Software Engineering IT314

Project: Student leave and TA assistantship management

Group: 9

- **Group Members:**
  - Preksha Anand - 202001061
  - Maharth Thakar - 202001069
  - Meet Patel - 202001074
  - Kartik Dangi - 202001079
  - Harsh Kathiriya - 202001088
  - Kuldipsinh Gohil - 202001092
  - Neha Miglani - 202001097
  - Harsh Anand - 202001101
  - Darshan Kheni - 202001109

Non-Functional Testing

Objectives of Non-Functional Testing:

- Non-functional testing aims to improve the overall quality of a product beyond its functional requirements. It helps in making the product more usable, effective, maintainable, and portable.
- By conducting non-functional testing, we can identify and reduce risks and expenses associated with the product's non-functional features such as performance, security, and reliability.
- Non-functional testing also helps in improving the installation, configuration, execution, management, and monitoring of the product.
- We can gather important measurements and metrics during non-functional testing that can help in analyzing and improving the product's behavior and current technology.
- Overall, non-functional testing is essential to ensure that the product is of high quality and meets the expectations of users.

Types of Non-Functional Testing:

1. Performance Testing
2. Security Testing
3. Usability Testing
4. Compatibility Testing
5. Scalability Testing
6. Recovery Testing

1. **Performance Testing:**
   Performance testing is required to make sure the response time of the system is as per the desired requirement.

- **Load Testing:**

  It is done by checking the system's ability to manage a load of multiple users using the system at the same time.
  During this testing, the testing team tries to put pressure on the system by logging with multiple users and accessing any particular module or function simultaneously.

  Tool used for Load Testing: **Locust**

  Video Link : Locust testing

  Below is the Locustfile.py File:

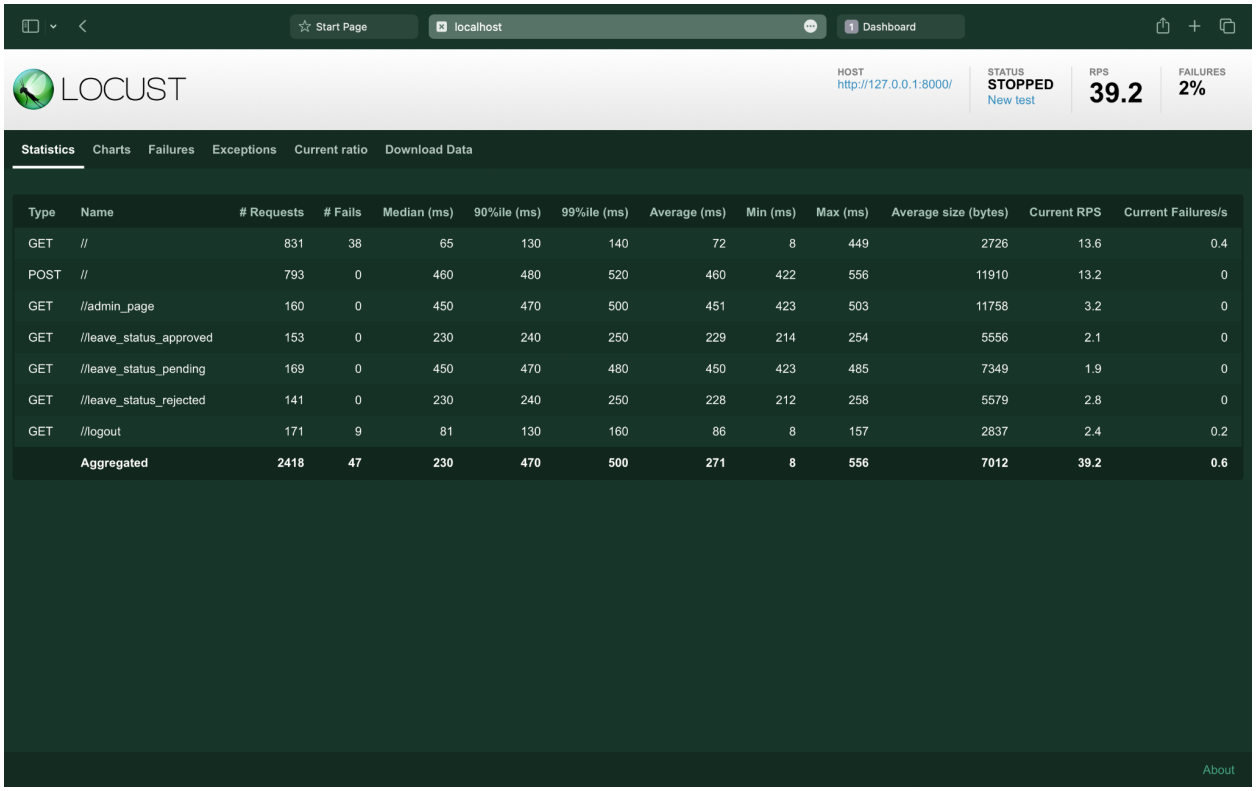
```
from locust import HttpUser, TaskSet, task, between

class MyLocust(HttpUser):
    # Define the wait time between requests to be between 10 and 5000 milliseconds
    wait_time = between(10, 5000)

    # Call the login function at the start of the test
    def on_Start(self):
        self.login()

    # Define the login task
    @task
    def login(self):
        # Send a GET request to the root URL to retrieve the CSRF token cookie
        response = self.client.get('/')
        # Extract the CSRF token from the cookie
        csrftoken = response.cookies['csrftoken']
        # Send a POST request with the login credentials and CSRF token
        response = self.client.post("/", {"username": "admin@gmail.com", "password": "admin"}, headers={"X-CSRFToken": csrftoken})
        # Return the response object
```

```
        return response

    # Define the logout task
    @task
    def logout(self):
        # Send a GET request to the logout URL
        response = self.client.get('/logout')
        # Return the response object
        return response

    # Define the admin page task
    @task
    def adminpage(self):
        # Call the login function to authenticate
        self.login()
        # Send a GET request to the admin page URL
        response = self.client.get('/admin_page')
        # Return the response object
        return response

    # Define the leave status rejected task
    @task
    def leave_status_rejected(self):
        # Call the login function to authenticate
        self.login()
        # Send a GET request to the leave status rejected URL
        response=self.client.get('/leave_status_rejected')
        # Return the response object
        return response

    # Define the leave status approved task
    @task
    def leave_status_approved(self):
        # Call the login function to authenticate
        self.login()
        # Send a GET request to the leave status approved URL
        response=self.client.get('/leave_status_approved')
        # Return the response object
        return response
```

```
    # Define the leave status pending task
    @task
    def leave_status_pending(self):
        # Call the login function to authenticate
        self.login()
        # Send a GET request to the leave status pending URL
        response=self.client.get('/leave_status_pending')
        # Return the response object
        return response
```

