Segmentation Server

**Intentions, Considerations and Assumptions:**

In this implementation, I assumed that the client could make input errors that need to be addressed, so I made sure every request will be checked for errors, and to inform the client on what exactly caused the error.

I implemented the server to be as compatible as possible, while making the server's port, IP and internal directories easy to change. I assumed python 3.8 (or higher) is installed on the host system, and every other module used by the server will be installed automatically by the bash script (or already installed if using the docker-image).

I placed high emphasis on documentation and readability and made sure that the logged messages will be as informative as possible.

I used Flask web framework, which is simple, flexible, and easy to use.

**Project's Structure:**

1. All required modules and files are included in the Segmentation Server folder.
2. The folder **additional\_files** contain a postman collection example on how to run the server and the various endpoints.

It also contains some images and files used in the postman requests. Those files should be copied to "C:\Users\user\Postman\files", for the requests to work straight out of the box. You can also choose your own files for the requests.

1. **server.py** will handle the client requests using the Flash module, while logging each action using the built it logger system in python.
2. **server\_functions.py** contains some helper functions that the server will use (e.g. creating and setting up the loggers)
3. **server\_properties.py** contains some settings the server will run with- port number, IP number, and directories paths.
4. A **Dockerfile** containing the commands on how to build the docker image (with comments on how to build and run them)
5. A **bash script** "run\_server.sh" as explained below.
6. The server will create (or delete and re-create) the following **directories**:
   1. **image\_queue:** will contain all the images uploaded by the client, which will be processed and segmented by the model.
   2. **segmented\_images:** will contain the segmented folders created by the model. If the multiple images endpoint was chosen by the client, the server will archive them together to send back to the client, if the single image endpoint was chosen, only that single image will be sent back to the client.
   3. **archives:** will contain the archive created from the segemented\_images, in the case that the multiple images endpoint was chosen by the client. This archive will be sent back to the client as a response to the request.

* The directories names and path can be changed through the server settings in the server\_properties.py file.

**The running process of the server:**

1. Run the bash script **run\_server.sh** (using "bash run\_server.sh" in the CMD) to **install** all requirements for the project, and to start **running** it (or use the docker-image as mentioned below)
2. **Dockerfile** is included to check the commands used to build the docker-image file and to test it yourself.

To use a finished docker-image, you can pull it from docker-hub using:

* By searching for the tag **"idansm/segmentation-server:1.0"**
* By using the terminal command: **"docker pull idansm/ segmentation-server:1.0"**.
* Command to run the docker container pulled from docker-hub:

**"docker run --name segmentation-server -p 8989:8989 idansm/segmentation-server:1.0"**

1. To send the requests to the server, you can use the postman collection included in the additional\_files folder (with example pictures and files as described above).

**Server endpoints:**

1. **Endpoint:** '/'

**Method:** 'GET'

An empty request that will get a response from the server if it is running. Used to check that the server is running properly, and that a connection can be established.

1. **Endpoint:** '/perform-segmentation'

**Method:** 'POST'

The server will expect to receive a request with a form-data containing a key of 'image' and value of a single image.

The server will check for bad requests, such as bad key, no image uploaded, or multiple images uploaded instead of one. It will also check for the format of the image received (only png, jpg and jpeg formats will be accepted).

1. **Endpoint:** '/perform-multi-segmentation'

**Method:** 'POST'

The server will expect to receive a request with a form-data containing a key of 'images' and value of at least one image.

If a file was uploaded and it was not in the correct format (png, jpg or jpeg) it will continue to the next file without sending any message back to the client, so that only the correct format files will be processed.

The server will check for bad requests, such as a bad key, or no images uploaded.

1. In case of an **internal server error**, an appropriate message will be sent to the client with the error code **500** (INTERNAL\_SERVER\_ERROR).
2. In case of a **bad endpoint** in the request, an appropriate message will be sent to the client with the error code **404** (NOT\_FOUND).
3. Every **request** (and request error) will be logged and shown to the server terminal using the **request\_logger**, and every action (if successful or an error has occurred) the server has done will be logged and shown using the **server\_logger**.