Project Development Phase
Model Performance Test

Model Performance Testing:

S.No.	Parameter	Values
1.	Metrics	**Regression Model:** MAE – 0.0476 MSE – 0.0046 RMSE – 0.0682 R2 score – 0.78
2.	Tune the Model	Hyper-parameter Tuning – GridSearchCV and RandomizedSearchCV Validation Method – Cross Validation with Ridge and Lasso
3.	Testing	Testing model: Total Request per Second, Response time, No of users
4.	Locus Testing Report	Request Statistics, Response Time Statistics, Final Ratio

1.Metrics :

In [55]:
```
from math import sqrt
from sklearn.metrics import mean_absolute_error, mean_squared_error

print("Mean Absolute Error :", mean_absolute_error(y_test, y_pred))
print("Mean Squared Error :", mean_squared_error(y_test, y_pred))
print("Rooted Mean Squared Error :", sqrt(mean_squared_error(y_test, y_pred)))
print("R2 Score :", r2_score(y_test, y_pred))
```

```
Mean Absolute Error : 0.04766740707308981
Mean Squared Error : 0.004658892249358201
Rooted Mean Squared Error : 0.06825607847919628
R2 Score : 0.7777563100798979
```

2.Tune the Model :

Hyperparameter Tuning

In [45]:
```
def evaluate(models):
    results = {}
    for i, j in models.items():
        results[i] = [j.best_params_, j.best_estimator_, j.best_score_]
    return pd.DataFrame(results, index=['Best Parameter', 'Best Estimator', 'Best Score'])
```

In [46]:
```
#Using GridSearchCV
from sklearn.model_selection import GridSearchCV

lasso_params = {'alpha':[0.002, 0.00024, 0.00025, 0.0026, 0.03]}
ridge_params = {'alpha':[0.002, 0.0024, 0.0025, 0.0026, 0.03, 0.04]}

lsgs = GridSearchCV(Lasso(), param_grid=lasso_params, cv=5)
rdgs = GridSearchCV(Ridge(), param_grid=ridge_params, cv=5)

models2 = {'OLS': LinearRegression(),
           'Lasso': lsgs.fit(X, y).best_estimator_,
           'Ridge': rdgs.fit(X, y).best_estimator_,}

test(models2, X, y)
```

Out[46]:

	OLS	Lasso	Ridge
Training Results	0.796207	0.794771	0.794814
Testing Results	0.774031	0.783381	0.782087

In [47]:
```
cv = {'Lasso': lsgs,
      'Ridge': rdgs}
evaluate(cv)
```

Out[47]:

	Lasso	Ridge
Best Parameter	{'alpha': 0.00025}	{'alpha': 0.002}
Best Estimator	Lasso(alpha=0.00025)	Ridge(alpha=0.002)
Best Score	0.759762	0.759304

In [48]:
```
#Using RandomizedSearchCV
from sklearn.model_selection import RandomizedSearchCV

lsrs = RandomizedSearchCV(estimator=Lasso(), param_distributions=lasso_params, cv = 3, n_iter = 5)
rdrs = RandomizedSearchCV(estimator=Ridge(), param_distributions=ridge_params, cv = 3, n_iter = 5)

models3 = {'OLS': LinearRegression(),
           'Lasso': lsrs.fit(X, y).best_estimator_,
           'Ridge': rdrs.fit(X, y).best_estimator_,}

test(models3, X, y)
```

Out[48]:

	OLS	Lasso	Ridge
Training Results	0.796065	0.796405	0.794866
Testing Results	0.777465	0.776379	0.783031

In [49]:
```
cv = {'Lasso': lsrs,
      'Ridge': rdrs}
evaluate(cv)
```

Out[49]:

	Lasso	Ridge
Best Parameter	{'alpha': 0.00024}	{'alpha': 0.002}
Best Estimator	Lasso(alpha=0.00024)	Ridge(alpha=0.002)
Best Score	0.721576	0.721907

LassoCV and RidgeCV Validation

In [50]:
```
models4 = {'LassoCV': LassoCV(alphas=lasso_params['alpha']),
           'RidgeCV': RidgeCV(alphas=ridge_params['alpha'])}
test(models4, X, y)
```

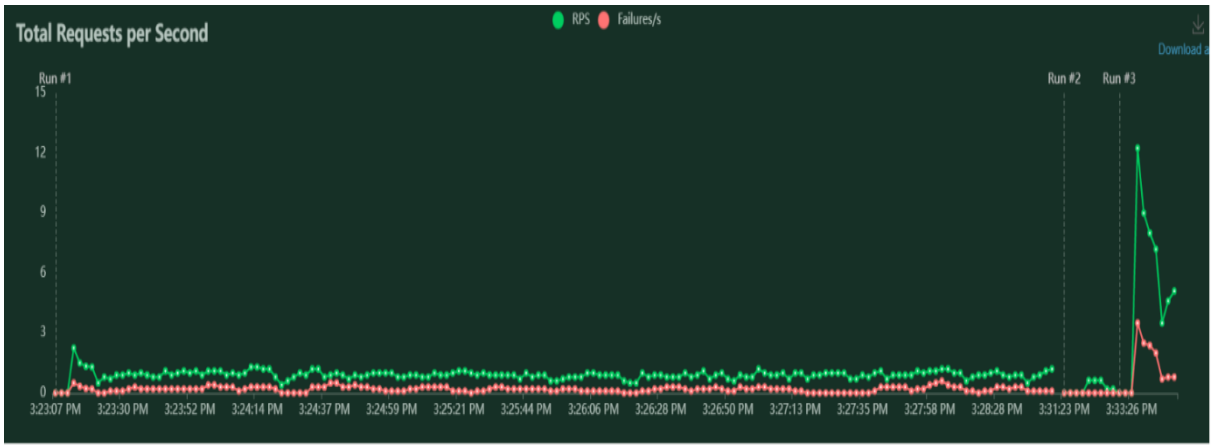
Out[50]:

