**P.E.S. UNIVERSITY**

**Department of Computer Science and Engineering**

**Session: Jan-May 2020**

UE17CS355 – Web Technologies-II Lab

Project Phase – II

# Test Report

Project Title: *BlogStop - For bloggers*

Section: 6A

Team Members:

Vishal Sathyanarayana (PES1201700183)

Hritvik Patel (PES1201700125)

Sparsha P (PES1201700226)

Bhargavi Priyasha Kumar (PES1201701802)

# Unit Testing

*-Vishal Sathyanarayana(PES1201700183)*

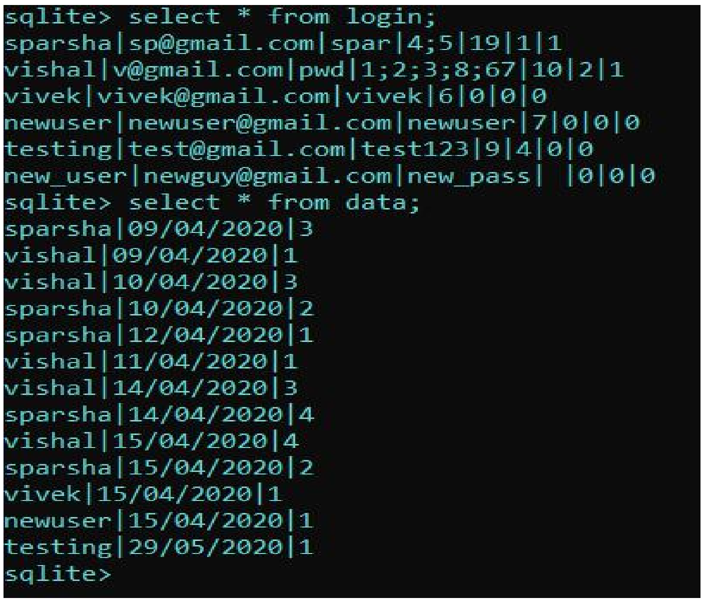
## 1. Introduction

The BlogStop web application uses REST APIs to communicate between the front end and the backend Python Flask server.

Unit testing is a level of software testing where individual units/ components of a software are tested. The purpose is to validate that each unit of the software performs its functionality as designed. It usually has one or multiple inputs and a single output. This is helpful to check if the rest of the application still works when adding or deleting functionalities.

## 2. Testing Details

Each Unit test tests a single Flask API. Each API serves a particular task. We have used the "unittest" python library to perform our testing on our web application. We basically check for response status codes and expected response data with the use of this library. The initial contents of our database are as follows:



