este necesară parcurgerea unor pași în linia de comandă (command line) pentru a crea un mediu
- Se poziționează linia de comandă în folderul unde se va crea mediu
- Comanda pentru crearea mediului :

```
conda create --name nume_mediu
```
- `nume_mediu` - numele mediului virtual atribuit de creator.
- Conda te întreabă dacă continui (Proceed (y/n)?). Se selectează y pentru a continua.

Crearea mediului virtual personalizat

- Crearea mediului cu specificarea unei pachet și versiunii acestuia:
`conda create --n nume_mediu nume_pachet=numar_versiune`
- Crearea mediului cu specificarea versiunii Python:
`conda create --name nume_mediu python=3.9`
- Listarea tuturor :
`conda env list`
- Activarea unui mediu:
`activate nume_mediu`
- Setarea mediului de dezvoltare - va permite actualizarea automată a fișierului pe server:
`set FLASK_ENV=development`
- Dezactivarea unui mediu:
`deactivate nume_mediu`

Instalarea pachetelor în mediu

- Se activează mediul
- Se poate instala un pachet cu comanda:

```
pip install nume_pachet
```

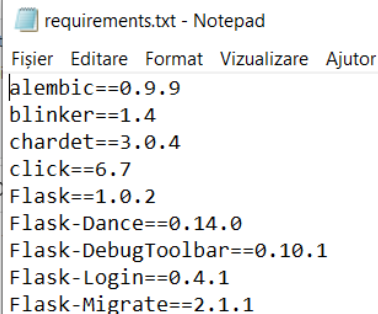
- Se pot instala mai multe pachete:

```
pip install nume_pachet_1 nume_pachet_2 ...
```

- Se pot instala pachete cu numele și versiunile acestora incluse într-un fișier txt (de exemplu requirements.txt):

```
pip install -r requirements.txt
```

- Exemplu de structura requirements.txt :



```
requirements.txt - Notepad  
Fișier Editare Format Vizualizare Ajutor  
alembic==0.9.9  
blinker==1.4  
chardet==3.0.4  
click==6.7  
Flask==1.0.2  
Flask-Dance==0.14.0  
Flask-DebugToolbar==0.10.1  
Flask-Login==0.4.1  
Flask-Migrate==2.1.1
```

2. Prima aplicație Flask

Esența primei aplicații

- Se va crea un site web cu utilizarea Flask
- Acesta este cel mai simplu site posibil
- Acesta va returna Salutarea tuturor când se va apela pagina web
- În folderul de proiect se creează un fișier python cu extensia .py

Scriptul primei aplicații

- Se importă clasa Flask a bibliotecii flask:

```
from flask import Flask
```

- Se crează aplicația Flask cu specificarea numelui variabilei `__name__`:

```
app = Flask(__name__)
```

- Se creează funcția ce sa returna această pagină (numită funcția view):

```
def index():  
    return '<h1>Salutare tuturor!</h1>'
```

- Se adaugă decoratorul `@app.route` funcției mai sus create:

```
@app.route('/')  
def index():  
    return '<h1>Hello Puppy!</h1>'
```

- Se execută aplicația:

```
if __name__ == '__main__':  
    app.run()
```


Scriptul complet

- Se creează un fișier app1.py ce va conține scriptul

```
from flask import Flask  
app = Flask(__name__)
```

```
@app.route('/')  
def index():  
    return '<h1>Salutare tuturor!</h1>'
```

```
if __name__ == '__main__':  
    app.run()
```

Execuția aplicației

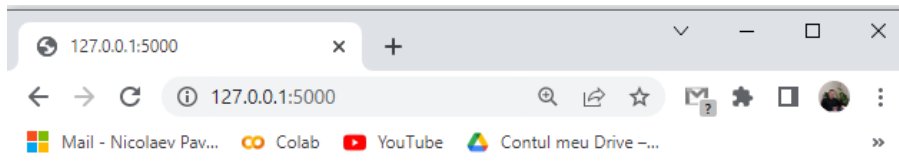
- În linia de comandă a mediului activa se execută comanda:

`python nume_fisier.py`

- Exemplu fișierului cu nume app1.py:

```
(mediul_flask) E:\Machine_learning\Python\Flask\Aplicatii>python app1.py
* Serving Flask app "app1" (lazy loading)
* Environment: production
  WARNING: Do not use the development server in a production environment.
  Use a production WSGI server instead.
* Debug mode: off
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

- Se deschide browser-ul și se introduce adresa `http://127.0.0.1:5000/`



Salutare tuturor!

3. Bazele rutării

- Am văzut cum se creează o aplicație web cu o singură pagină (care se returnează sub forma unui string).
- Dar cum se pot adăuga rute multiple spre mai multe pagini.
- Soluția se află în decoratorul `@app.route()`.
- Stringul care este trecut ca parametru decoratorului determină extensia URL transmisă funcției view.
- Pagina curentă sau domeniul este adresa locală <http://127.0.0.1:5000/>
- Se poate accesa adresa http://127.0.0.1:5000/nume_pagina dacă se specifică stringul `“/nume_pagina”` decoratorului
`@app.route(“/some_page”)`
- În cazul aplicării în producție, 127.0.0.1 se va substitui cu numele domeniului (de exemplu `www.site.com`)

Exemplu script rutare multiplă

- Se modifică conținutul fișierului app1.py

