Performance Testing roles in Software Testing, Approach, and Use Case on Web Application

Huynh
Computer Sciences Dept
Purdue University Fort Wayne
Fort Wayne, USA
huyntl02@pfw.edu

***Abstract*—An Introduction to Performance Testing. An approach to performance testing is discussed. A case study using Locust IO to perform performance testing on a blog website is presented (*Abstract*)**

***Keywords*—Software performance testing, performance testing, software testing, locust IO, automated testing** (*keywords*)

I. Introduction

A. Performance Testing Definition

Performance testing is a testing technique, performed to determine the responsiveness and stability of a software system under the various workload. Performance testing metrics are scalability, reliability, resource usage. There are four primary types of performance testing: Load Testing, Stress Testing, Soak Testing, and Spike Testing. Performance testing is non-functional testing [3].

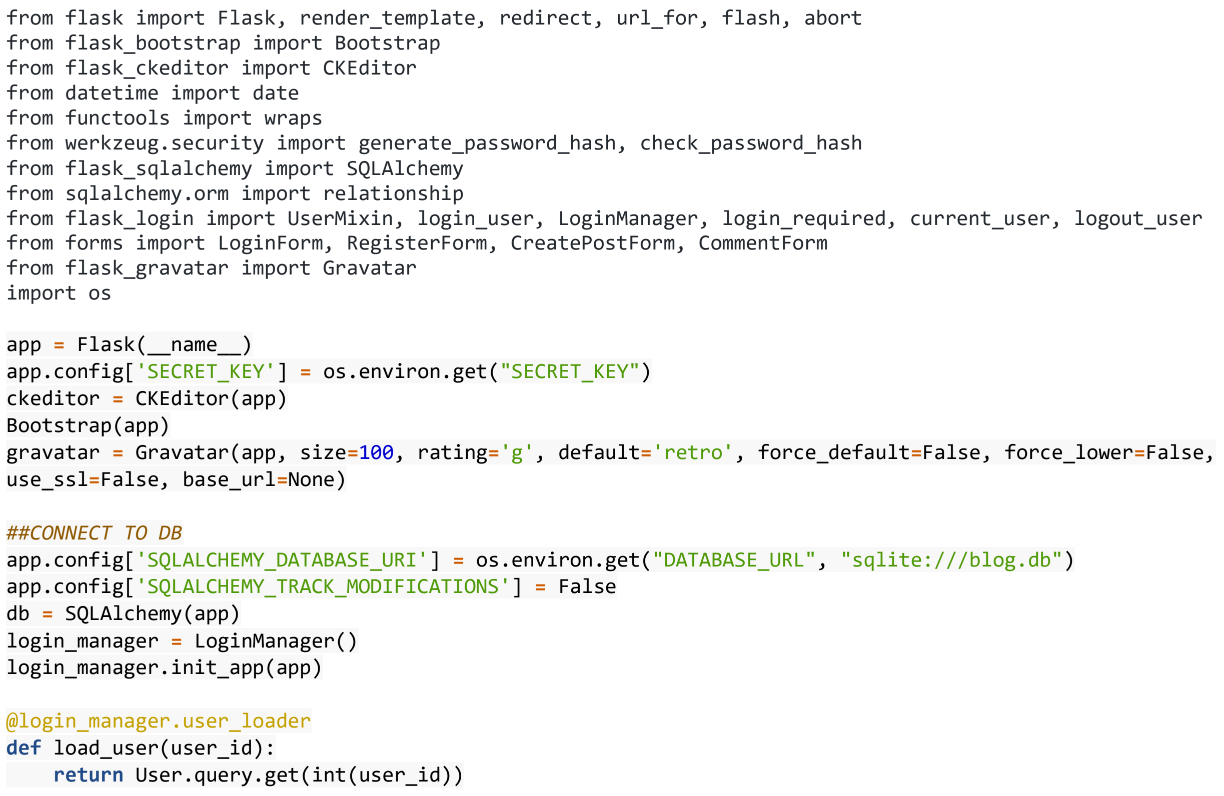
B. System Performance Degradation

According to [1], system performance degradation or problems handling required system throughput is an extremely significant issue for many large industrial projects. Although the software system has undergone extensive functionality testing, it was never tested to assess its expected performance [1].