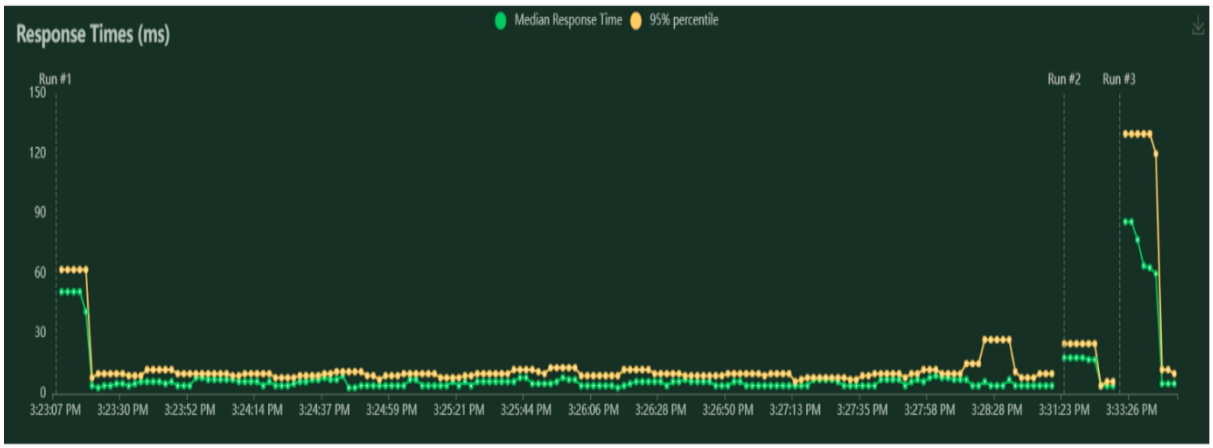
	LassoCV	RidgeCV
Training Results	0.794466	0.795869
Testing Results	0.780977	0.776775

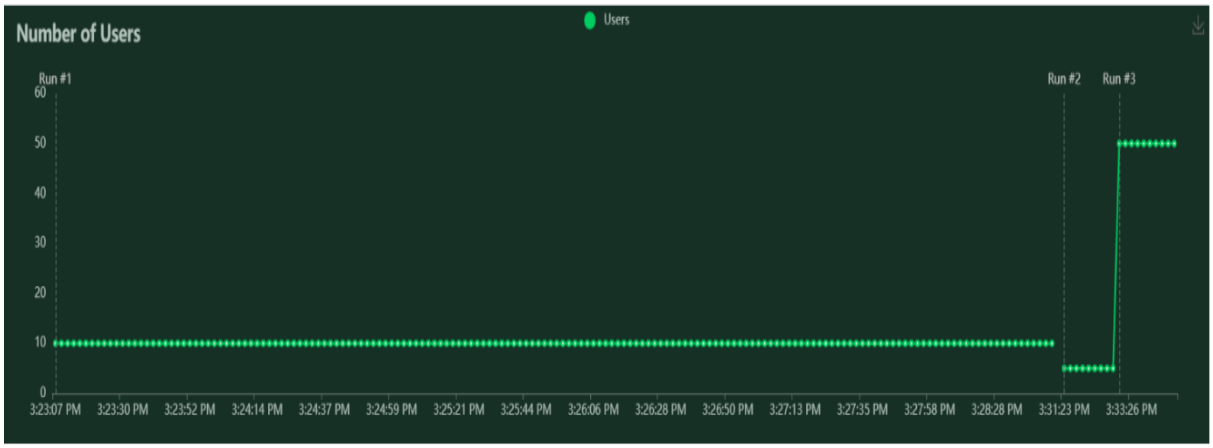
Build, Train and Test the Best Model

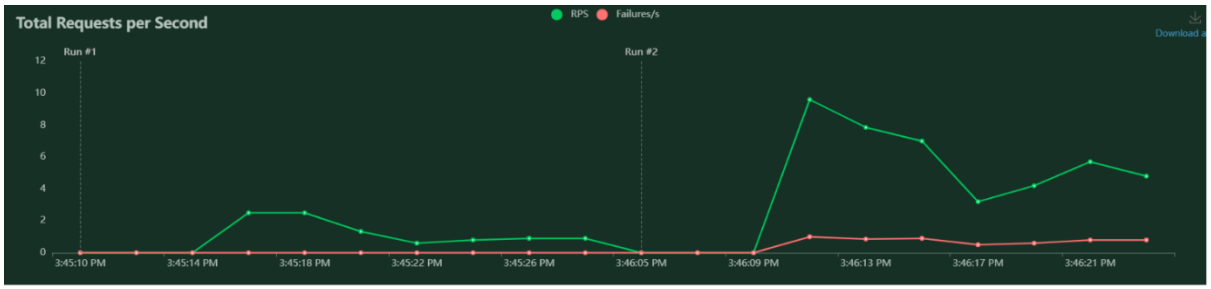
In [51]:
```
ridge_cv = RidgeCV(alphas=ridge_params['alpha'])

ridge_cv.fit(X_train, y_train)

y_pred_train = ridge_cv.predict(X_train)
y_pred = ridge_cv.predict(X_test)
```