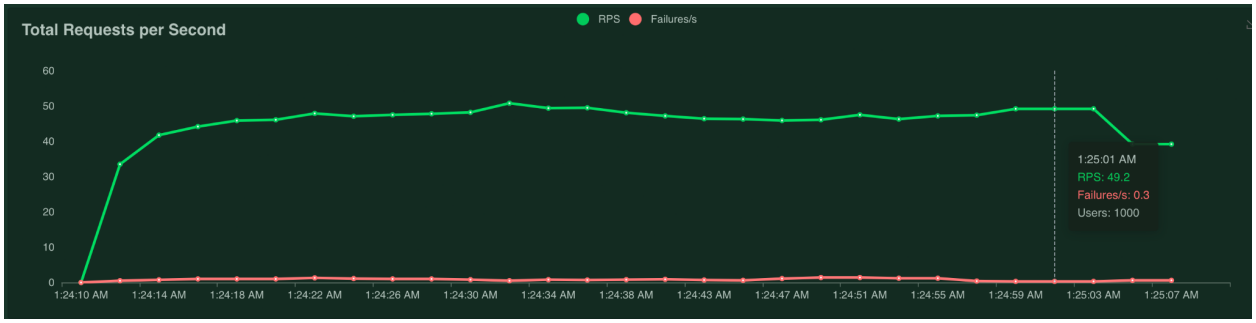
Screenshots of output:

Number Of Users: 1000
Hatch Rate: 10

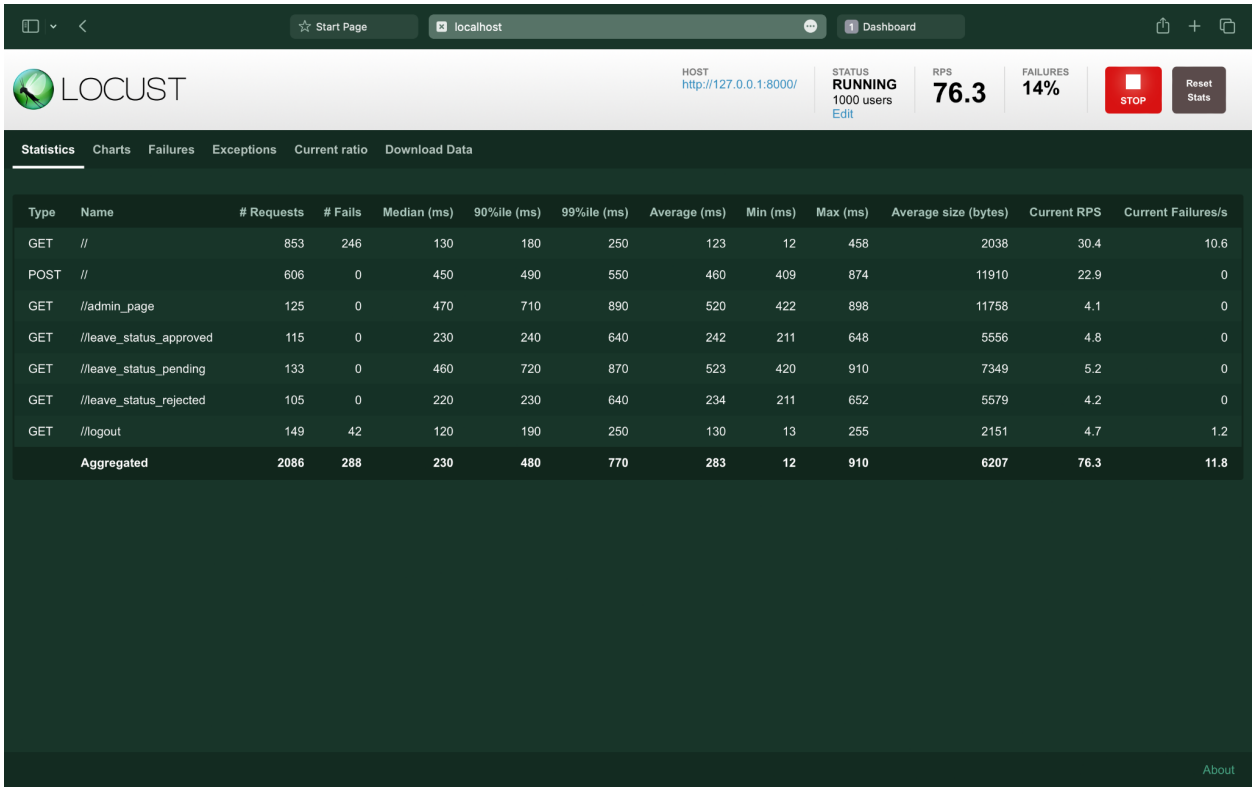


Type	Name	# Requests	# Fails	Median (ms)	90%ile (ms)	99%ile (ms)	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	Current RPS	Current Failures/s
GET	//	831	38	65	130	140	72	8	449	2726	13.6	0.4
POST	//	793	0	460	480	520	460	422	556	11910	13.2	0
GET	//admin_page	160	0	450	470	500	451	423	503	11758	3.2	0
GET	//leave_status_approved	153	0	230	240	250	229	214	254	5556	2.1	0
GET	//leave_status_pending	169	0	450	470	480	450	423	485	7349	1.9	0
GET	//leave_status_rejected	141	0	230	240	250	228	212	258	5579	2.8	0
GET	//logout	171	9	81	130	160	86	8	157	2837	2.4	0.2
	Aggregated	**2418**	**47**	**230**	**470**	**500**	**271**	**8**	**556**	**7012**	**39.2**	**0.6**

Graph Presentation:

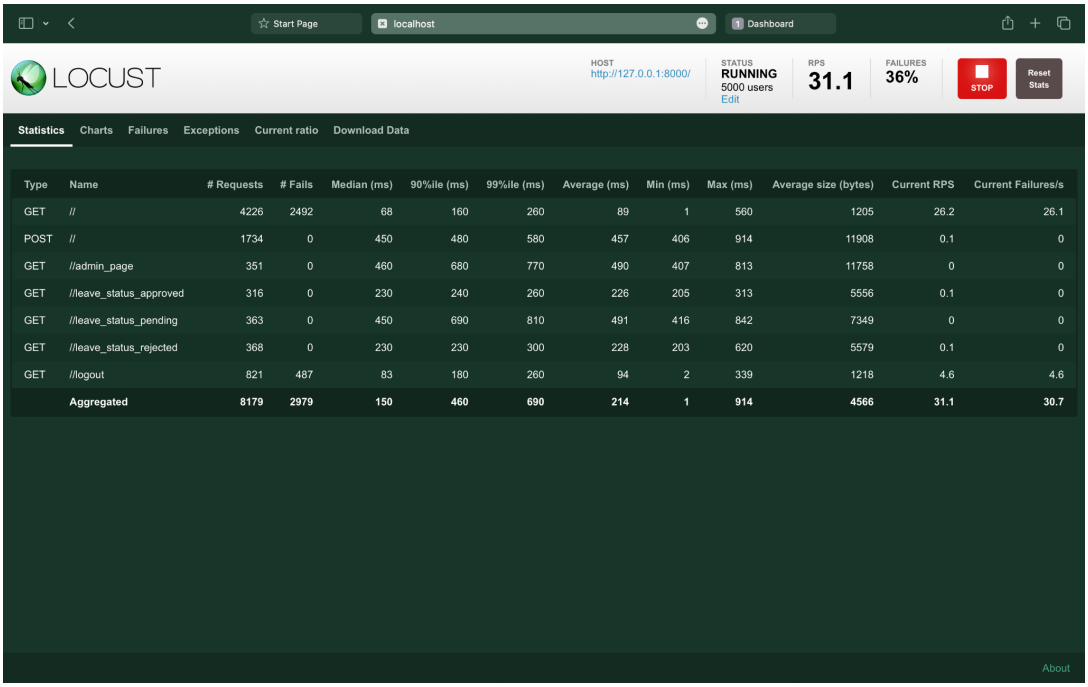


Number of Users: 1000
Hatch Rate: 50



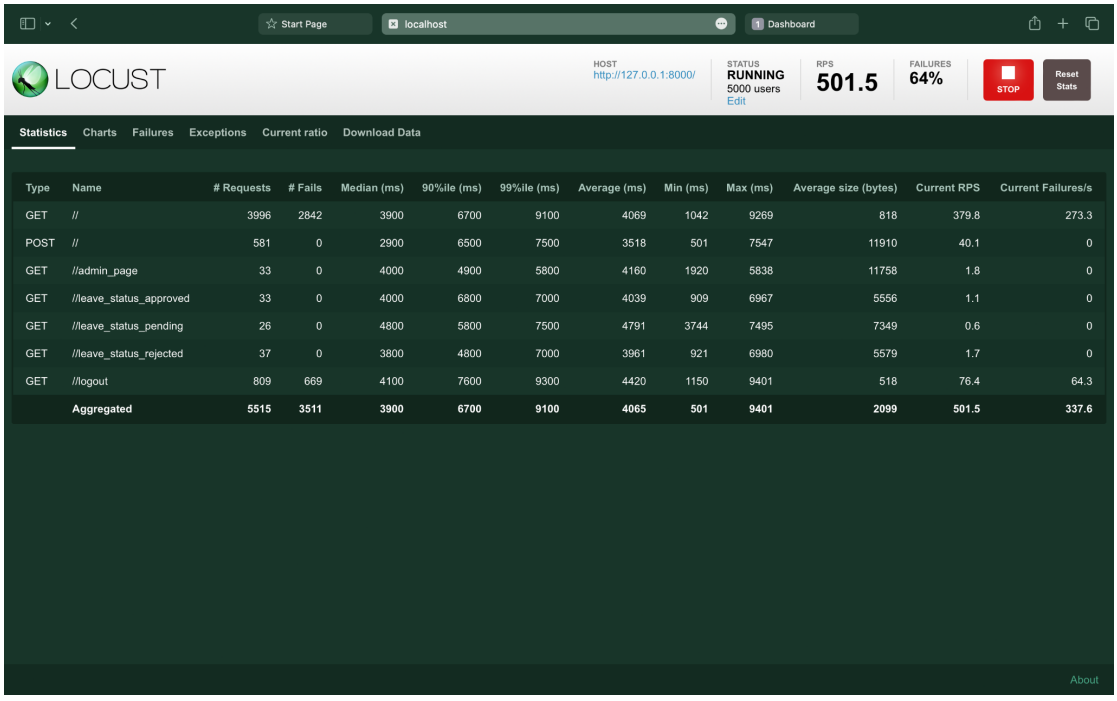
Type	Name	# Requests	# Fails	Median (ms)	90%ile (ms)	99%ile (ms)	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	Current RPS	Current Failures/s
GET	//	853	246	130	180	250	123	12	458	2038	30.4	10.6
POST	//	606	0	450	490	550	460	409	874	11910	22.9	0
GET	//admin_page	125	0	470	710	890	520	422	898	11758	4.1	0
GET	//leave_status_approved	115	0	230	240	640	242	211	648	5556	4.8	0
GET	//leave_status_pending	133	0	460	720	870	523	420	910	7349	5.2	0
GET	//leave_status_rejected	105	0	220	230	640	234	211	652	5579	4.2	0
GET	//logout	149	42	120	190	250	130	13	255	2151	4.7	1.2
	Aggregated	**2086**	**288**	**230**	**480**	**770**	**283**	**12**	**910**	**6207**	**76.3**	**11.8**

