MEEC - ADInt

Laboratory 3

In this laboratory students will learn how to develop web applications (with static pages) using a web programming framework.

Students will also learn how to transfer documents from the server to browser.

The application code (on the server) will be developed in python using the Flask framework.

Student should look with attention at the two provided applications to better understand the presented contents.

1 Flask

https://flask.palletsprojects.com/en/1.1.x/

https://www.tutorialspoint.com/flask/

https://blog.miguelgrinberg.com/post/the-flask-mega-tutorial-part-i-hello-world

Flask is a web programming framework that allows programmers to develop web applications.

Flask hides completely all the network and low level programming details. Programmers only have to define the endpoints (URL) of the various pages, the code to be executed when the server receives the requests from the browsers, and the HTML to be sent to the browser.

1.1 Flask installation

https://flask.palletsprojects.com/en/1.1.x/installation/

https://blog.miguelgrinberg.com/post/the-flask-mega-tutorial-part-i-hello-world

Flask is provided as a regular Python library, as such students should use the pip command to install it:

```
$ pip install flask
```

1.2 Flask usage

https://flask.palletsprojects.com/en/1.1.x/quickstart/

https://blog.miguelgrinberg.com/post/the-flask-mega-tutorial-part-i-hello-world

https://www.tutorialspoint.com/flask/flask_application.htm

https://www.tutorialspoint.com/flask/flask_routing.htm

Flask provides mechanisms to define the endpoints, process the data that is sent from the browsers and a library to use templates in the production of the HTML.

For a minimal usage of flask, it is necessary to import the library, create a flask application object and define the endpoints and the code to be executed.

The flask framework will handle all the network communication protocols and data transfer between the browser and the code

```
1  from flask import Flask
2  app = Flask(__name__)
3  @app.route("/")
4  def index():
5     return "Hello "
6  if __name__ == "__main__":
7      app.run(host='0.0.0.0', port=8000, debug=True)
```

Line 1 imports the Flask class that is used in line 2 to create the **app** object.

From this moment on, it is possible to define the endpoint from where the server will receive requests.

In lines 5..7 the **app** is configure to run the **index()** function whenever a request is received in the **/** endpoint.

After all the configurations it is possible to start the server (line 9)

This server will accept connection on all network cards (**0.0.0.0**) on port **8000** and debug information will be generated.

When running the application it is possible to access various URLs:

- the URL http://0.0.0.0:8000/ will show on the browser the work Hello
- the URL http://0.0.0.0:8000/index.html will show in the browser an error.

It is possible to assign multiple endpoints the same functionalities

```
1  @app.route("/")
2  @app.route("index.html")
3  def index():
4     return "Hello "
```

or define different functions for different endpoints

```
5  @app.route("/otherEndpoint")
6  def otherFunction():
7     return "<b>other</b> endpoint"
```

Flask will format the resulting string into HTML, so that the browser can format it correctly.

1.3 Flask templates

https://flask.palletsprojects.com/en/1.1.x/tutorial/templates/

https://jinja.palletsprojects.com/en/2.11.x/

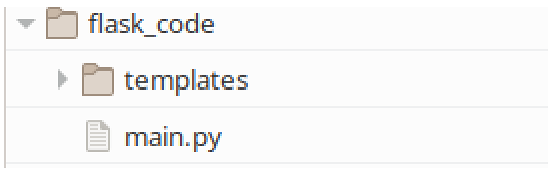
https://blog.miguelgrinberg.com/post/the-flask-mega-tutorial-part-ii-templates

https://www.tutorialspoint.com/flask/flask_templates.htm

In order to ease the production of the HTML that will be visible in the browser, Flask uses a template system.

This template system allows the definition of a file, mostly with constant content and with placeholders that will be replaced by the values of python variables.

Templates are regular HTML files that should be stored in the template directory:



To use a template the programmer should import the render_template method

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

and then use it to generate the HTML

```
return render_template("newFile.html")
```

The previous example copies the whole content of the **newFile.html** file, without doing any transformation.

If the template has placeholders for the value of some variables, it is possible to assign such placeholders with the correct values when calling the **render_template** method:

```
return render_template("index.html",
                       message="Dynamic content!")
```

here, the internal **message** placeholder will be instantiated with a string.

When defining the template file, the placeholders are referred inside **{{ }}**:

```
    <div>
      <p>{{ message }}</p>
    </div>
```

Template content can be inherited:

index.html

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

{% block content %}
    <div>
      <p>{{ message }}</p>
    </div>
{% endblock %}
```

layout.html

```
<!DOCTYPE html>
<html>
 <head>
  <title>Title </title>
 </head>
 <body>
 {% include 'pageHeader.html' %}

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

Other templates can also be included to reuse the page definition from other files.

For complex python variables (lists, dictionaries, it is possible to iterate on their values using pseudo python instructions or even use condition values to render different parts of the document

```
<ul>
  {% for j in numbers %}
    <li>{{j}}</li>
  {% endfor %}
</ul>
```

```
<title>
Hello from Flask
</title>
{% if name %}
  <h1>Hello {{ name }}!</h1>
{% else %}
  <h1>Hello, World!</h1>
{% endif %}
```

In HTTP the browser can send data to the server using GET or POST methods.