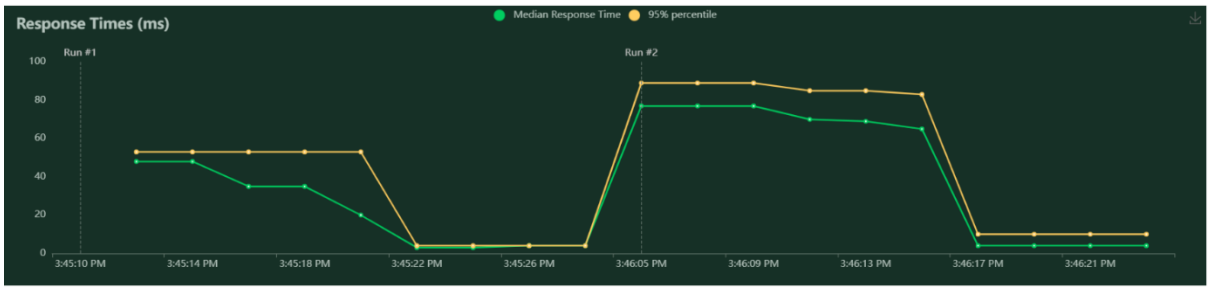
3.Testing:

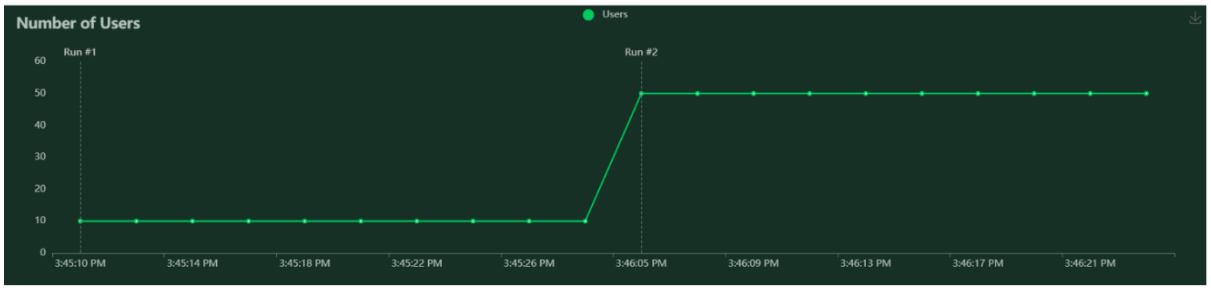

Total Requests per Second
RPS
Failures/s
Run #1
Run #2
Run #3
15
12
9
6
3
0
3:23:07 PM
3:23:30 PM
3:23:52 PM
3:24:14 PM
3:24:37 PM
3:24:59 PM
3:25:21 PM
3:25:44 PM
3:26:06 PM
3:26:28 PM
3:26:50 PM
3:27:13 PM
3:27:35 PM
3:27:58 PM
3:28:28 PM
3:31:23 PM
3:33:26 PM

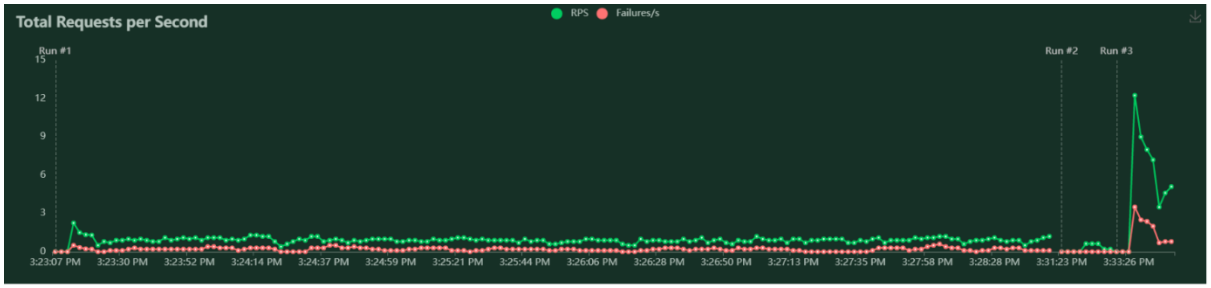

Response Times (ms)
Median Response Time
95% percentile
Run #1
Run #2
Run #3
150
120
90
60
30
0
3:23:07 PM
3:23:30 PM
3:23:52 PM
3:24:14 PM
3:24:37 PM
3:24:59 PM
3:25:21 PM
3:25:44 PM
3:26:06 PM
3:26:28 PM
3:26:50 PM
3:27:13 PM
3:27:35 PM
3:27:58 PM
3:28:28 PM
3:31:23 PM
3:33:26 PM