During an architecture review at AT&T, a group of engineers has found that “performance issues account for one of the three major fault categories. Performance problems identified might include such things as the lack of performance estimates, the failure to have proposed plans for data collection, or the lack of a performance budget”. They also claimed that insufficient planning for performance issues is the major issue when deploying software to the field [1]. Major issues are issues that will impact user satisfaction.

C. Uneven Distribution of Resource Usage

Pareto-type distribution is also known as a very uneven distribution of resource usage. The main reason for system performance degradation is claimed distribution of project-affecting issues (Pareto-type distribution). According to [1], it was found that 70 percent of the most severe class of problems resided in the weakest 30 percent of the project.



```
from flask import Flask, render_template, redirect, url_for, flash, abort
from flask_bootstrap import Bootstrap
from flask_ckeditor import CKEditor
from datetime import date
from functools import wraps
from werkzeug.security import generate_password_hash, check_password_hash
from flask_sqlalchemy import SQLAlchemy
from sqlalchemy.orm import relationship
from flask_login import UserMixin, login_user, LoginManager, login_required, current_user, logout_user
from forms import LoginForm, RegisterForm, CreatePostForm, CommentForm
from flask_gravatar import Gravatar
import os

app = Flask(__name__)
app.config['SECRET_KEY'] = os.environ.get("SECRET_KEY")
ckeditor = CKEditor(app)
Bootstrap(app)
gravatar = Gravatar(app, size=100, rating='g', default='retro', force_default=False, force_lower=False,
use_ssl=False, base_url=None)

##CONNECT TO DB
app.config['SQLALCHEMY_DATABASE_URI'] = os.environ.get("DATABASE_URL", "sqlite:///blog.db")
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
db = SQLAlchemy(app)
login_manager = LoginManager()
login_manager.init_app(app)

@login_manager.user_loader
def load_user(user_id):
    return User.query.get(int(user_id))
```

Fig 1. Python Package of Blog-Template

II. Previous Work

According to [1], there is not any adequate performance testing knowledge on the Internet. there is neither an existing body of research literature nor an extensive collection of practical or experiential information available to help testers faced with the prospect of doing software performance testing [1].

III. Performance Testing Tools

A. Locust IO

Locust is an easy-to-use, scriptable and scalable performance testing tool. Instead of being constrained by a UI or domain-specific language, Locust is infinitely expandable and very developer-friendly due to its codebase tools. Locust IO is required Python programming knowledge, and it is easy to use and develop test cases.

- **Write user test scenarios in Python code:**

  If the tester wants the users to loop, perform some conditional behavior, or do some calculations. Locust runs every user inside its greenlet (a lightweight process/coroutine) which enables the tester to write tests like normal Python code (instead of using callbacks or other methods). The tester can use any Python IDE, and version control tester’s test script as regular code (as opposed to some other tools that use XML or binary formats).

- **Distributed & Scalable - supports hundreds of thousands of users:**

  Locust makes it easy to run load tests distributed over multiple machines. It is event-based which makes it possible for a single process to handle many thousands of concurrent users.

```python
@app.route('/')
def get_all_posts():
    posts = BlogPost.query.all()
    return render_template("index.html", all_posts=posts, current_user=current_user)

@app.route('/register', methods=["GET", "POST"])
def register():
    form = RegisterForm()
    # ... Code to Register ...

@app.route('/login', methods=["GET", "POST"])
def login():
    form = LoginForm()
    # ... Code to log in...

@app.route('/logout')
def logout():
    logout_user()
    # ... Code to Logout ...

@app.route("/post/<int:post_id>", methods=["GET", "POST"])
def show_post(post_id):
    form = CommentForm()
    # ... Code to show post ...

@app.route("/about")
def about():
    return render_template("about.html", current_user=current_user)

@app.route("/contact")
def contact():
    return render_template("contact.html", current_user=current_user)

@app.route("/new-post", methods=["GET", "POST"])
@admin_only
def add_new_post():
    form = CreatePostForm()
    # ... Code to add new post ...

@app.route("/edit-post/<int:post_id>", methods=["GET", "POST"])
@admin_only
def edit_post(post_id):
    post = BlogPost.query.get(post_id)
    # ... Code to edit the post ...

@app.route("/delete/<int:post_id>")
@admin_only
def delete_post(post_id):
    post_to_delete = BlogPost.query.get(post_id)
    # ... Code to delete the post ...
```

Fig 2. Python route structure of Blog-Template

- **Locust Features:**
  Running in Docker, distributed load generation, running test in a debugger, running Locust distributed with Terraform/AWS, running Locus automation, retrieved test statistics in CSV format, testing non-HTTP systems, testing request base SDKs, increase performance with a faster HTTP client, event hooks, logging, using locust as a library.

