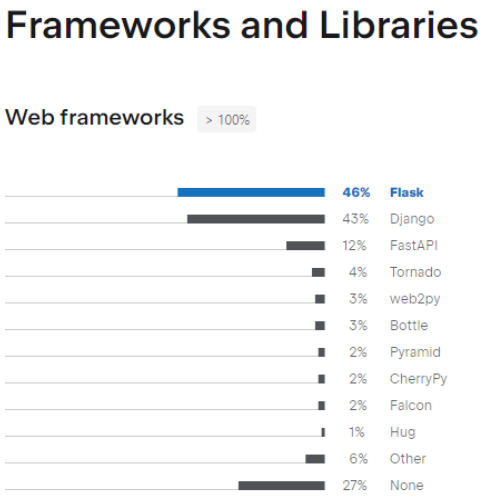
Bălașa Antonio, Chung Hanna, Păun Dani
Grupa 609 – Web
15 ianuarie 2023

Proiect 1 – Revizuire de Framework “Flask”



Dintre toate framework-urile folosite pentru a crea o aplicație Web, Flask este unul dintre cele mai folosite.[1] Mai anume, acest framework este folosit pentru a gestiona interacțiuni între front-end (elemente de HTML și JavaScript) și back-end (aplicație de Python și baze de date, e.g., SQLite).

Ca să înțelegem ce mai exact face Flask în nivel conceptual, să zicem că avem o pagină de front-end care aranjează poze și text în mod frumos cu ajutorul lui HTML și CSS. La un moment dat, ne plicitisim de pagini statice care doar aranjează text și poze. Vrem această pagină de web să interacționeze cu acțiunile utilizatorului. Dacă doar era vorba de a schimba atributele unor elemente pe baza unor evenimente declanșate de utilizator sau de a stoca datele pe care un utilizator le-a oferit pentru o clipă la o variabilă în memorie locală, puteam să folosim JavaScript. Dar aceste utilaje sunt insuficiente dacă vrem să prelucrăm cu datele care nu sunt generate fix în momentul folosirii.

Dacă vrem să accesăm datele de la o sursă externă (e.g., date care sunt salvate din sesiuni trecute, alte date de la alți utilizatori, sau de la o bază de date), trebuie o modalitate prin care pagina Web ar putea să acceseze aceste date stocate în altă parte, care ar putea să fie accesate și prelucrate prin ajutorul unui alt limbaj de programare (e.g., Python). Ca să facă o legatură între cerințe generate în JavaScript la pagină Web și funcții care gestionează datele în Python, trebuie un API—ceva care poate să primească o cerere pentru o anumite acțiune de la pagină Web prin HTTP (GET, POST, PUT, DELETE) și să cheme o funcție în Python care ar putea să facă această acțiune pe bază de date.[2] După aceea, ar fi nevoie și de o legatură prin care ceea ce returnează funcțiile Python trebuie să ajungă înapoi la pagina Web să fie prelcurată mai departe în JavaScript sau afișată prin HTML. Flask ne ajută să creeăm acest API.

Afară de capacitatea lui Flask să faciliteze crearea API-ului, Flask oferă și alte avantaje. Este un micro-framework cu puține dependențe la alte biblioteci: Werkzeug și jinja2. Werkzeug,

[1] Python Developers Survey 2020 Results. (2020). https://www.jetbrains.com/lp/python-developers-survey-2020/.
[2] IBM. What is a REST API? https://www.ibm.com/topics/rest-apis.

care este bazat pe Web Server Gateway Interface (WSGI), dă un standard care stabilește cum să comunice între servere și aplicații web și gestionează partea de cereri și răspunsuri cu servere. Jinja2, în schimb, este folosit pentru a ușura codarea prin șabloane (templates), în care putem să folosim variabile de la Python de la back-end să dea valori pentru a folosi în tag-uri și coduri la front-end. Prin folosirea șabloanelor, nu trebuie să codăm repetitiv aceeași instrucțiuni sau mai multe variante de pagini web asemănătoare; putem să folosim aceste șabloane.[3]

Urmează o explicație despre cum să instalăm Flask și cum funcționează. Noi am creat o mini-aplicație, Spânzurătoarea, ca un exemplu funcțional utilizând Flask, pe care puteți găsi în GitHub pe acest link: https://github.com/Titanul1/Proiect-Seminar-Web.

Cum să instalăm Flask?

Pentru că Flask este un modul care funcționează doar cu Python, mai întâi trebuie să instalăm Python. Flask este compatibil începând cu Python versiunea 3.7, deci trebuie să alegem o versiune 3.7 sau mai recentă.[4] Putem să instalăm Python pe python.org sau îl descarcăm prin Microsoft Store dacă avem Windows ca OS.[5] Putem intra în Python și să verificăm dacă este instalat la Command Prompt (dăm click pe butonul de Windows și tastăm "cmd") dacă tastăm "Python". Dacă este instalat, sistemul va intra în Python și va afișa pe ecran versiunea instalată.