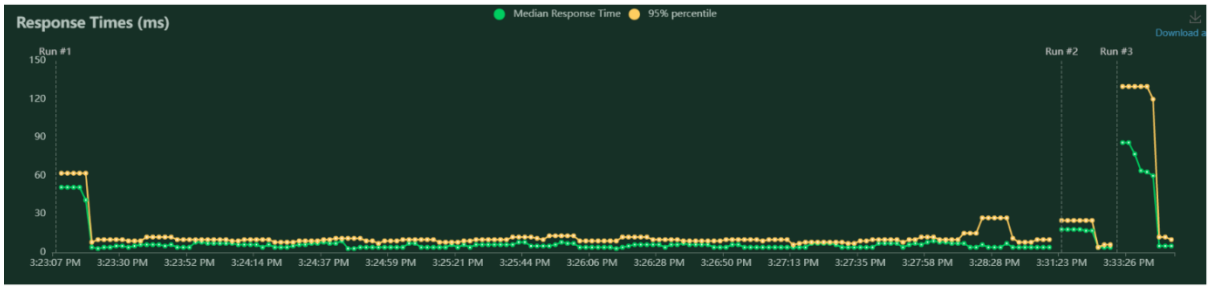

Number of Users
Users
Run #1
Run #2
Run #3
60
50
40
30
20
10
0
3:23:07 PM
3:23:30 PM
3:23:52 PM
3:24:14 PM
3:24:37 PM
3:24:59 PM
3:25:21 PM
3:25:44 PM
3:26:06 PM
3:26:28 PM
3:26:50 PM
3:27:13 PM
3:27:35 PM
3:27:58 PM
3:28:28 PM
3:31:23 PM
3:33:26 PM

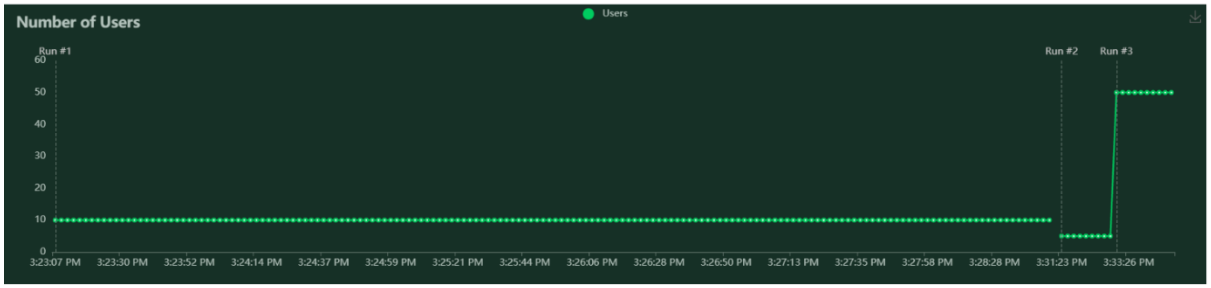

Total Requests per Second
RPS
Failures/s
Run #1
Run #2
12
10
8
6
4
2
0
3:45:10 PM
3:45:14 PM
3:45:18 PM
3:45:22 PM
3:45:26 PM
3:46:05 PM
3:46:09 PM
3:46:13 PM
3:46:17 PM
3:46:21 PM

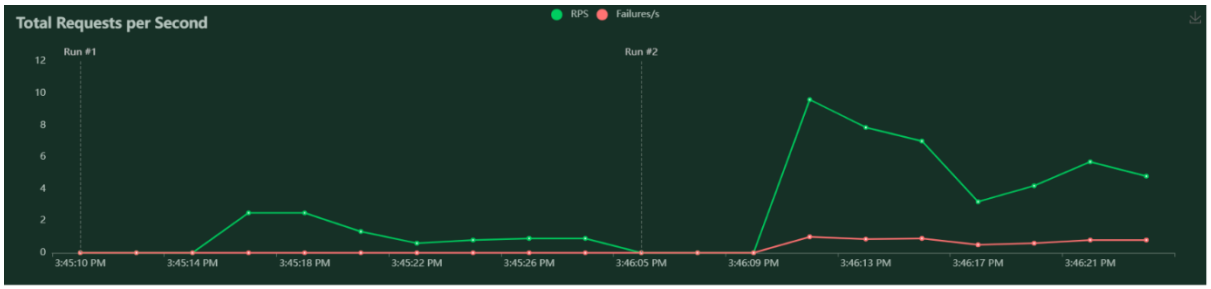

Response Times (ms)
Median Response Time
95% percentile
Run #1
Run #2
100
80
60
40
20
0
3:45:10 PM
3:45:14 PM
3:45:18 PM
3:45:22 PM
3:45:26 PM
3:46:05 PM
3:46:09 PM
3:46:13 PM
3:46:17 PM
3:46:21 PM

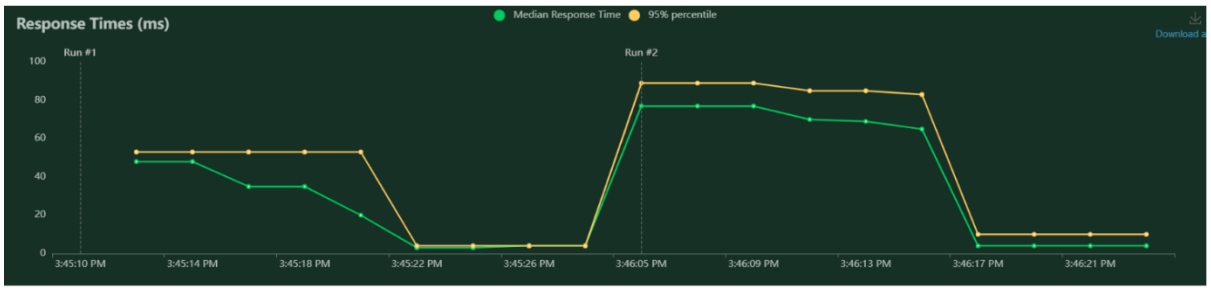

Number of Users
Users
Run #1
Run #2
60
50
40
30
20
10
0
3:45:10 PM
3:45:14 PM
3:45:18 PM
3:45:22 PM
3:45:26 PM
3:46:05 PM
3:46:09 PM
3:46:13 PM
3:46:17 PM
3:46:21 PM