C. Expertus

Retrieved from [4], Expertus is designed to automate large-scale distributed experiment studies in IaaS clouds to address three challenges:

- **Experiment challenges:**

  Performance testing for enterprise applications consists of multiple closely related scenarios by varying a few configurable (preferably one) parameters at a time.

- **Application challenges:**

- In distributed software testing the applications should start efficiently and in the provably correct order by simultaneously enforcing serialization constraints and leveraging the distributed system's inherent parallelism.

- **Cloud challenges:** Selecting the most appropriate cloud from many cloud offerings is a non-trivial task. Also, migrating an application between two clouds is a complex, time-consuming, and error-prone task. Finally, communication, coordination, synchronization, monitoring, and complete management challenges; lastly, the dynamic nature of the cloud introduces extra complexity.

IV. USE CASE: TESTING WEBSITE (BLOG) USING LOCUST IO

In this experiment, I will also perform performance tests (automated tests) on a deployed website that was designed by Dr. Angela Yu [8] and implemented by myself.

A. Introduction

Blog-Template was originally designed by Dr. Angela Yu [8] and implemented by me. Application is a blog template that allows registered users to edit or comment on a post. Only the admin can create, edit, or delete posts. The application was hosted at heroku.com, and the hyperlink is "https://template-blog.herokuapp.com/".

The Blog-Template was designed as a template ("https://github.com/jackyhuynh/blog-template"). Anyone can use the blog for any purpose. The first user will be automatically set as the administrator of the web application. The admin will have the power to create, edit, and delete the post as well as create, edit, and delete comments. Regular users (after signing in) can create, edit, and delete comments only.

B. Requirement Analyst

The application name is Blog-Template, and it was designed using Flask Framework, and Jinja Template (Python). It's also using PostgreSQL for the database engine. The database schema is used primarily to store blog content and user information. This website is fully functional, extendable, and scalable. Code review was conducted by myself. Technologies that are implemented in the application: Flask framework, Jinja template, Object-Oriented Programming, SQL Alchemy, Password Hashing (werkzeug security), Bootstrap, CK Editor (user input), HTML, CSS, JavaScript…

In fig 1 and fig 2, I only show the template and class of each element within the application. For the full code, implementation please refer to [7]. The application connects to PostgreSQL as its main data warehouse and SQLite as its backup.

My test plan is mainly created on the route of the application. For example, the user needs to log in by using the login function of login's route (@app.route('/login', methods=["GET", "POST"])). The users can create comments by navigating to the post route, selecting the post, then adding comments (@app.route("/post/<int:post_id>", methods=

```python
from locust import HttpUser, task, between


class TestCases(HttpUser):
    host = "https://template-blog.herokuapp.com/"
    # Set the wait time for each user from 1 to 300 seconds
    # So that each user will be able to navigate and read 1 post or do some comment
    wait_time = between(1, 300)

    # When a user in login the user
    def on_start(self):
        self.client.post("/login", json={"Email": "jackyhuynh87@gmail.com", "Password":"1234"})

    # task number 1 is to navigate to the contact page (and stay there for 40 seconds)
    @task(10)
    def visit_contact(self):
        self.client.get("/contact")

    # after 50 seconds navigate to post 1
    @task(50)
    def visit_post_1(self):
        self. client.get("/post/1")
```

Fig 4. Locust Performance Test Script for Blog Template (locust_test.py)

["GET","POST"])) (fig 2). All tasks can be scheduled with Locus IO.

The unit testing of the applications has been tested using the Python Unittest package (fig 2). Each unit in the program has passed all the tests. The program develops incremental (bottom-up). After conducting all the unit tests, the application goes through an integration test. Before conducting performance testing, the application must pass all unit test cases and integration tests.

C. Structure of Backend

main.py is the python script containing all components required to run the application (Blog Template). The "main.py" consists of all required packages and the route for the front end. The front-end was built with HTML, CSS, and Bootstrap, and stored in the templates folder (about.html, contact.html, footer.html, header.html, index.html, login.html, make-post.html, post.html, register.html) [7].

