

**P.E.S. UNIVERSITY**

**Department of Computer Scince and Engineering**

**Session: Jan-May 2020**

UE17CS355 – Web Technologies-II Lab

Project Phase – II

**Test Report**

Project Title: SanAdiUl - A Career Networking Website

Section: 6G

Team Members:

Sanjay Chari PES1201700278

Aditya Shankaran PES1201700710

Athul Sandosh PES1201701110

Unit Testing

By: Sanjay Chari PES1201700278

1. Introduction

The application is a career networking website built with HTML, CSS, Javascript, Bootstrap and jQuery on the front end; and Flask and MySQL on the backend. For unit testing, we created scripts in unit testing using the unittest module. We called our API endpoints through the requests module in each test case that was part of the unittest main loop.

1. Objective

In unit testing, we are trying to observe if all the backend APIs of our application return the expected response when provided with a given JSON request. Basically, unit testing checks that the backend portion of the application returns the expected output for all test cases, including corner cases.

1. Test Report

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test case No. | Test Case Description | Expected Output | Actual Output | Test Result (Pass/Fail) |
| 1 | Test Add User(Password in Wrong Format),  Method : POST  Request Type : JSON  URL Arguments : No | Password in Wrong Format | Password in Wrong Format | Pass |
| 2 | Test Add User(Password and Repeat Password do not match),  Method : POST  Request Type : JSON  URL Arguments : No | Password and Repeat Password do not match | Password and Repeat Password do not match | Pass |
| 3 | Test Add User(Email Already Registered),  Method : POST  Request Type : JSON  URL Arguments : No | Email Already Registered | Email Already Registered | Pass |
| 4 | Test Add User(Profile Created Successfully),  Method : POST  Request Type : JSON  URL Arguments : No | Profile Created Successfully | Profile Created Successfully | Pass |
| 5 | Test Login(Password in Wrong Format),  Method : POST  Request Type : JSON  URL Arguments : No | Password in Wrong Format | Password in Wrong Format | Pass |
| 6 | Test Login(Email Not Registered),  Method : POST  Request Type : JSON  URL Arguments : No | Email Not Registered | Email Not Registered | Pass |
| 7 | Test Login(Invalid Password),  Method : POST  Request Type : JSON  URL Arguments : No | Invalid Password | Invalid Password | Pass |
| 8 | Test Login(Login Successful),  Method : POST  Request Type : JSON  URL Arguments : No | Login Successful | Login Successful | Pass |
| 9 | Test Add Job,  Method : POST  Request Type : JSON  URL Arguments : No | Job Created Successfully | Job Created Successfully | Pass |
| 10 | Test List Jobs,  Method : GET  Request Type : None  URL Arguments : Yes | HTML response containing <div class=\"row align-items-start job-item border-bottom pb-3 mb-3 pt-3\"> | HTML response containing <div class=\"row align-items-start job-item border-bottom pb-3 mb-3 pt-3\"> | Pass |
| 11 | Test Add Connections(Invalid Email),  Method : POST  Request Type : JSON  URL Arguments : No | Invalid Email | Invalid Email | Pass |
| 12 | Test Add Connections(Connection Request Sent),  Method : POST  Request Type : JSON  URL Arguments : No | Connection Request Sent | Connection Request Sent | Pass |
| 13 | Test Update Connections,  Method : POST  Request Type : JSON  URL Arguments : No | Request Approved | Request Approved | Pass |
| 14 | Test List Connections(List All),  Method : GET  Request Type : None  URL Arguments : Yes | HTML response containing <div class=\"row mb-5 justify-content-center\"> | HTML response containing <div class=\"row mb-5 justify-content-center\"> | Pass |
| 15 | Test List Connections That Match with the search tag,  Method : GET  Request Type : None  URL Arguments : Yes | HTML response containing <div class=\"row mb-5 justify-content-center\"> | HTML response containing <div class=\"row mb-5 justify-content-center\"> | Pass |
| 16 | Test Add Messages(Invalid Username),  Method : POST  Request Type : JSON  URL Arguments : No | Invalid Username | Invalid Username | Pass |
| 17 | Test Add Messages(Message Sent Successfully),  Method : POST  Request Type : JSON  URL Arguments : No | Message Sent Successfully | Message Sent Successfully | Pass |
| 18 | Test Get Messages(Check if new message was added),  Method : GET  Request Type : None  URL Arguments : Yes | New messages added | No new message added | Fail |
| 19 | Test Get Messages that match with the search tag,  Method : GET  Request Type : None  URL Arguments : Yes | HTML response containing  <div class=\"chat\_list\" | HTML response containing  <div class=\"chat\_list\" | Pass |
| 20 | Test Get Messages(Fetch all messages),  Method : GET  Request Type : None  URL Arguments : Yes | HTML response containing  <div class=\"chat\_list\" | HTML response containing  <div class=\"chat\_list\" | Pass |
| 21 | Test Add Post,  Method : POST  Request Type : JSON  URL Arguments : No | Post Created | Post Created | Pass |
| 22 | Test Add Comments,  Method : POST  Request Type : JSON  URL Arguments : No | Comment Posted Successfully | Comment Posted Successfully | Pass |
| 23 | Test Get Posts(Fetch Personal Feed for a given user),  Method : GET  Request Type : None  URL Arguments : Yes | HTML response containing <div class=\"col-md-6 col-lg-4 mb-5\"> | HTML response containing <div class=\"col-md-6 col-lg-4 mb-5\"> | Pass |
| 24 | Test Get Post(Fetch content of a single post with the user who posted it and comments),  Method : GET  Request Type : None  URL Arguments : Yes | HTML response containing <div class=\"pt-5\"><h3 class=\"mb-5\"> | HTML response containing <div class=\"pt-5\"><h3 class=\"mb-5\"> | Pass |