După această verificare, putem instala Flask prin Python Pip, un component pentru gestionarea modulelor, care vine automat cu versiuni recente de Python. Intrăm în cmd, și tastăm "pip install flask". După aceea, Flask va fi o parte de biblioteci disponibile pentru folosire în Python și putem să-l utilizăm în orice code de Python cu o declarație la început: "from flask import Flask". Codarea cu Python se poate întâmpla în Python shell în sine, printr-un IDE (integrated development environment) precum Python IDLE, sau printr-un editor ca VS Code.

În plus, documentația pe Flask recomandă folosirea de un "virtual environment", pentru că diferite aplicații de Python ar putea să folosească diferite versiuni și combinații de biblioteci de cod, sau chiar diferite versiuni de Python. Pentru a atașa Flask pentru un proiect anume într-un virtual environment, folosim modulul "venv" (deja disponibil cu instalare de Python). Mai întâi facem un dosar pentru proiectul nostru. Intrăm în acest dosar și creeăm un virtual environment și îl dăm un nume cu "py -3 -m venv nume" unde "nume" aici reprezintă un path anume unde vrem

[3] Python Tutorial. (2021). What is Flask Python. https://pythonbasics.org/what-is-flask-python/.
[4] The Pallets Projects. (2010). Installation. https://flask.palletsprojects.com/en/2.2.x/installation/.
[5] Python. (2023). Download Python. https://www.python.org/downloads/.

acest virtual environment să locuiască. Activăm acest environment prin “nume\Scripts\activate” și acolo instalăm, ca de obicei, cu “pip install Flask”.[6] Pentru noi, pentru că foloseam VS Code, am mai intrat în “View” și “Command Palette” și am selectat “Python Select Interpreter”, de unde am selectat Python 3.10.

Cum funcționează Flask?

Trecând de la discuția la nivel conceptual la nivel practic, facem o revizuire despre ce componente exact trebuie să apară în coduri folosind Flask, pas cu pas. Pentru că Flask este un framework bazat pe Python, începem cu coduri de acolo. Urmează cum ar arăta structura minimă când folosim Flask într-un cod de Python, și o explicație scurtă despre ce face fiecare linie de cod:

```
from flask import Flask          #Aici adaugăm bibliotecă de functionalități de la framework Flask.
app = Flask(__name__)            #Aici Flask() este un fel de constructor care creează un obiect de
                                 #clasa Flask, și inițializăm cu __name__, care identifică în ce scop
                                 #suntem (practic, în ce fișier de .py suntem). Acest __name__
                                 #asigură că obiectul app de tip Flask este creat în scop unde lucrăm.
@app.route(`/ceva`)              #Aici acest obiect app o să redirecționeze un browser care se duce
                                 #la adresa pusă în paranteză să declanșeze orice funcție care
                                 urmează #imediat după acest @app.route. Vom discuta mai încolo
                                 niște #metode pe care am putea pune după parametru de locația
                                 paginii.
def o_functie():
    #Aici putem să scriem instrucțiuni de cod. Dacă nu specificăm altfel, ceea ce returnăm va
    #fi în formă de HTML și va apare când intrăm în adresa web “../ceva”.
    return `<p>Aici este un exemplu</p>`
if __name__ == `__main__`: #Aici verifică dacă scopul în care rulăm instrucțiunile este de fapt
    app.run()                #cel principal pentru acest fișier, adică __main__. Dacă da, atunci
                             #app.run() o să inițializeze un server local când acest __main__
                             #rulează. Acest server pot fi găsit la http://127.0.0.1:5000/ceva.
```

[6] The Pallets Projects, citat pe 4.

Bălașa Antonio, Chung Hanna, Păun Dani
Grupa 609 – Web
15 ianuarie 2023

Dacă vrem ceea ce returnăm să interacționeze cu alte elemente într-o pagină web, putem să folosim una dintre funcționalități importante pe care Flask oferă—șabloane (templates)—să inserăm ceea ce returnăm în niște HTML, astfel:

Să zicem că avem o pagină de HTML undeva (“generic.html”), unde avem anumite părți pe care vrem să le schimbăm în funcție de ce răspunsuri primim de la back-end. Pentru că Flask are jinja2, ne dă posibilitatea să mărcăm părți din instrucțiunile noastre de front-end cu două acolade să semnaleze că variabilă înauntru (“mesaj”) va schimba în funcție de ce primim.

```
<html><body>Vreau sa-ti spun un mesaj important: “{{mesaj}}”.</body></html>
```

Pentru că intenționăm să folosim acest fișier de HTML ca un șablon, îl punem într-un dosar numit /Templates asociat cu proiectul nostru de Python. (Locația exactă contează pentru că jinja2 este programat să caute șabloane la /Templates.) În fișierul nostru de Python care inițializează aplicația noastră, trebuie să adaugăm funcționalitatea aceasta (render_template) și o altă funcționalitate pentru această aplicație de back-end să primească argumente din cereri front-end (request).