The static folder contains all the .css files, images, javascript. The package (folders containing main.py and all the necessary files) is then pushed through GitHub and deployed to Heroku from GitHub. Therefore, every time I make a change, the application will automatically change. I also schedule automatic testings (mostly functional testing and GUI testing) for the app every time I make a change (just to make sure nothing will break when I make a new commit). PyCharm IDE was used to develop this code, but the user can use any IDE to navigate the code.

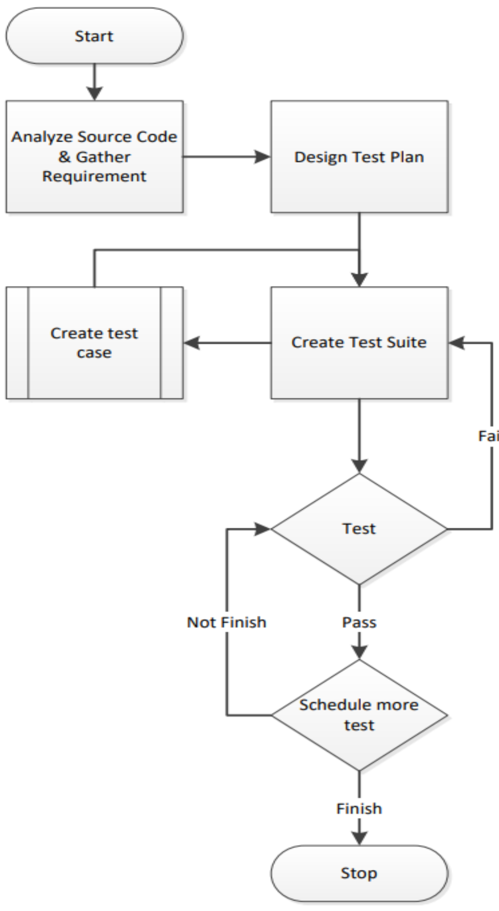


Fig 3. Tets Suite Design Process

B. Test Plan Design

After analyzing Blog-Template Source Code and requirements, I design my test plan. My plan is simple to increase the number of users over time and stop at 3 million users (fig 3). I will only perform 3 tasks on the websites: login, visit the contact page and visit post number 1. Even though, Locust IO allows me with much more tests such as creating, editing, or deleting tasks; creating new users; importing new documentation into user posts… However, I just want to keep my tests simple for this use case. I can also write custom scripts such as creating, updating, and deleting posts simultaneously to make sure the app is full capacity, but that will complicate my demonstration and not the point I want to deliver today. In fact, Locus IO provides testers with the tools to accomplish all these tasks that I mentioned in a few lines of code (fig 3).

I will run an automated test starting at 1000 users at a spawn of 1.00 seconds then I will increase by 1000 for each test case. There are 2 scripts that I need to design: the performance test script and the schedule test script (fig 5).

C. Test Suite Implementation

Plan:

Run the test script (or use another script to schedule the test script). The schedule script should be able to increase the number of users after each successful test. The process is described below. Please refer to fig 3 for the test plan diagram and fig 5 for the test suite implementation process diagram.

Test Script: Set the wait time for each user from 1 to 300 seconds so that each user will be able to navigate and read 1 post or do some comments. Task number 1 is to navigate to the contact page (and stay there for 40 seconds). After 50 seconds navigate to post 1. PyCharm IDE is used to