1. Observation and Conclusion

We observed that the Flask application returns the expected output for 23 out of 24 test requests sent as part of 12 test cases to the application. This gives a success rate of 95.83% for the application. The request for which a wrong output was returned was the the test case which checks if new messages were added to the database. This might be related to a bug in eventaul consistency on our application. In conclusion, our Flask application is working as expected for the requirements and unit testing results reflect the same observation, with a success rate of 95.83% for incoming requests.

System Testing

By: Athul Sandosh PES1201701110

1. Introduction

The application is a career networking website built with HTML, CSS, Javascript, Bootstrap and jQuery on the front end; and Flask and MySQL on the backend. For system testing, we used the Selenium IDE tool. We used the record and playback feature of the Selenium IDE for testing our project.

1. Objective

In system testing, we are trying to observe if the front end of our website is working as expected. Basically, the objective of system testing is to simulate end user engagement from all possible scenarios, so that it can be ensured that an end user faces no hassle while the website is deployed in a production environment. To test for this, we recorded various scenarios that an end user using our website would be exposed to and executed them multiple times.

1. Test Report

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test case No. | Test Case Description | Expected Output | Actual Output | Test Result (Pass/Fail) |
| 1 | Accept Connection Request :  Logged in user accepts a connection request sent by another user. | <p> tag with id Request Approved will appear on screen, and  assert element present id=Request Approved will be triggered. | <p> tag with id Request Approved appears on screen, and  assert element present id=Request Approved is triggered. | Pass |
| 2 | Add Comment : Logged in user adds comment to an existing post. | <p> tag with id Comment Posted Successfully will appear on screen, and  assert element present id= Comment Posted Successfully will be triggered. | <p> tag with id Comment Posted Successfully  appears on screen, and  assert element present id= Comment Posted Successfully is triggered. | Pass |
| 3 | Add Job Listing :  Logged in user adds a job listing | User is redirected to index.html after pressing the submit button. | User is redirected to index.html after pressing the submit button. | Pass |
| 4 | Add Message in Existing Chat :  Logged in user adds a message in an existing chat. | Message is added and can be seen in the chat. | Message is added and can be seen in the chat. | Pass |
| 5 | Add Post :  Logged in user creates a new post | User is redirected to feed.html after pressing the submit button. | User is redirected to feed.html after pressing the submit button. | Pass |
| 6 | Login(Email Not Registered) :  Login Failed because email id is not present in the database. | User is redirected to signup.html after pressing the submit button. | User is redirected to signup.html after pressing the submit button. | Pass |
| 7 | Login(Invalid Password) :  Login Failed because entered password is incorrect. | <p> tag with id Invalid Password will appear on screen, and  assert element present id= Invalid Password will be triggered. | <p> tag with id Invalid Password appears on screen, and  assert element present id= Invalid Password is triggered. | Pass |
| 8 | Login(Login Successful) | User is redirected to index.html after pressing the submit button. | User is redirected to index.html after pressing the submit button. | Pass |
| 9 | Search For Job Listings :  Logged in user searches for job listings | Job listings that match the search parameters are displayed on the index.html page. | Job listings that match the search parameters are displayed on the index.html page. | Pass |