Number Of Users: 5000
Hatch Rate: 50



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	//	4226	2492	68	160	260	89	1	560	1205	26.2	26.1
POST	//	1734	0	450	480	580	457	406	914	11908	0.1	0
GET	//admin_page	351	0	460	680	770	490	407	813	11758	0	0
GET	//leave_status_approved	316	0	230	240	260	226	205	313	5556	0.1	0
GET	//leave_status_pending	363	0	450	690	810	491	416	842	7349	0	0
GET	//leave_status_rejected	368	0	230	230	300	228	203	620	5579	0.1	0
GET	//logout	821	487	83	180	260	94	2	339	1218	4.6	4.6
	Aggregated	**8179**	**2979**	**150**	**460**	**690**	**214**	**1**	**914**	**4566**	**31.1**	**30.7**

Number Of Users: 5000
Hatch Rate: 500



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	//	3996	2842	3900	6700	9100	4069	1042	9269	818	379.8	273.3
POST	//	581	0	2900	6500	7500	3518	501	7547	11910	40.1	0
GET	//admin_page	33	0	4000	4900	5800	4160	1920	5838	11758	1.8	0
GET	//leave_status_approved	33	0	4000	6800	7000	4039	909	6967	5556	1.1	0
GET	//leave_status_pending	26	0	4800	5800	7500	4791	3744	7495	7349	0.6	0
GET	//leave_status_rejected	37	0	3800	4800	7000	3961	921	6980	5579	1.7	0
GET	//logout	809	669	4100	7600	9300	4420	1150	9401	518	76.4	64.3
	Aggregated	**5515**	**3511**	**3900**	**6700**	**9100**	**4065**	**501**	**9401**	**2099**	**501.5**	**337.6**

As Ratio of Number of Users and Hatch Rate decreases, The Failure Rate increases. Because as the number of users spawned per second are increasing, the response time of system will decrease and failure rate will be high.

Limitations Faced during Locust Testing:

- Locust will gather and display stats as it's creating users, but as soon as it reaches your requested number, all the stats are reset and it starts gathering new ones. If one wants to see how many users it takes to cause problems on your site, one needs to sit and stare at the web interface as it ramps up or that information will be gone.
- Locust won't request a URL unless you explicitly tell it to. A legitimate user with an actual web browser will most likely end up requesting several URLs when loading a page (for images, style sheets, and scripts).

Stress Tesing:

- It is performed to find the behavior of the application under extreme load.
- Application's quality attributes determined during stress testing-
  Breaking point, Robustness, Recoverability, Stability, etc.

2. **Security Testing:**

1) Check if access privileges are implemented correctly. Yes
2) Password and other sensitive fields should be masked while typing.Yes
3) The password should not be stored in cookies.Yes
4) Check application logout functionality.Yes
5) Sensitive fields like passwords should not have to autocomplete enabled.Yes
6) Page crash should not reveal application or server information. Error page should be displayed for this. Yes