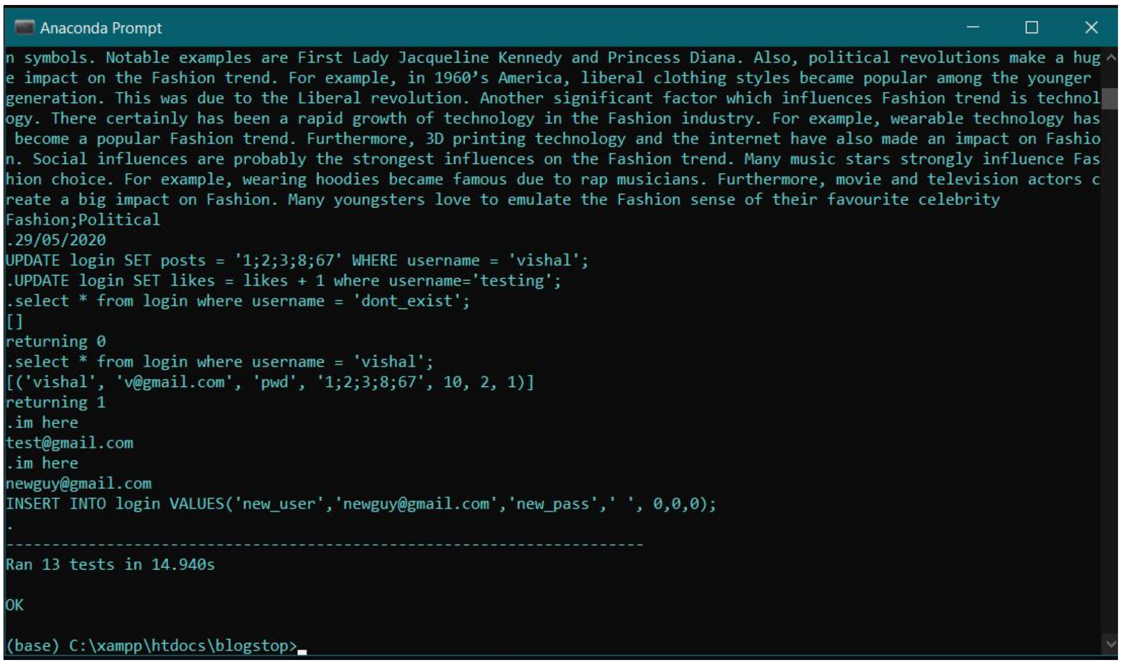
```
sqlite> select * from login;
sparsha|sp@gmail.com|spar|4;5|19|1|1
vishal|v@gmail.com|pwd|1;2;3;8;67|10|2|1
vivek|vivek@gmail.com|vivek|6|0|0|0
newuser|newuser@gmail.com|newuser|7|0|0|0
testing|test@gmail.com|test123|9|4|0|0
new_user|newguy@gmail.com|new_pass| |0|0|0
sqlite> select * from data;
sparsha|09/04/2020|3
vishal|09/04/2020|1
vishal|10/04/2020|3
sparsha|10/04/2020|2
sparsha|12/04/2020|1
vishal|11/04/2020|1
vishal|14/04/2020|3
sparsha|14/04/2020|4
vishal|15/04/2020|4
sparsha|15/04/2020|2
vivek|15/04/2020|1
newuser|15/04/2020|1
testing|29/05/2020|1
sqlite>
```

**3. Test Report**

| Test Case No. | Unit Test Description | Expected Output | Pass / Fail |
|---|---|---|---|
| 1. | Register user (success): The username, email, password needs to be added to the database | Empty dictionary with status code 200 OK. | Pass |
| 2. | Register User (fail): In case of duplicate username/ email | Dictionary as {“returned”:0} with status code 200 OK. | Pass |
| 3. | User login (success): This API tells us if a username, password combination exists in the login database. This is the test case where it is valid. | Dictionary as {"returned":1} with status code 200 OK. | Pass |
| 4. | User login (fail): This API tells us if a username, password combination exists in the login database. This is the test case where it is invalid. | Dictionary as {"returned":0} with status code 200 OK. | Pass |
| 5. | Check user (success): To check if user is valid or not. This is the test case where user is valid. | Dictionary as {"returned":1} with status code 200 OK. | Pass |
| 6. | Check user (fail): To check if user is valid or not. This is the test case where user is invalid. | Dictionary as {"returned":0} with status code 200 OK. | Pass |
| 7. | Recommend tag: This API recommends 2 tags given a string input that is most suitable for the aforementioned | Tags such as Sports, Fitness, Politics etc are present. The 2 most relevant one’s are selected and output in the form: tag1; tag2. With | Pass |

| | | | |
|---|---|---|---|
| | test. | status code 200 OK. | |
| 8. | Retrieve profile details: This API retrieves all the information of a given user from the database. | The respective user’s Email, Total number of likes, Number of posts, post ids and username are present in the response body with status code 200 OK. | Pass |
| 9. | Update post: Adds the post for that respective user along with the date of posting into the database. | Empty dictionary as the response data along with 200 OK status code. | Pass |
| 10. | Update likes: This API updates the number of likes for a given user | Empty dictionary as the response data along with 200 OK status code. | Pass |
| 11. | Positive and Negative reviews: This is an API which performs sentiment analysis on all the comments on the respective user’s profile. | Response body has a dictionary with: number of positive and negative reviews along with a 200 OK status code. | Pass |
| 12. | Get statistics: This API call gets the statistics for a user. It helps track the number of posts posted by the user on a daily basis. | Empty dictionary as the response data along with 200 OK status code. | Pass |

