ADVANCED SOFTWARE ENGINEERING – LAB TEST EXERCISES – 3rd December 2024

NAME: ____________________ SURNAME: ____________________ ID (MATRICOLA): ____________
YOUR CODE (Needed in some exercises): 123456

Instructions

It is mandatory to put the number (YOUR CODE) of the header in the first line of the *solutions.txt* file. A missing or misspelt code is considered an automatic fail.
For example, if your code is 123456 the first line of the *solutions.txt* must be:

Your code =:= 123456

DISCLAIMER: This is an example of a sheet for the lab test. Ignore the delivery instructions, they are there only to show you the real case scenario. The number of exercise of this sheet is greater than the one on a real lab test. Here I put an exercise for each topic to cover all the syllabus. In the exercises, it is asked to use YOUR_CODE. For these mock exercises, the number is 123456. In a real lab test, every student will have a different number.

You will have 60 minutes to deliver your solution. The deadline is strict, in the course Moodle you will not be able to upload anything after the deadline. If you deliver the solution, you must also give back this sheet.

You can access the slides of the course from the Moodle and the documentation of the tools to be used in the test. Any other material, website or application (generative AI included, e.g. Copilot) consulted will result in an automatic fail of the test.

You have to retrieve the files of the test and upload an archive file with the requested folders and files as a solution before the deadline.

The test assumes a clean Docker environment containing only the image `python:3.12-slim`.

Material description

Download *material.zip* file from the Moodle.
It contains the following:

- folder *app*: a folder containing Flask code, specifically:
  - file *app.py*: python code file of the service you will modify;
  - file *requirements.txt*: file with requirements for running the service;
- file *locustfile.py*: file to run the performance test;
- file *solutions.txt*: text file you have to fill with answers, add only text after the =:= symbols without adding new lines, escape symbols, or quotes (yes, you can put a space after the symbols):

Delivery - 6 points needed to pass

In general the exercises will guarantee a maximum of 11 points. You need to reach a score of 6 to pass the test. This sheet has a number of exercises which sum over 11 to cover all the topics in the syllabus. Put the filled *solutions.txt* file in the Moodle submission. All the other files must be on the GitHub repository needed in exercise 1. If you commit something *after* the deadline, your lab test will be automatically discarded and it is considered failed.

The solutions will be automatically evaluated by an offline script, so be careful to not modify the structure of directories and of the solution file to fill. Add only the answers to the exercises and do not insert new lines, comments, or anything else. To pass this test you need to reach 6 points.
Every exercise but the `Dockerfile` one, will grant points for partial solutions.

At the end, your repository must have the following structure:

```
.
├── app/
│   ├── app.py #to modify in ex 3
│   ├── asekey.pem #created for exercise 2
│   ├── asecert.pem #created for exercise 2
│   ├── Dockerfile #created for exercise 4
│   └── requirements.txt
├── docker-compose.yml #created for exercise 8
├── Dockerfile_locust #created for exercise 9
├── locustfile.py #to modify in ex 6
├── mockex.json #created for ex 10
└── solutions.txt
```

1 Git and GitHub (2 points)

Create a public repository in your account called *ASE00-YOUR_CODE* where YOUR_CODE is the number in this sheet header. Move all the files downloaded from the Moodle to your local repository. Write the HTTPS link of your repository in the *solutions.txt* file, answering the first question. When you have completed all the following exercises, push your local repository into the remote one with commit message "Submitted". Upload a copy of your *solutions.txt* file in the Moodle delivery to confirm your submission before the deadline.

2 HTTPS certificates (2 points)

Create a TLS x509 certificate and its private key. Put them inside the `app` folder.
The key:

- must be stored in a file called `asekey.pem`,
- must not be encrypted,
- must use RSA encryption algorithm,
- its size must be 4096 bits.

The certificate:

- must be stored in a file called `asecert.pem`,
- must expire after 100 days,
- must refer to the common name `localhost`,
- must have in the organization name YOUR_CODE.
- you can put whatever you want in other fields.