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

În rest, la fel ca sus, inițializăm un obiect app și definim un endpoint @app.route(`/ceva`) și îl asociăm cu o funcție. Însa, data această, mai adaugăm în funcție o instrucțiune să detectăm ce argumente au fost trimise de la front-end și să căutăm o variabilă anume dîn ele, prin request.args.get(), punând numele variabilei în ghilimele (să zicem, “cesaspun”) ca parametru.

```
propozitie = request.args.get(“cesaspun”)
```

După ce stocăm valoarea parametrului într-o variabilă locală (exemplu: “propozitie”) și să prelucrăm cu variabila ca orice variabilă în Python. La sfârșit putem să returnăm orice rezultat după prelucrare cu return, numai că data această, folosim render_template care primește doi parametri: numele fișierului de HTML unde am avut șablonul și numele variabilei în șablon setat cu o valoare pe care luăm din prelucrările noastre de back-end. De exemplu:

```
return render_template(“generic.html”, mesaj=propozitie)
```

Atunci, data viitoare instrucțiuni din front-endul nostru accesează /ceva, va arăta o copie exactă pe care Flask a luat de la generic.html, numai că a înlocuit orice a fost în render_template în loc de {{mesaj}}. De exemplu, dacă creeăm un <form action=“/ceva?cesaspun=Buna> într-o altă pagină de HTML și chemăm endpoint-ul nostru, ar declanșa o_functie și render_template să creeze o pagină pe /ceva bazat pe șablonul generic.html cu {{mesaj}} înlocuit cu valoarea lui cesaspun

(Buna) primit prin folosirea simbolului `?` (un simbol care semnalează că orice denumire și valoare urmează vor fi trimise la back-end prin metoda GET din http). Odată cu declanșarea lui /ceva, request.args.get(cesaspun) este gata să preia variabila trimisă de aceeași nume. De asemenea, cu Flask putem să înlocuim nu numai variabile prin {{ }}, dar și linii întregi de cod prin {% extends "generic.html"}{% block body %} {% endblock %}, unde punem cod între block body și endblock.

Însă acest exemplu simplu de Flask lipsește interacțiuni cu o bază de date. Dacă doar am vrut să gestionăm niște acțiuni de la utilizator sau să lucrăm cu niște variabile luate de la front-end fără să le stocăm de la o sursă externă sau să luăm ceva de acolo, am fi putut să folosim funcții de front-end în JavaScript, fără să umblăm cu Python și Flask. Flask este folositoare nu numai pentru șabloane care fac pagini mai dinamice, dar și pentru interacțiuni cu date dintr-o sursă externă prin HTTP. Pe scurt, dacă vrem să creeăm funcții pe Python să facă astfel de interacțiuni, trebuie să adaugăm orice dintre metode pe care le intenționăm să folosim la @app.route(`/`, methods = [`POST`, `GET`, `PUT`, `DELETE`]). Dacă vreau să primesc date pentru salvare, trebuie să includ POST. De acolo, putem să includem în funcția noastră în Python coduri să interacționeze cu date salvate undeva. De exemplu, dacă am un fișier bazadate.csv cu un atribut și o valoare (două coloane), aș putea să pun "import csv" la început să adaugăm funcționalități pentru fișiere de tip .csv, și în funcția o_functie să adaugăm valori luate de la request.form.get(), folosind fisier = open("bazadate.csv", "a") să pregatesc un obiect deschis pentru adaugare (parametru "a"). După aceea, putem crea un obiect de tip "writer" inițializat cu acest fisier (de exemplu: scriitor = csv.writer(fisier)), care scrie o înregistrare în forma unui tuplu—de exemplu: scriitor.writerow((camp, val)), unde valorile de la camp și val am luat de la elemente din HTML prin request.form.get(). După ce terminăm cu adaugarea înregistrărilor, închidem scriitorul cu scriitor.close(). De asemenea, putem citi înregistrări prin open("bazadate.csv", "r") și csv.reader.[7]

Concluzie

Flask este un microframework simplu dar foarte folositor pentru a genera pagini dinamice prin șabloane și a trimite/primi date între front-end și back-end. Acum că știm niște funcționalități importante pentru Flask și cum să le implementăm în cod, putem exersa practic ce noi am învățat prin aplicația noastră, Spânzurătoarea. Vă invit să vedeți acest exemplu de Flask în acțiune.

[7] freeCodeCamp.org. (2019). Web Programming with Flask - Intro to Computer Science - Harvard's CS50 (2018). https://youtu.be/zdgYw-3tzfI.