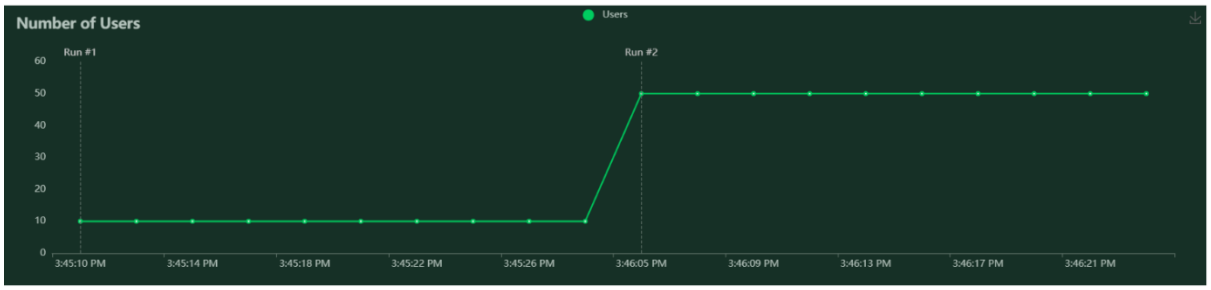

Total Requests per Second
RPS
Failures/s
Run #1
Run #2
Run #3
15
12
9
6
3
0
3:23:07 PM
3:23:30 PM
3:23:52 PM
3:24:14 PM
3:24:37 PM
3:24:59 PM
3:25:21 PM
3:25:44 PM
3:26:06 PM
3:26:28 PM
3:26:50 PM
3:27:13 PM
3:27:35 PM
3:27:58 PM
3:28:28 PM
3:31:23 PM
3:33:26 PM

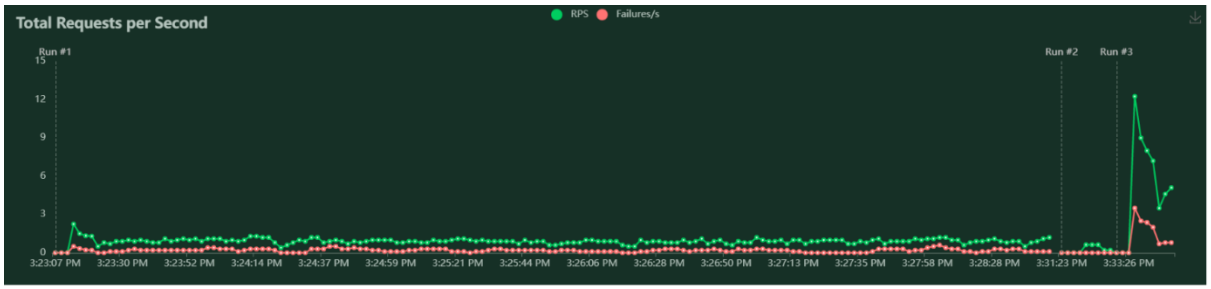

Response Times (ms)
Median Response Time
95% percentile
Run #1
Run #2
Run #3
150
120
90
60
30
0
3:23:07 PM
3:23:30 PM
3:23:52 PM
3:24:14 PM
3:24:37 PM
3:24:59 PM
3:25:21 PM
3:25:44 PM
3:26:06 PM
3:26:28 PM
3:26:50 PM
3:27:13 PM
3:27:35 PM
3:27:58 PM
3:28:28 PM
3:31:23 PM
3:33:26 PM

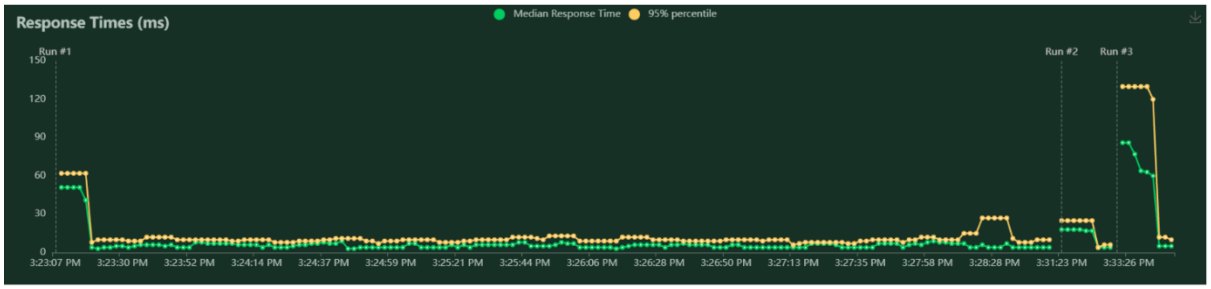

Number of Users
Users
Run #1
Run #2
Run #3
60
50
40
30
20
10
0
3:23:07 PM
3:23:30 PM
3:23:52 PM
3:24:14 PM
3:24:37 PM
3:24:59 PM
3:25:21 PM
3:25:44 PM
3:26:06 PM
3:26:28 PM
3:26:50 PM
3:27:13 PM
3:27:35 PM
3:27:58 PM
3:28:28 PM
3:31:23 PM
3:33:26 PM