3 Flask (2 points)

Inside the *app* folder, open the *app.py* and add an endpoint responding to an HTTP GET at

`/ex1?a=X`

which should calculate the following function

$$f(X) = D/C$$

where X is a string, D is the number of digits in X, C is the number of alphabetic characters in X, and the division is the *integer division.* To accept an operation, at least one digit must be part of X and you must avoid division by zero.

The function result should be returned in a JSON file {'s': result}, where *result* is an integer.
The endpoint should return the following HTTP codes:

- 200: when an operation is completed without problems.
- 400: if the operation cannot be performed, with payload a JSON file {'s': "Invalid Input"}.

4 Dockerfile (3 points)

In the same folder, create and write a `Dockerfile` to make the following image:

- based on the image `python:3.12-slim`,
- take only `app.py` and `requirements.txt` from your system and put them in the image inside a folder called *YOUR_CODE* where YOUR_CODE is the number in this sheet header,
- make that folder the working directory,
- use pip to install the requirements of the *requirements.txt* file,
- make the containers based on the image listen to port 5001,
- set as starting command the command to run Flask listening to the host `0.0.0.0` and the port of the previous point.

Do not add unnecessary commands, e.g. `apt-get update`. Do not create a directory with `mkdir`. Do not use the flag `--no-cache-dir` when using pip.

5 Container execution (4 points)

In the *solutions.txt* file, write the following:

- The command to build a docker image from the previous Dockerfile and tag it with `mockex`.
- The command to run a container based on such image which is reachable from port 5002 and named `mockexcont`.
- The complete string of the PORTS field of the container after executing the command to list all the running containers.
- The value of the field [“Config”][“Volumes”] from the JSON output of the docker command to inspect the executed container.

The commands must not have unrequested options.

6 Locustfile (1 points)

In the root folder, add to the *locustfile.py* a test which invokes the endpoint developed in Exercise 3 with a value of X equal to “C1a0”. The test must be a task with weight equal to 1.

7 Dockerfile 2 (2 points)

In the root folder create a `Dockerfile_locust` to make the following image:

- based on the image `python:3.12-slim`,
- take only the `locustfile.py` and put it in the image,
- use pip to install `locust`,
- make the containers based on the image listen to port 8089,
- set as starting command the command to run `locust`.

8 Docker compose (3 points)

In the root folder, create and write a `docker-compose.yml` file to run the application having two services and two secrets containing the certificate and key of Exercise 2. The services are:

- `ex1`, which must be built from the Dockerfile of Exercise 3 and must have the exposed port bound with the 8089 of the local machine. Overwrite the command in order to launch the Flask service with HTTPS using the above secrets.
- `locust`, which must be built from the Dockerfile of Exercise 7 and must have the exposed port bound with the 8090 of the local machine.

9 Locust execution (3 points)

Run the services with `docker compose`. Use a web browser to reach locust's web service and run performance tests on the application. Select 100 users, 10 as spawn rate and use the base URL of the Flask service as the host.
Analyse the statistic table and fill the *solutions.txt* file answering the following questions (wait at least 10 seconds to collect significant statistics):

- What is the endpoint's name with the most number of requests?
- What is the endpoint's name with the most number of failed requests?
- What is the endpoint's name with the highest median response time?

10 Postman (3 points)

Create a Postman collection called `mockex.json` with tests for the endpoint developed in Exercise 3. You have to make a request for testing (removing the quotes):

- X = "1a0".
- X = "123".
- X = "ciao"

Each test should check the expected HTTP code and response body.

11 Bandit (2 points)

Run Bundit on the *app* folder. Check the output table and fill the *solutions.txt* file with the requested answer about:

- Total number of issues.
- Number of issue with *high* severity.

12 Extra (1 point)

Run the container of Exercise 3 with `Docker` or `Docker compose` and perform an HTTP GET to the endpoint `/secret?X=code`, where `code` is YOUR_CODE of this sheet header. Use your browser to check the HTTP **header** named `exam` in the HTTP response and put its value in the *solutions.txt* file (without any quotes).