| 10 | Send Connection Request to non suggestion :  Logged in user sends connection request to non suggestion | User is redirected to connections.html after pressing the submit button. | User is redirected to connections.html after pressing the submit button. | Pass |
| 11 | Send Connection Request to suggestion :  Logged in user sends connection request to suggestion | <p> tag with id Request Sent will appear on screen, and  assert element present id= Request Sent will be triggered. | <p> tag with id Request Sent appears on screen, and  assert element present id= Request Sent is triggered. | Pass |
| 12 | Signup(Email Already Registered) :  Signup failed because email is already present in the database. | User is redirected to login.html after pressing the submit button. | User is redirected to login.html after pressing the submit button. | Pass |
| 13 | Signup(Password and Repeat Password do not match) :  Signup failed because password and repeat password are not the same. | <p> tag with id Password and repeat password do not match will appear on screen, and  assert element present id= Password and repeat password do not match will be triggered. | <p> tag with id Password and repeat password do not match will appear on screen, and  assert element present id= Password and repeat password do not match will be triggered. | Pass |
| 14 | Signup(Profile Created Successfully) : | User is redirected to index.html after pressing the submit button. | User is redirected to index.html after pressing the submit button. | Pass |
| 15 | Start New Chat(Invalid Username) :  Logged in user attempts to start a new chat with a non existent user. | <p> tag with id Message too long or Invalid Username will appear on screen, and  assert element present id= Message too long or Invalid Username will be triggered. | <p> tag with id Message too long or Invalid Username will appear on screen, and  assert element present id= Message too long or Invalid Username will be triggered. | Pass |
| 16 | Start New Chat(Message Sent Successfully) :  Logged in user starts a new chat with another user successfully. | User is redirected to messages.html after pressing the submit button. | User is redirected to messages.html after pressing the submit button. | Pass |

1. Observation and Conclusion

We observed our website mimics the expected behaviour for 16 out of 16 test cases. This gives a success rate of 100% for the application. In conclusion, our website is working as expected for the requirements and system testing results reflect the same observation, with a success rate of 100% for the the test cases that we recorded.

Performance/Load Testing

By Aditya Shankaran: PES1201700710

1. Introduction

The application is a career networking website built with HTML, CSS, Javascript, Bootstrap and jQuery on the front end; and Flask and MySQL on the backend. For load testing, we used the Locust software built in Python. We mentioned the specifications of our testing in a locustfile.py, which Locust reads and performs tests accordingly.

1. Objective

In performance/load testing, we are trying to observe the volume and velocity of incoming requests that our Flask application can handle. Basically, the objective of load testing is to simulate a huge number of concurrent users at one time, in order to test the robustness and durability of our backend application. Locust takes three input parameters on its console : the IP address where the application is running, the number of concurrent users to simulate, and the hatch rate or velocity of the user creation. We run our Flask application on localhost:5000, and we tested our application with 100 concurrent users and a hatch rate of 5. It should be noted that the number of users does not match with the number of incoming requests. For each user spawned, 23 requests are sent to the Flask application. This means that a total of 2300 requests are sent to the Flask application by Locust, with 115 requests per second being sent.