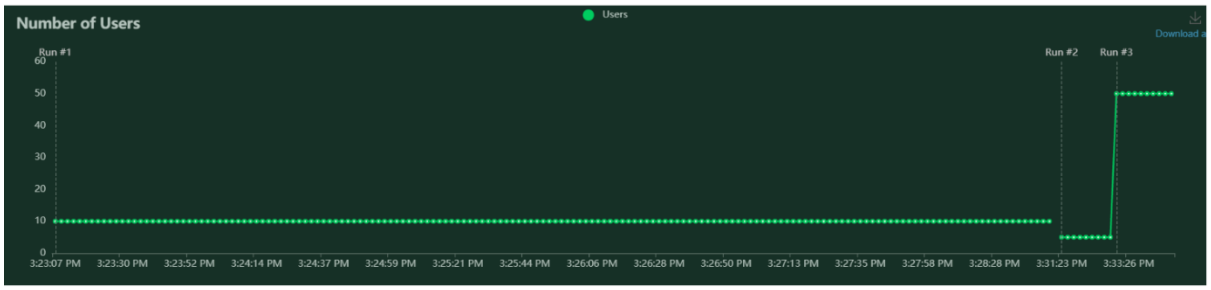

Total Requests per Second
RPS
Failures/s
Run #1
Run #2
12
10
8
6
4
2
0
3:45:10 PM
3:45:14 PM
3:45:18 PM
3:45:22 PM
3:45:26 PM
3:46:05 PM
3:46:09 PM
3:46:13 PM
3:46:17 PM
3:46:21 PM

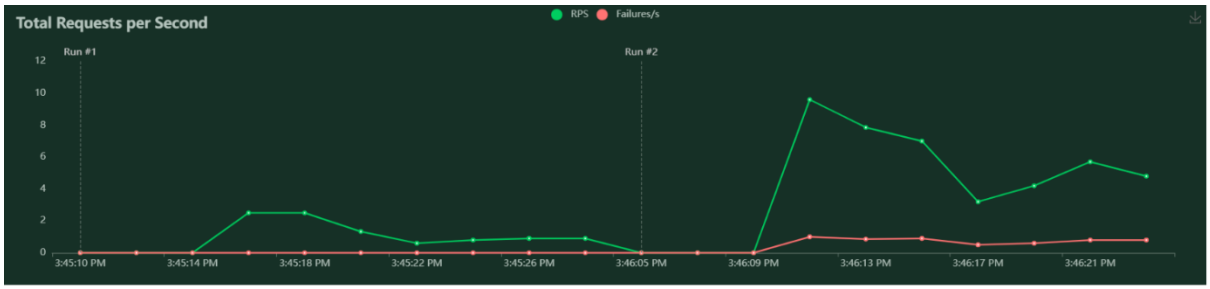

Response Times (ms)
Median Response Time
95% percentile
Download
Run #1
Run #2
100
80
60
40
20
0
3:45:10 PM
3:45:14 PM
3:45:18 PM
3:45:22 PM
3:45:26 PM
3:46:05 PM
3:46:09 PM
3:46:13 PM
3:46:17 PM
3:46:21 PM

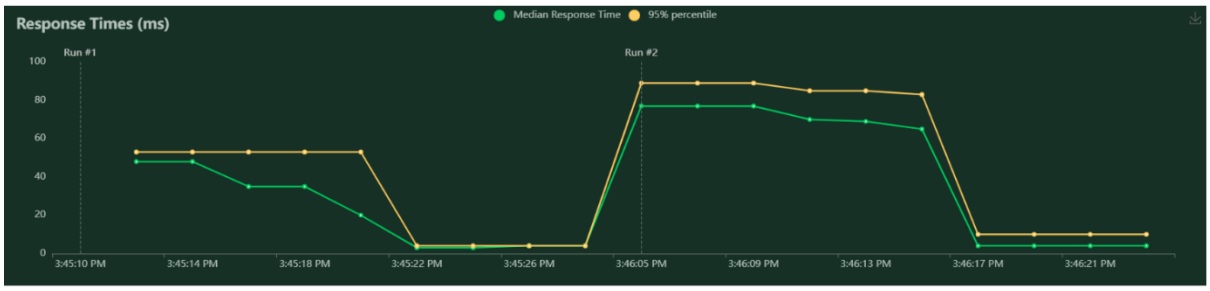

Number of Users
Users
Run #1
Run #2
60
50
40
30
20
10
0
3:45:10 PM
3:45:14 PM
3:45:18 PM
3:45:22 PM
3:45:26 PM
3:46:05 PM
3:46:09 PM
3:46:13 PM
3:46:17 PM
3:46:21 PM

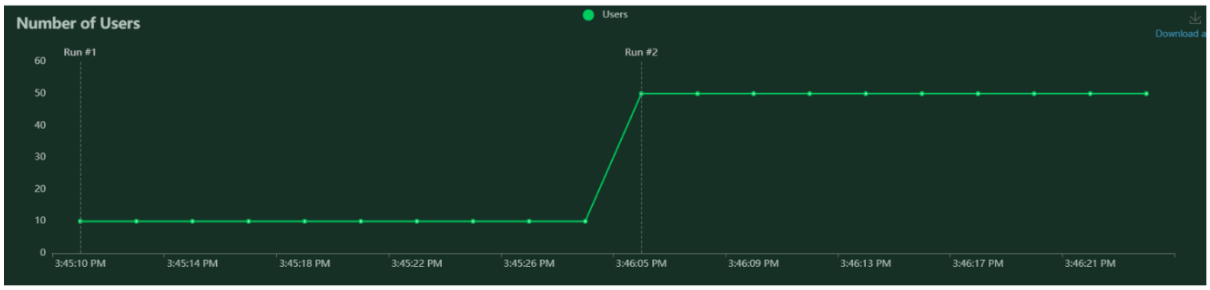

Total Requests per Second
RPS
Failures/s
Run #1
Run #2
Run #3
15
12
9
6
3
0
3:23:07 PM
3:23:30 PM
3:23:52 PM
3:24:14 PM
3:24:37 PM
3:24:59 PM
3:25:21 PM
3:25:44 PM
3:26:06 PM
3:26:28 PM
3:26:50 PM
3:27:13 PM
3:27:35 PM
3:27:58 PM
3:28:28 PM
3:31:23 PM
3:33:26 PM