The same endpoint/URL can be used with a GET or POST method, and as such in the application code, the programmer should be able to distinguish such methods.

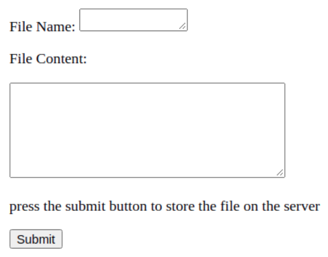
In the route definitions (**@app.route(str)**), if no method is defined the system assumes the **GET** method.

To distinguish code to these two methods the programmer should use the **methods** argument inside the **@app.route**:

```
@app.route("/getFile",
           methods=['GET'])
def newJobGET():
 ...
```

```
@app.route("/getFile",
           methods=['POST'])
def newJobPOST():
 ...
```

The post method is usually used to transmit the data that the user inserts in forms from the browser to the server.



```
<form action = "newFile" method="post">
File Name:
    <textarea id="fileName" name="fileName">
    </textarea>
<p>File Content:</p>
    <textarea id="fileCont" name="fileCont">
    </textarea>
...
   <input type="submit">
</form>
```

The **@app.route("/newFile", methods=['POST'])** endpoint will be called from the browser when the user clicks on submit.

For the server code to access the data transmitted by the browser it is necessary to use the **request** object:

```
from flask import request
. . .
```

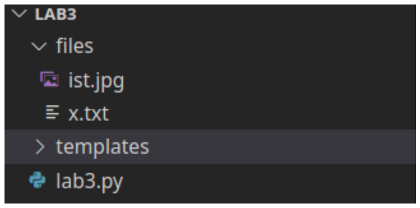
```
@app.route("/newFile", methods=['POST'])
def newFilePost():
    print (request.form)
```

This object is a dictionary with the data that the inserted in the browser form.

2 Exercise 1

Implement a simple web server that allows users to download files from a directory on the server.

The available files for download are stored in the **files** directory.



The **flask** server should show a list of files available on the **files** directory through the following URL: http://0.0.0.0:8000/listFiles

Each file can be downloaded writing in the browser the corresponding URL:

- http://0.0.0.0:8000/getFile/x.txt
- http://0.0.0.0:8000/getFile/ist.jpg

The server will also implement endpoint that allows the download of the list of files and two other to the creation of files.

For the resolution of this laboratory assignment, students should continue the implementation that is partially provided.

The file **lab3** directory contains the **lab3.py** files and a series of templates that already provide the skeleton for this laboratory functionalities. Two files to be downloaded are also provided.

Before starting to implement the assignment, student should look to all the provided code (initialization, routes, and functions) and templates, and relate them to the documentation of Flask.

Students should also execute the provides code and understand what is working (relate this to the provided code) and what is missing.

For the completion of this laboratory students should follow the **TODO** notes scattered in the code. Depending of the situations students should:

- complete the provided functions
- create new templates
- create new endpoints
- …

2.1 TODO 1

Get the list of files in the directory and send it to the template.

To accomplish this, students should use the **listdir** function from the **os** library

https://docs.python.org/3/library/os.html

https://docs.python.org/3/library/os.html

2.2 TODO 2

Change the listFiles.html template to show the files names and the corresponding download URLs.

2.3 TODO 3

Verify if the file exists.

If the file does not exist the browser should be redirected to a specific page that shows the file name.

https://flask.palletsprojects.com/en/1.1.x/quickstart/#redirects-and-errors

https://www.tutorialspoint.com/flask/flask_redirect_and_errors.htm

2.4 TODO 4

Create a new page that only shows the file name that does not exist but was tried to be downloaded.

2.5 TODO 5

Send the content of the file to the browser using the **send_from_directory** function.

https://flask.palletsprojects.com/en/2.0.x/api/

2.6 TODO 6

Implement an endpoint (http://0.0.0.0:8000/files) that returns that list of files to the browser as json.

https://flask.palletsprojects.com/en/2.0.x/api/

2.7 TODO 7

Implement the method that is called by the http://0.0.0.0:8000/createFile page.

This endpoint will receive the file name, the text content and create such file. Afterwards it is possible to download it using the provided endpoints.

https://www.tutorialspoint.com/flask/flask_request_object.htm

2.8 TODO 8

Implement all the necessary pages (template and form handling) so that the user can upload a new file

https://www.tutorialspoint.com/flask/flask_file_uploading.htm

3 Exercise 2

Implement a python program that contacts the server developed in the previous exercise and downloads all the available files.

To implement this program access the previously developed endpoints using the **requests** API

https://docs.python-requests.org/en/latest/

Install it using the pip command

```
pip3 install requests
```

The basic usage for this library is:

GET

```
import requests

r = requests.get(URI)
print(r.status_code)
data = r.json()
print data
```

POST

```
import requests
payload = {'ids': [12, 3, 4, 5, 6] }
r = requests.post(ENPOINT_URL,
                  json=payload)
r = requests.post(ENPOINT_URL,
            data=json.dumps(payload))
print(r.text)
```

To save the downloaded file, follow first instructions of this tutorial:

https://www.tutorialspoint.com/downloading-files-from-web-using-python