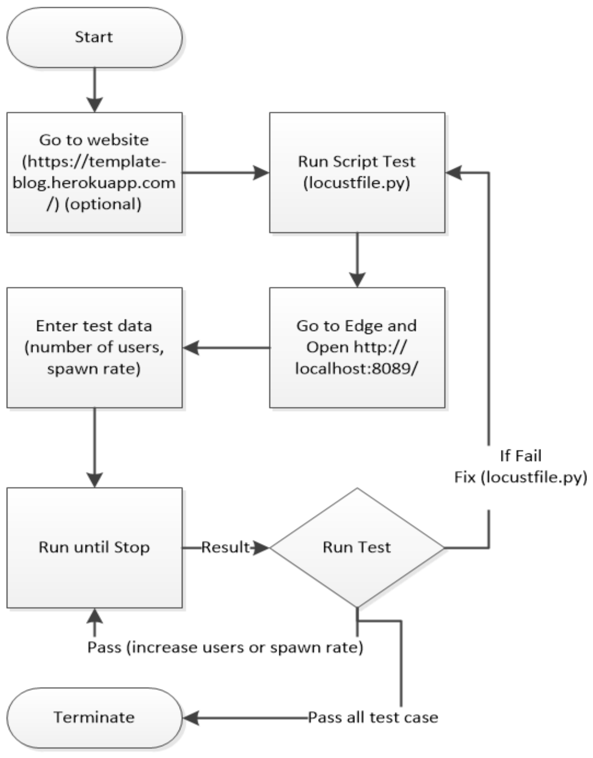

Start
Go to website (https://template-blog.herokuapp.com/) (optional)
Run Script Test (locustfile.py)
Enter test data (number of users, spawn rate)
Go to Edge and Open http://localhost:8089/
If Fail Fix (locustfile.py)
Run until Stop
Result
Run Test
Pass (increase users or spawn rate)
Terminate
Pass all test case

Fig 5. Tets Suite Implementation Process

run the code. This is the content of the test script (locust_test.py). The detailed content can be found in fig 4.

Execute:

Run this script within the Terminal of PyCharm IDE:

“locust -f locust_test.py”.

Then go to “http://localhost:8089/” for monitoring.

Automation Testing:

For automation testing without the WEB UI, run this script instead:

“locust -f locust_test.py -u 1000 -r 100 --run-time 1h30m --stop-timeout 99.”

To schedule a full test suite, we need to write another Python script that increments the users as our test plan and simply schedules it. Depend on the application report on peak time (e.g., 2 million people operate at the same time). The tester can simply set the user to that peak and schedule the performance test when they have a change in the system (or just for regular maintenance).

Call this: “python3 schedule-script.py”. Schedule script (schedule-script.py) can be used to scan and detect code changes. After the scan, the schedule script will then decide if a performance test is needed. If needed, it will call the “locust_test.py”. Please note that the content of “schedule-script.py”

C. Result

Plan: Run the test script (or use another script to schedule the test script). The schedule script should be able to increase the number of users after each successful test. The process is described below Result for the first test case is 1000 users at a 1.00 spawn rate. The result returned with a 0% failure rate. Testers can download the PDF, CVS version for data analysis. Web UI can also be used for monitoring with real-time monitoring and interactive chart.

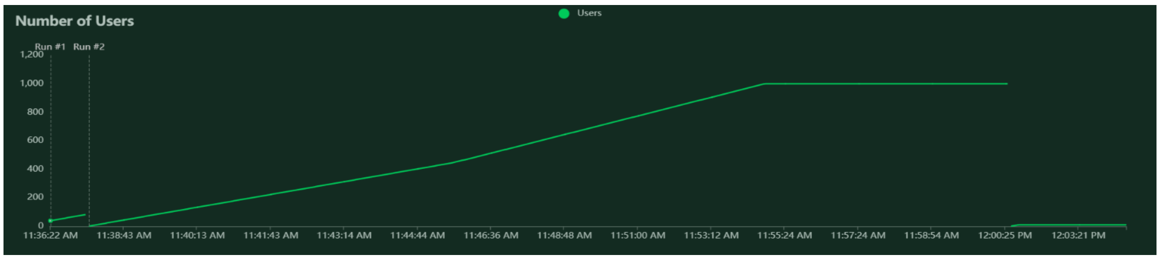


Fig 7. The number of User Chart. This is the WEB UI chart (1000 users at 1.00 spawn rate)

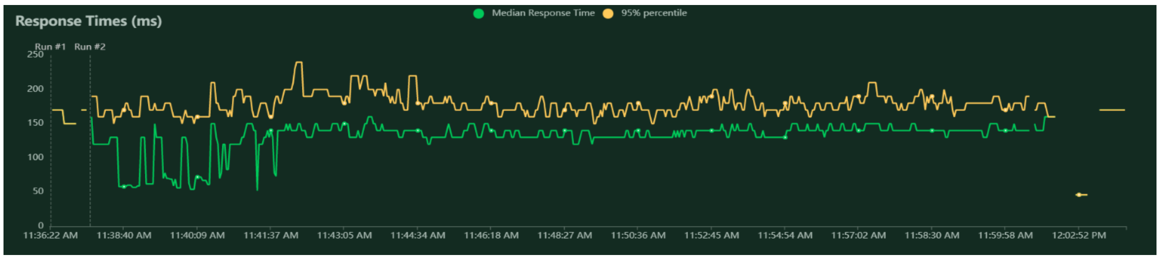


Fig 8. The number of Response Time. This is the WEB UI chart (1000 users at 1.00 spawn rate)