Response Times (ms)
Median Response Time
95% percentile
Run #1
Run #2
Run #3
150
120
90
60
30
0
3:23:07 PM
3:23:30 PM
3:23:52 PM
3:24:14 PM
3:24:37 PM
3:24:59 PM
3:25:21 PM
3:25:44 PM
3:26:06 PM
3:26:28 PM
3:26:50 PM
3:27:13 PM
3:27:35 PM
3:27:58 PM
3:28:28 PM
3:31:23 PM
3:33:26 PM


Number of Users
Users
Download
Run #1
Run #2
Run #3
60
50
40
30
20
10
0
3:23:07 PM
3:23:30 PM
3:23:52 PM
3:24:14 PM
3:24:37 PM
3:24:59 PM
3:25:21 PM
3:25:44 PM
3:26:06 PM
3:26:28 PM
3:26:50 PM
3:27:13 PM
3:27:35 PM
3:27:58 PM
3:28:28 PM
3:31:23 PM
3:33:26 PM







LOCUST

HOST http://127.0.0.1:5000/ | STATUS STOPPED New test | RPS 5.1 | FAILURES 21%

Statistics | Charts | Failures | Exceptions | Current ratio | Download Data

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	//	25	0	8	130	130	36	6	130	3919	1.7	0
GET	//home	34	0	4	99	110	33	3	110	2959	1.4	0
GET	//register	26	26	63	110	110	55	8	115	265	0.6	0.6
GET	//university	37	0	4	93	110	33	3	109	19010	1.4	0
	Aggregated	**122**	**26**	**9**	**110**	**130**	**38**	**3**	**130**	**7450**	**5.1**	**0.6**

LOCUST

HOST http://127.0.0.1:5000 | STATUS STOPPED New test | RPS 4.8 | FAILURES 15%

Statistics | Charts | Failures | Exceptions | Current ratio | Download Data

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	/	22	0	4	81	90	27	3	90	3919	0.7	0
GET	/contact	20	0	4	83	89	39	3	89	3341	0.4	0
GET	/home	22	0	65	82	83	44	3	83	2959	0.6	0
GET	/register	19	19	10	80	90	28	9	90	265	0.8	0.8
GET	/university	21	0	4	79	85	23	3	85	19010	1.1	0
GET	/visual	22	0	4	80	82	26	3	82	16479	1.2	0
	Aggregated	**126**	**19**	**6**	**81**	**90**	**31**	**3**	**90**	**7817**	**4.8**	**0.8**

4.Locus Testing Report:

Request Statistics

Method	Name	# Requests	# Fails	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	RPS	Failures/s
GET	/	22	0	27	3	90	3919	1.1	0.0
GET	/contact	20	0	38	3	88	3341	1.0	0.0
GET	/home	22	0	44	3	83	2959	1.1	0.0
GET	/register	19	19	28	8	89	265	1.0	1.0
GET	/university	21	0	23	3	85	19010	1.1	0.0
GET	/visual	22	0	26	3	82	16479	1.1	0.0
	Aggregated	**126**	**19**	**31**	**3**	**90**	**7816**	**6.4**	**1.0**

Response Time Statistics

Method	Name	50%ile (ms)	60%ile (ms)	70%ile (ms)	80%ile (ms)	90%ile (ms)	95%ile (ms)	99%ile (ms)	100%ile (ms)
GET	/	4	4	69	74	81	81	90	90
GET	/contact	62	69	70	79	83	89	89	89
GET	/home	66	68	70	78	82	82	83	83
GET	/register	10	10	11	77	80	90	90	90
GET	/university	4	5	6	57	79	83	85	85
GET	/visual	5	5	61	74	80	80	82	82
	Aggregated	**6**	**10**	**67**	**74**	**81**	**83**	**90**	**90**