## 4. Observations and Conclusions



```
n symbols. Notable examples are First Lady Jacqueline Kennedy and Princess Diana. Also, political revolutions make a hug
e impact on the Fashion trend. For example, in 1960's America, liberal clothing styles became popular among the younger
generation. This was due to the Liberal revolution. Another significant factor which influences Fashion trend is technol
ogy. There certainly has been a rapid growth of technology in the Fashion industry. For example, wearable technology has
 become a popular Fashion trend. Furthermore, 3D printing technology and the internet have also made an impact on Fashio
n. Social influences are probably the strongest influences on the Fashion trend. Many music stars strongly influence Fas
hion choice. For example, wearing hoodies became famous due to rap musicians. Furthermore, movie and television actors c
reate a big impact on Fashion. Many youngsters love to emulate the Fashion sense of their favourite celebrity
Fashion;Political
.29/05/2020
UPDATE login SET posts = '1;2;3;8;67' WHERE username = 'vishal';
.UPDATE login SET likes = likes + 1 where username='testing';
.select * from login where username = 'dont_exist';
[]
returning 0
.select * from login where username = 'vishal';
[('vishal', 'v@gmail.com', 'pwd', '1;2;3;8;67', 10, 2, 1)]
returning 1
.im here
test@gmail.com
.im here
newguy@gmail.com
INSERT INTO login VALUES('new_user','newguy@gmail.com','new_pass',' ', 0,0,0);
.
----------------------------------------------------------------------
Ran 13 tests in 14.940s

OK

(base) C:\xampp\htdocs\blogstop>
```

The 13-unit test cases were run in the same order as shown in the table above. The 13th test case is another instance of testing for the Recommend tags API. These test cases took 14.94 seconds to complete. All API tests have successfully passed.

This proves that all the individual components and APIs of our web application are working perfectly.

## Load Testing

*Hritvik Patel [PES1201700125]*

### 1. Introduction

Our project’s objective was to create a web application for bloggers. Its features include: registering of users, sign in, uploading new posts, liking other writers' posts, viewing the statistics of the logged in user, auto-recommendation of tags, detecting the expression on the comments posted to each blog post. I will be performing load testing. I used Postman collections and k6 to perform tests on this application.

### 2. Objective

The objective of this type of testing is to find out the load our application can take. The most important testing that is performed before an application is put for production use is load testing. Load testing will just give us the statistics of how much load the application can take before failing or become non-responsive.

### 3. Test Report

| Test case No. | Test Case Description [Test is performed with the following parameters] | Expected Output | Actual Output | Test Result (Pass/Fail) |
|---|---|---|---|---|
| 1 | Users = 1; Duration = 10 seconds; | All API endpoints work | All API endpoints work | PASS |
| 2 | Users = 1; Duration = 20 seconds; | All API endpoints work | All API endpoints work | PASS |
| 3 | Users = 1; Duration = 30 seconds; | All API endpoints work | All API endpoints work | PASS |
| 4 | Users = 1; Duration = 40 seconds; | All API endpoints work | All API endpoints work | PASS |
| 5 | Users = 1; | All API | All API | PASS |