1. Test Report

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test case No. | Test Case Description | Expected Output | Actual Output | Test Result (Pass/Fail) |
| 1 | API path : /api/v1/connections  Method : POST | Invalid Email/Connection Request Sent/Request Approved | Same as expected output in 55/62 cases | Pass in 55/62 cases |
| 2 | API path : /api/v1/connections  ?fetchdata=True  &username=Sanjay  %20Chari  Method : GET | HTML response containing <div class=\"row mb-5 justify-content-center\"> | Same as expected output in 9/59 cases | Pass in 9/59 cases |
| 3 | API path : /api/v1/connections  ?search\_tag=A  &username=Sanjay  %20Chari  Method : GET | HTML response containing <div class=\"row mb-5 justify-content-center\"> | Same as expected output in 5/40 cases | Pass in 5/40 cases |
| 4 | API path : /api/v1/jobs  Method : POST | Job Created Successfully | Same as expected output in 18/22 cases | Pass in 18/22 cases |
| 5 | API path : /api/v1/jobs  ?term=Part%20Time  &location=Anywhere  &specs=Software  Method : GET | HTML response containing <div class=\"row align-items-start job-item border-bottom pb-3 mb-3 pt-3\"> | Same as expected output in 23/63 cases | Pass in 23/63 cases |
| 6 | API path : /api/v1/messages  Method : POST | Invalid Username/  Message Sent Successfully | Same as expected output in 31/38 cases | Pass in 31/38 cases |
| 7 | API path : /api/v1/messages  ?fetchdata=True  &username=Sanjay  %20Chari  Method : GET | HTML response containing <div class=\"chat\_list\" | Same as expected output in 10/18 cases | Pass in 10/18 cases |
| 8 | API path : /api/v1/messages  ?search\_tag=A  &username=Sanjay  %20Chari  Method : GET | HTML response containing <div class=\"chat\_list\" | Same as expected output in 10/23 cases | Pass in 10/23 cases |
| 9 | API path : /api/v1/posts  Method : POST | Post Created/  Comment Posted Successfully | Same as expected output in 26/35 cases | Pass in 26/35 cases |
| 10 | API path : /api/v1/posts  ?fetchdata=True  &username=Sanjay  %20Chari  Method : GET | HTML response containing <div class=\"col-md-6 col-lg-4 mb-5\"> | Same as expected output in 4/15 cases | Pass in 4/15 cases |
| 11 | API path : /api/v1/posts  ?post\_id=6  Method : GET | HTML response containing <div class=\"pt-5\"><h3 class=\"mb-5\"> | Same as expected output in 6/14 cases | Pass in 6/14 cases |
| 12 | API path : /api/v1/users  Method : POST | Signup :  Email Already Registered,  Password and Repeat Password do not match, Password in wrong format, Profile Created Successfully;  Login :  Email Not Registered, Invalid Password, Login Successful | Same as expected output in 355/355 cases | Pass in 355/355 cases |

1. Observation and Conclusion :

After 1 minute of Locust Execution :

A screenshot of a computer

Description automatically generated

After 2 Minutes of Locust Execution :

A screenshot of a computer

Description automatically generated

Final Output of Locust Execution :

A screenshot of a computer screen

Description automatically generated

As can be seen in the above screenshots, the percentage of failures to total requests goes up from 4% after 1 minute, to 15% after 2 minutes, to 26% finally. Also, other metrics like median, average, min and max also seem to go up.

Additionally, after 2 minutes, the Flask application began reporting a maximum queue size reached error, which implies that the maximum capacity of the backend application was reached due to inavailability of sufficient RAM at that instant of time.

Thus, as the load testing software ran for a longer amount of time, performance metrics like failure rate,median, average, min, max began deteriorating rapidly.

To conclude, load testing of our Flask application reported a failure rate of 26% for 100 concurrent users(each user 23 requests) with a hatch rate of 5. It can be concluded that the performance of a Flask application begans to deteriorate rapidly after the number of concurrent incoming requests exceeds the maximum queue size dictated by the computing capacity of the host machine on which the application is deployed.