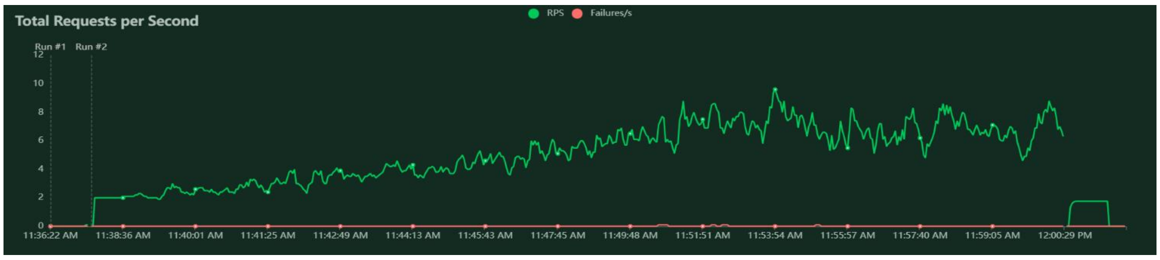
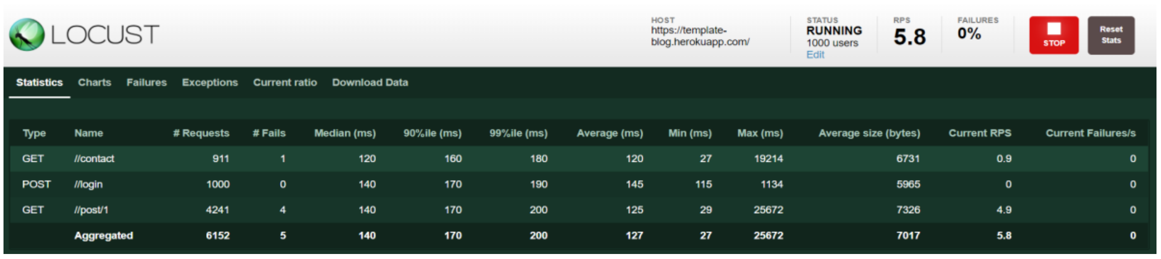


Fig 9. Total Request per second. This is the WEB UI chart (1000 users at 1.00 spawn rate)



Type	Name	# Requests	# Fails	Median (ms)	90%ile (ms)	99%ile (ms)	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	Current RPS	Current Failures/s
GET	//contact	911	1	120	160	180	120	27	19214	6731	0.9	0
POST	//login	1000	0	140	170	190	145	115	1134	5965	0	0
GET	//post/1	4241	4	140	170	200	125	29	25672	7326	4.9	0
	Aggregated	6152	5	140	170	200	127	27	25672	7017	5.8	0

Fig 10. Locus Web UI

In my latest test, Template-Blog was able to handle 20,000 users at a 0.5 spawn rate.

Test Result: This is printed within my IDE (Py Charm) in Fig 6. More results can be reviewed or downloaded at "http://localhost:8089/". I include some screenshots of monitoring live at localhost (Fig 7, fig 8, fig 9). As I mentioned above, testers can also export the result into a .csv spreadsheet for statical analysis.

V. Advantages And Disadvantages Of Locust

A. Advantages:

Verifies the speed, accuracy, and stability of the software match expectation. Assists the system by authenticating the responsiveness and managing the scalability and reliability of software features. It is free and open source.