| | | | | |
|---|---|---|---|---|
| | Duration = 50 seconds; | endpoints work | endpoints work | |
| 6 | Users = 1; Duration = 1 minute | All API endpoints work | All API endpoints work | PASS |
| 7 | Users = 10; Duration = 10 seconds; | All API endpoints work | Like update API fails<br>Register API fails | FAIL<br>*Reason: As a large number of requests flood the endpoint, there are database locking errors.* |
| 8 | Users = 10; Duration = 20 seconds; | All API endpoints work | Like update API fails<br>Register API fails | FAIL<br>*Reason: As a large number of requests flood the endpoint, there are database locking errors.* |
| 9 | Users = 10; Duration = 30 seconds; | All API endpoints work | Like update API fails<br>Register API fails | FAIL<br>*Reason: As a large number of requests flood the endpoint, there are database locking errors.* |
| 10 | Users = 10; Duration = 40 seconds; | All API endpoints work | Like update API fails<br>Register API fails | FAIL<br>*Reason: As a large number of requests flood the endpoint, there are database locking errors.* |
| 11 | Users = 10; Duration = 50 seconds; | All API endpoints work | Like update API fails<br>Register API fails | FAIL<br>*Reason: As a large number of requests flood* |

| | | | | |
|---|---|---|---|---|
| | | | | *the endpoint, there are database locking errors.* |
| 12 | Users = 10; Duration = 1 minute; | All API endpoints work | Like update API fails<br>Register API fails | FAIL<br>*Reason: As a large number of requests flood the endpoint, there are database locking errors.* |

## 4. Observation and Conclusion

What we can observe from these tests are, the application works as expected when there is only one user, but when multiple users make requests which modify the database in some way, we get unexpected results. It doesn't mean that all the requests fail to that endpoint, some of them such as the first request to that endpoint returns successfully. Only the requests which come during the time when the first request is being processed fail.

The following conclusion would be to solve the database lockup by using a distributed database or write all the requests which are pending to a queue and then the database is made to read the queue and perform all the operations. The drawback of the second approach, although easy to implement is that the later requests have to wait till their requests are processed and this would significantly increase the response time. The first approach, although tricky to implement, solves all these issues as we have multiple copies of the database and the databases can be synced at every fixed time interval so that data is consistent across all databases.

The load testing was performed using Postman collection and k6.

The following results are excerpts when load testing was performed:

---------------------------- ***10s 1 user*** ------------------------------------

```
data_received.............: 513 kB 51 kB/s

data_sent.................: 118 kB 12 kB/s

http_req_blocked...........:   avg=610.37µs   min=0s                  med=986.2µs   max=2.06ms
p(90)=1.01ms   p(95)=1.08ms
```

http_req_connecting........: avg=492.14µs min=0s      med=0s     max=2.06ms   p(90)=1.01ms p(95)=1.02ms

**http_req_duration..........:  avg=13.29ms     min=0s                med=12.99ms   max=78.6ms p(90)=26.11ms  p(95)=30.99ms**

http_req_receiving.........: avg=304.53µs min=0s        med=0s       max=1.49ms    p(90)=1ms p(95)=1ms

http_req_sending...........: avg=107.02µs min=0s       med=0s      max=1.35ms   p(90)=959.7µs p(95)=1ms

http_req_waiting...........:   avg=12.88ms      min=0s                 med=12.66ms   max=78.6ms p(90)=25.79ms  p(95)=30.27ms

http_reqs..................: 685    68.496754/s

iteration_duration.........:   avg=334.45ms   min=268.23ms   med=332ms        max=454.76ms p(90)=388.57ms p(95)=397.27ms

iterations.................: 29     2.899863/s

--------------------------- ***20s 1 user*** ------------------------------------

data_received..............: 726 kB 36 kB/s

data_sent..................: 167 kB 8.3 kB/s

http_req_blocked...........:   avg=650.2µs     min=0s               med=986.8µs     max=15.09ms p(90)=1.03ms   p(95)=1.14ms

http_req_connecting........:   avg=520.21µs   min=0s             med=0s              max=14.01ms p(90)=1.01ms   p(95)=1.04ms

**http_req_duration..........:  avg=19.46ms     min=0s              med=17.67ms     max=98.72ms p(90)=37.72ms  p(95)=46.12ms**

http_req_receiving.........: avg=324.84µs min=0s        med=0s        max=2.01ms    p(90)=1ms p(95)=1.02ms

http_req_sending...........: avg=124.6µs  min=0s       med=0s       max=1.24ms   p(90)=990.2µs p(95)=1ms

http_req_waiting...........:   avg=19.01ms     min=0s              med=17.15ms     max=98.72ms p(90)=36.99ms  p(95)=45.77ms

http_reqs..................: 961    48.049004/s

```
iteration_duration.........: avg=478.12ms min=274.46ms med=474.81ms max=856.78ms p(90)=600.54ms p(95)=650.18ms

iterations.................: 41     2.049958/s
```

