

Python Software Engineer - Coding task

Test executor web application:


Implement a web application that provides a central place to run Python-based tests. These tests will be run in a shared environment that provides limited hardware resources to developers.

To have some sample tests, just create dummy Python tests (some failing, some not, maybe add some delays for more realism).

You might also run the tests for your own application if you like.

The tests will be executed right on the machine that hosts the test executor application.

The application might look remotely like this:



New request

Submit

Test execution requests

Request ID	Requested By	Created At	Test Env	Test Path	Status	Details
18	John	2019-03-11 11:32:15	80	Tests/test_stgs.py	SUCCEED	View Details
16	John	2019-03-07 16:16:35	100	Tests/test_2.py	RUNNING	View Details
15	John	2019-03-07 10:37:12	100	Tests/test_vms.py	SUCCEED	View Details
12	Janie	2019-03-06 14:36:45	100	Tests/test_nics.py	FAILED	View Details
10	Johnny	2019-03-06 14:36:45	100	TestsServers/	SUCCEED	View Details
5	John	2019-03-05 13:28:05	10	Tests/	SUCCEED	View Details
3	John	2019-03-03 13:27:58	45	Tests/test_1.py	FAILED	View Details
2	Jane	2019-03-02 13:26:17	50	Tests/test_storages.py	SUCCEED	View Details

Copyright © 1&1 IONOS Cloud GmbH 2019

Test details

ID	17
Requested by	John
Env	1
Path	Tests/test_nics.py
Status	FAILED

Logs

```

Tests/test_nics.py F. [100%]
===== FAILURES =====
test_add
def test_add():
> assert False
E assert False
Tests/test_nics.py:2: AssertionError
===== 1 failed, 1 passed in 0.02 seconds =====

```

Features to be implemented:

- Create a new test run, providing:
 - Username
 - Test environment ID (choose an ID between 1 and 100)
It must not be possible to run a test in an environment where there is already a test running!
 - Choose one or more files to test, available tests are read from file system
 - The interface between the test executor web application and the Python test runner itself can be chosen freely
- List all previous test runs as a table including their outcome (failed, success, running)
 - Bonus:
Live-update of running/finished tests via Ajax or Websockets
- List details for one test run
 - Show all data provided in 1.
 - Show all logs that a test created
 - Bonus:
Live-update the current state and the logs of this test run via Ajax or Websockets

Requirements:

You are free to choose any library, tool and framework that helps fulfilling the task. The only hard requirement is to use Python (3.6+) for the backend.

It is important that the resulting code is clean, concise, maintainable and (not overly) documented.

We do expect you to write tests for your backend code.

Backend:

The backend of the web application should be implemented as a REST or GraphQL service (we use Flask and Marshmallow).

To persist data, use any database (SQL or NoSQL, depending on what fits best in your opinion).

Frontend:

For the frontend, choose any Javascript framework you like (we use React), or, if you find it easier, don't use any framework.

Note: Some basic HTML templates are provided so you can focus on implementing the most important features (feel free to use your own templates if you like).

Tests Runner:

Choose any test runner you like – unittest, py.test, nose or whatever.

Installation:

The resulting application should be runnable on a standard Debian or Ubuntu machine. Please provide a short documentation about the dependencies/requirements and how to run the final application.

Bonus:

The application is packaged using docker or vagrant with virtualbox.

Important Notes:

1. The task should be completed within one week. In case you need more time, please let us know.
2. Please ask if you have any questions.

Good luck!