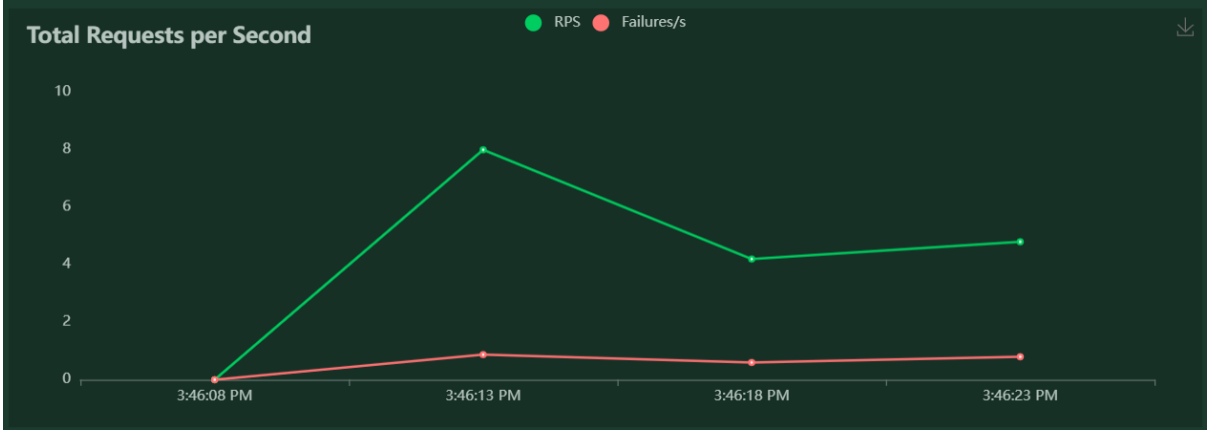
Failures Statistics

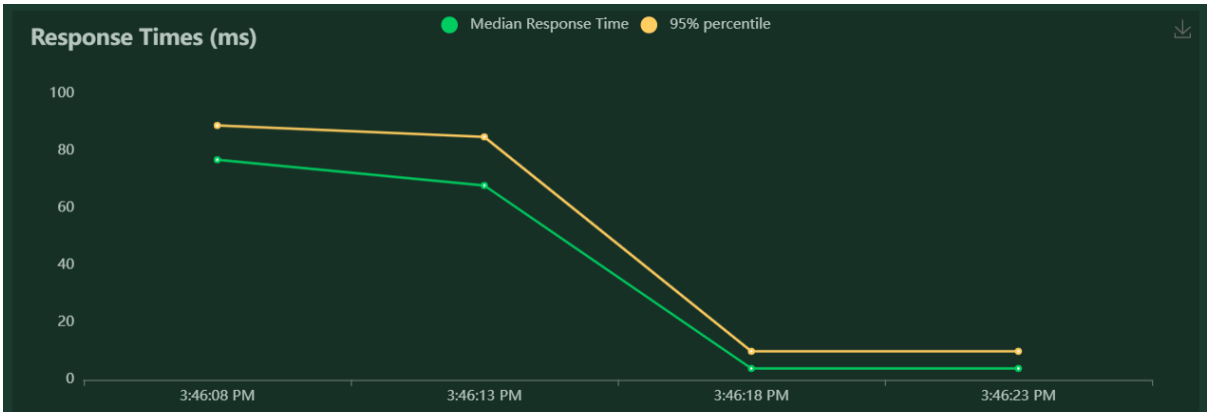
Method	Name	Error	Occurrences
GET	/register	500 Server Error: INTERNAL SERVER ERROR for url: http://127.0.0.1:5000/register	19

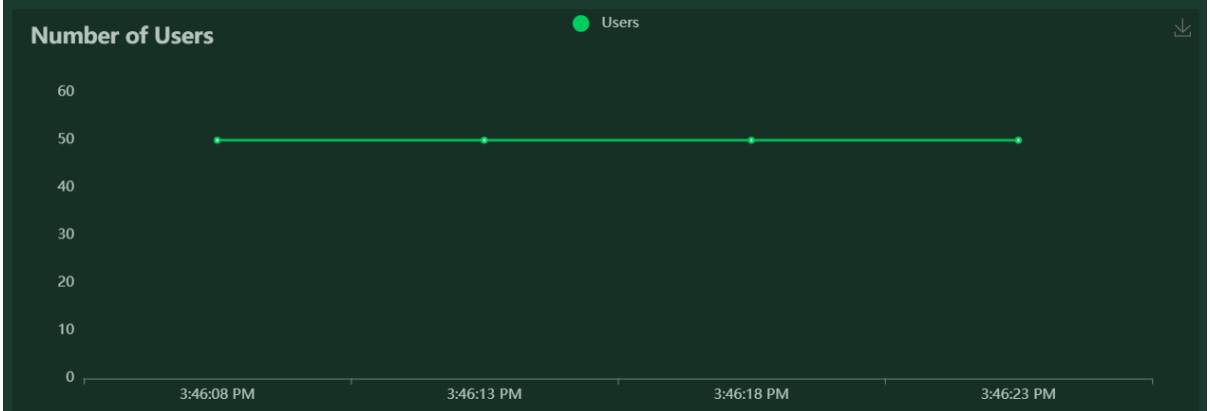
Exceptions Statistics

Count	Message	Traceback	Nodes
5	tuple index out of range	File "C:\Users\vijay\anaconda3\lib\site-packages\locust\user\task.py", line 347, in run self.execute_next_task() File "C:\Users\vijay\anaconda3\lib\site-packages\locust\user\task.py", line 372, in execute_next_task self.execute_task(self._task_queue.pop(0)) File "C:\Users\vijay\anaconda3\lib\site-packages\locust\user\task.py", line 493, in execute_task task(self.user) File "C:\Users\vijay\Desktop\IMP File\IBM Assignment\flask app 1\locustfile.py", line 23, in home data=data[random.randint(0,3)], headers=post_headers)	local

Charts







Final ratio

Ratio per User class

- 100.0% WebsiteUser
  - 16.7% index
  - 16.7% home
  - 16.7% home1
  - 16.7% visual
  - 16.7% visual1
  - 16.7% visual2

Total ratio

- 100.0% WebsiteUser
  - 16.7% index
  - 16.7% home
  - 16.7% home1
  - 16.7% visual
  - 16.7% visual1
  - 16.7% visual2