--------------------------- ***30s 1 user*** ------------------------------------

```
data_received..............: 992 kB 33 kB/s

data_sent..................: 228 kB 7.6 kB/s

http_req_blocked...........: avg=694.17µs min=0s med=994.2µs max=15ms p(90)=1.02ms p(95)=1.28ms

http_req_connecting........: avg=548.59µs min=0s med=968.55µs max=13.99ms p(90)=1.01ms p(95)=1.02ms

http_req_duration..........: avg=21.58ms min=0s med=19ms max=93.01ms p(90)=43.98ms p(95)=52.97ms

http_req_receiving.........: avg=297.51µs min=0s med=0s max=7ms p(90)=1ms p(95)=1ms

http_req_sending...........: avg=105.44µs min=0s med=0s max=1.06ms p(90)=956.84µs p(95)=1ms

http_req_waiting...........: avg=21.18ms min=0s med=18.99ms max=91.99ms p(90)=43.12ms p(95)=52.08ms

http_reqs..................: 1302   43.398947/s

iteration_duration.........: avg=528.92ms min=251.22ms med=482.32ms max=870.76ms p(90)=788.16ms p(95)=816.51ms

iterations.................: 56     1.866621/s
```

All we need to focus on from the excerpts is the value of “http_req_duration” as this gives us the average response time. The trend is clear that the response time increases linearly when the load is increased linearly.

# System Testing

*-- Sparsha P. (PES1201700226)*

## 1. Introduction

The main objective of Blogspot is to provide users with an interactive blogging experience. It manages all information about the blog itself, its categories while also providing features such as like and comment to express your opinion freely and anonymously. We also have intelligence functionalities such as sentiment analysis and automatic tagging to improve user experience. The user can access his blog statistics which show with the help of graphs how well his posts are being received by the community as well as his trend of uploading posts.

The type of testing being performed is system testing. We are trying to perform automated testing using Selenium with python.

## 2. Objective

System testing is a level of testing that validates the complete and fully integrated software product. The purpose of a system test is to evaluate the end-to-end system specifications. The test involves the external workings of the software from the user's perspective.

- Testing the fully integrated applications to check how components interact with one another and with the system as a whole.
- Verify thorough testing of every input in the application to check for desired outputs.
- Testing of the user's experience with the application.

## 3. Test Report

Testing is done with the help of selenium in python.
For testing, an automated test script for each test case is created. Using the chrome driver, the execution of the test can be viewed on the browser which is controlled by selenium. The results of the test can be determined if results of executed test cases are as expected.

| Test case No. | Test case Description | Expected output | Actual output | Test Result (Pass/Fail) |
|---|---|---|---|---|
| 1. | To check the working of all links in the website. | All links navigate to the correct pages | All links navigate to the correct pages. | Pass |
| 2. | To check the working of likes button on posts. Every time the button is clicked, the number of likes for the respective post should match the number of times the button is clicked. | On clicking the likes button once, the number of likes is incremented by 1. | Number of likes for the respective post is incremented by 1. | Pass |
| 3. | To check the working of comments option for posts.<br>On posting a new comment for a post, this comment needs to show up the particular post. | On typing a new comment for a post and clicking the comment button, the comment should be posted. | The comment for the respective post is posted. | Pass |
| 4. | To check if the plots for blog statistics are rendered for the user logged in. The plots for a user are<br>1) sentiment analysis for comments<br>2) Trend on how often you post | On clicking the plots button in my profile, the statistics about my blog posts need to show up. | The blog posts for my profile render correctly. | Pass |
| 5. | To check if the recommend tags is working as expected. Taking the post content as the input, it needs to recommend suitable tags for it. | For the given post, it needs to recommend sports and fitness tags. | It recommends sports and fitness tags. | Pass |
| 6. | To check the working of user suggestions when searching for a particular user. With the help of the suggestions, should be able to view the profile of the user as well. | On typing the character “v”, all the usernames starting with v need to be suggested.<br>On searching for that user, their profile needs to show up. | On typing “v”, all usernames starting with v are suggested and their profile shows up on search | Pass |