B. Disadvantages:

Locust can be a costly mistake if done haphazardly, leading to inaccurate results and conclusions. Need Python programming knowledge to perform the test (must be very familiar with Python to design complicated test cases).

```
2022-03-10T17:00:41Z
[2022-03-10 12:00:41,483] DESKTOP-J4K30PB/INFO/locust.main: Shutting down (exit code 1)
 Name                                                                                  # reqs      # fails  |     Avg
    Min     Max  Median  |   req/s failures/s
--------------------------------------------------------------------------------------------------------------------
----------------------------------------------
 GET //contact                                                                           1095     1(0.09%)  |     120
     27   19214     130  |    0.81    0.00
 POST //login                                                                            1000     0(0.00%)  |     144
    114    1134     140  |    0.74    0.00
 GET //post/1                                                                            5162     4(0.08%)  |     125
     28   25672     140  |    3.83    0.00
--------------------------------------------------------------------------------------------------------------------
----------------------------------------------
--------------------------------------------------------------------------------------------------------------------
----------------------------------------------
 2                   GET //post/1: ConnectionResetError(10054, 'An existing connection was forcibly closed by the r
emote host', None, 10054, None)
 2                   GET //post/1: RemoteDisconnected('Remote end closed connection without response')
 1                   GET //contact: ConnectionResetError(10054, 'An existing connection was forcibly closed by the
remote host', None, 10054, None)
--------------------------------------------------------------------------------------------------------------------
----------------------------------------------
 Name                                                                                  # reqs      # fails  |     Avg
    Min     Max  Median  |   req/s failures/s
--------------------------------------------------------------------------------------------------------------------
----------------------------------------------
 GET //contact                                                                              1     0(0.00%)  |      42
     42      42      42  |    0.01    0.00
 POST //login                                                                              10     0(0.00%)  |     162
    137     181     160  |    0.05    0.00
 GET //post/1                                                                              18     0(0.00%)  |     106
     40     194      66  |    0.09    0.00
--------------------------------------------------------------------------------------------------------------------
----------------------------------------------
 Aggregated                                                                                29     0(0.00%)  |     123
     40     194     150  |    0.15    0.00

Response time percentiles (approximated)
 Type     Name                                                                                    50%    66%    75%
   80%    90%    95%    98%    99%  99.9% 99.99%   100% # reqs
--------|--------------------------------------------------------------------------------|---------|------|------|
------|------|------|------|------|------|------|------|------|
 GET      //contact                                                                                42     42     42
    42     42     42     42     42     42     42     42      1
 POST     //login                                                                                 170    170    170
   170    180    180    180    180    180    180    180     10
 GET      //post/1                                                                                 69    170    170
   170    180    190    190    190    190    190    190     18
--------|--------------------------------------------------------------------------------|---------|------|------|
------|------|------|------|------|------|------|------|------|
 None     Aggregated                                                                              150    170    170
   170    180    180    190    190    190    190    190     29
```

Fig 6. Test Result is printed within Py Charm IDE

REFERENCES

[1] E. J. Weyuker and F. I. Vokolos, "Experience with performance testing of software systems: issues, an approach, and case study," in IEEE Transactions on Software Engineering, vol. 26, no. 12, pp. 1147-1156, Dec. 2000, DOI: 10.1109/32.888628. *(references)*

[2] Kumar. K., "Write your first performance/load test in Python.", Retrieved from https://medium.com/@kundan3034/write-your-first-performance-load-test-in-python-e8e2132ef775.

[3] Tutorial Points (n.d.), "Performance Testing", Retrieved from https://www.tutorialspoint.com/software_testing_dictionary/performance_testing.htm.

[4] D. Jayasinghe et al., "Expertus: A Generator Approach to Automate Performance Testing in IaaS Clouds," 2012 IEEE Fifth International Conference on Cloud Computing, 2012, pp. 115-122, DOI: 10.1109/CLOUD.2012.98.K.

[5] Bystron. C., Heyman. J., Hamren J., Heyman. H. and Holmberg. L., Locust, Retrieved from https://github.com/locustio/locust.

[6] Y. Yorozu, M. Hirano, K. Oka, and Y. Tagawa, "Electron spectroscopy studies on magneto-optical media and plastic substrate interface," IEEE Transl. J. Magn. Japan, vol. 2, pp. 740–741, August 1987 [Digests 9th Annual Conf. Magnetics Japan, p. 301, 1982].

[7] T. Huynh, Blog Template, retrieved from jackyhuynh/blog-template: a simple blog template that you can post anything. (github.com)

[8] A. Yu, 100 Days of Code: The Complete Python Pro Bootcamp for 2022, Retrieved from 100 Days of Code: The Complete Python Pro Bootcamp for 2022 | Udemy