```
from flask import Flask
app = Flask(__name__)

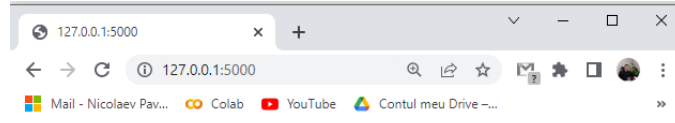
@app.route('/')
def index():
    return '<h1>Salutare tuturor!</h1>'

@app.route('/info')
def info():
    return '<h1>Acesta este un curs Flask</h1>'

if __name__ == '__main__':
    app.run()
```

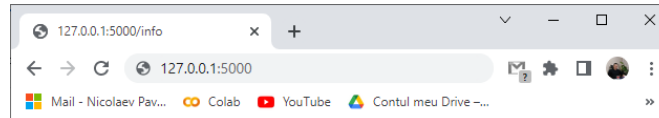
Execuția aplicației cu rutare multiplă

- Se repetă procedura de execuție a fișierului app1.py
- Se reîncarcă browser-ul și se introduce adresa `http://127.0.0.1:5000/`



Salutare tuturor!

- Se introduce adresa `http://127.0.0.1:5000/info`



Acesta este un curs Flask

4. Rutarea dinamică

Esența rutării dinamice

- De cele mai multe ori se dorește ca extensia URL să fie dinamică în funcție de situație.
- De exemplu se dorește câte o pagină pentru fiecare utilizator, deci extensia va fi în formatul

[www.site.com/nume utilizator](http://www.site.com/nume_utilizator)

- Pentru a ajunge la acest efect se utilizează rute dinamice.
- Rutele dinamice constau din 2 aspecte cheie:
 - O variabilă în rutare <variabila>
 - Un parametru transmis funcției view

Exemplu script rutare multiplă

- Se modifică conținutul fișierului app1.py

```
from flask import Flask  
app = Flask(__name__)
```

```
@app.route('/')  
def index():  
    return '<h1>Salutare tuturor!</h1>'
```

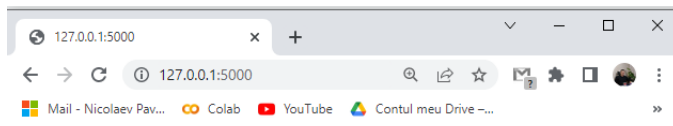
```
@app.route('/info')  
def info():  
    return '<h1>Acesta este un curs Flask</h1>'
```

```
@app.route('/<utilizator>')  
def utilizator(utilizator):  
    return f'<h1>Salutare utilizatorului {utilizator}!</h1>'
```

```
if __name__ == '__main__':  
    app.run()
```

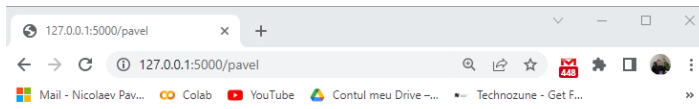
Execuția aplicației cu rutare dinamică

- Se repetă procedura de execuție a fișierului app1.py
- Se reîncarcă browser-ul și se introduce adresa `http://127.0.0.1:5000/`



Salutare tuturor!

- Se introduce adresa `http://127.0.0.1:5000/pavel`



Salutare utilizatorului pavel!

5. Debug Mode

Esența debug mode

- Pe durata elaborării aplicațiilor se pot realiza unele erori
- Pentru a putea capta aceste erori se setează parametrul `debug=True` la execuția aplicației elaborate.
 - `app.run(debug=True)`
- Modul debug, de asemenea, permite accesul al consola browserului!

Exemplu script cu erori fără capturare

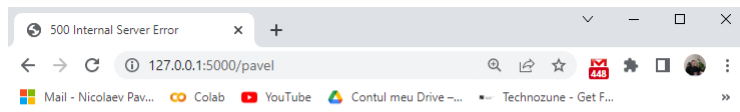
■ Se modifică conținutul fișierului app1.py

```
from flask import Flask
app = Flask(__name__)

@app.route('/')
def index():
    return '<h1>Salutare tuturor!</h1>'

@app.route('/<utilizator>')
def utilizator(utilizator):
    return f'<h1>Salutare utilizatorului {utilizator[100]}!</h1>'

if __name__ == '__main__':
    app.run()
```



Internal Server Error

The server encountered an internal error and was unable to complete your request. Either the server is overloaded or there is an error in the application.

Exemplu script cu erori cu captare

■ Se modifică conținutul fișierului app1.py

```
from flask import Flask
app = Flask(__name__)

@app.route('/')
def index():
    return '<h1>Salutare tuturor!</h1>'

@app.route('/<utilizator>')
def utilizator(utilizator):
    return f'<h1>Salutare utilizatorului {utilizator[100]}!</h1>'

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