| | | | | |
|---|---|---|---|---|
| 7. | To check the option of submitting new posts with all the details filled. The new post needs to show up in the user's profile. | On blogging a new post, the post needs to shows up on their profile | The new post des show up on the user's profile | Pass |
| 8. | To check the working of plots (sentiment analysis for comments) for a new user. A new user who puts up his first post and hasn't received any comments. | If a new user who hasn't received any comments should be able to check is blog statistics. | The plots in case aren't rendered and throws an error. | Fail |
| 9. | To check if a user can add a new post without any tags for a post. Tags just provide context regarding the post. | User should be able to add posts without any tags. | User is able to post with any tags. | Pass |
| 10. | To check if the sentiment analysis for comments works as expected. If I comment on a post of a user and check their blog statistics, the new comment should be considered for sentiment analysis. | The plot needs to show the correct number of positive and negative comments. | The plot shows the correct nuber of positive and negative comments. | Pass |

## 4. Observation and Conclusion

90% of the test cases have passed. The application works as per the requirements. The website does not cause any considerable issues to the user. The user experience of the website is good.

# VULNERABLITY TESTING

By: Bhargavi Priyasha Kumar (PES1201701802)

## 1. Introduction

BlogStop is an interactive blogging web application. Vulnerability testing will be performed on the application manually.

## 2. Objective

Vulnerability Testing detects and classifies security loopholes (vulnerabilities) in the infrastructure of the application. It tests against security attacks such as SQL Injection, XSS and CSRF.

## 3. Test Report

| Test case No. | Test Case Description | Expected Output | Actual Output | Test Result (Pass/Fail) |
|---|---|---|---|---|
| 1. | To test if SQL insert query can be used to insert entry into database | Entry should not be added to database | Values passed in query are added to login table in database | Fail |
| 2. | To test if SQL delete query can be used to delete entry from database | Entry should not be deleted from table in database | All entries are deleted from login table in database | Fail |
| 3. | To test if SQL drop query can be used to drop tables from database | Table should not be dropped from database | Login table is dropped from database | Fail |
| 4. | To test if SQL insert query can be used to insert entry into database | Entry should not be added to database | Values passed in query are added to login table in database | Fail |
| 5. | To test if SQL delete query can be used to delete entry from database | Entry should not be deleted from table in database | All entries are deleted from login table in database | Fail |
| 6. | To test if SQL drop query can be used to drop | Table should not be dropped from database | Login table is dropped from database | Fail |

| | | | | |
|---|---|---|---|---|
| | tables from database | | | |
| 7. | Stored XSS attack to post image into DOM | Inserted script should be sanitized and displayed as plain text | Script is executed and Image is posted | Fail |
| 8. | Stored XSS attack to post GIF into the DOM | Inserted script should be sanitized and displayed as plain text | Script is executed and GIF is posted | Fail |
| 9. | Stored XSS attack to post hyperlinks into DOM | Inserted script should be sanitized and displayed as plain text | Script is executed and hyperlink is posted | Fail |
| 10. | Stored XSS attack to post form into DOM | Inserted script should be sanitized and displayed as plain text | Script is executed and form is posted | Fail |

## 4. Observation and Conclusion

The web application BlogStop is highly vulnerable to security attacks such as SQL injection and XSS. SQL injection can be prevented by taking measures such as prepared statements, parameterized queries or stored procedures instead of converting user inputs to SQL queries. XSS attacks can be prevented by filtering inputs, encoding outputs and using appropriate response headers.