San Jose State University

Fall 2017

Lab 1

Calculator/Dropbox Web Application

CMPE 273

SUBMITTED BY –
Dishant Kimtani

SUBMITTED TO-
Prof. Simon Shim

Calculator Web Application

Goal :

The goal is to develop a Calculator web application to demonstrate how a “React Js” client interacts with a “Node Js” server.

Purpose :

The purpose of this assignment is to understand and implement the basic concepts of stateless connection , SQL connection pooling and understanding the concepts of Node Js framework , JMeter , React Js .

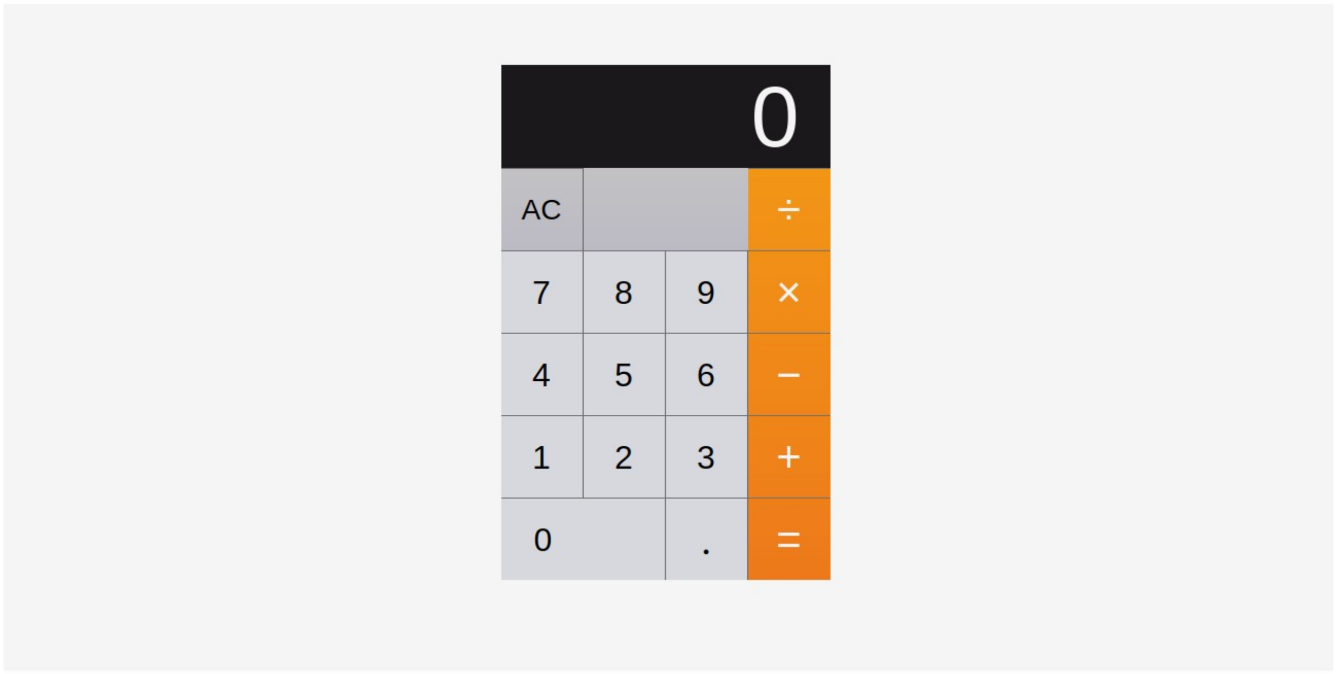
System Design :

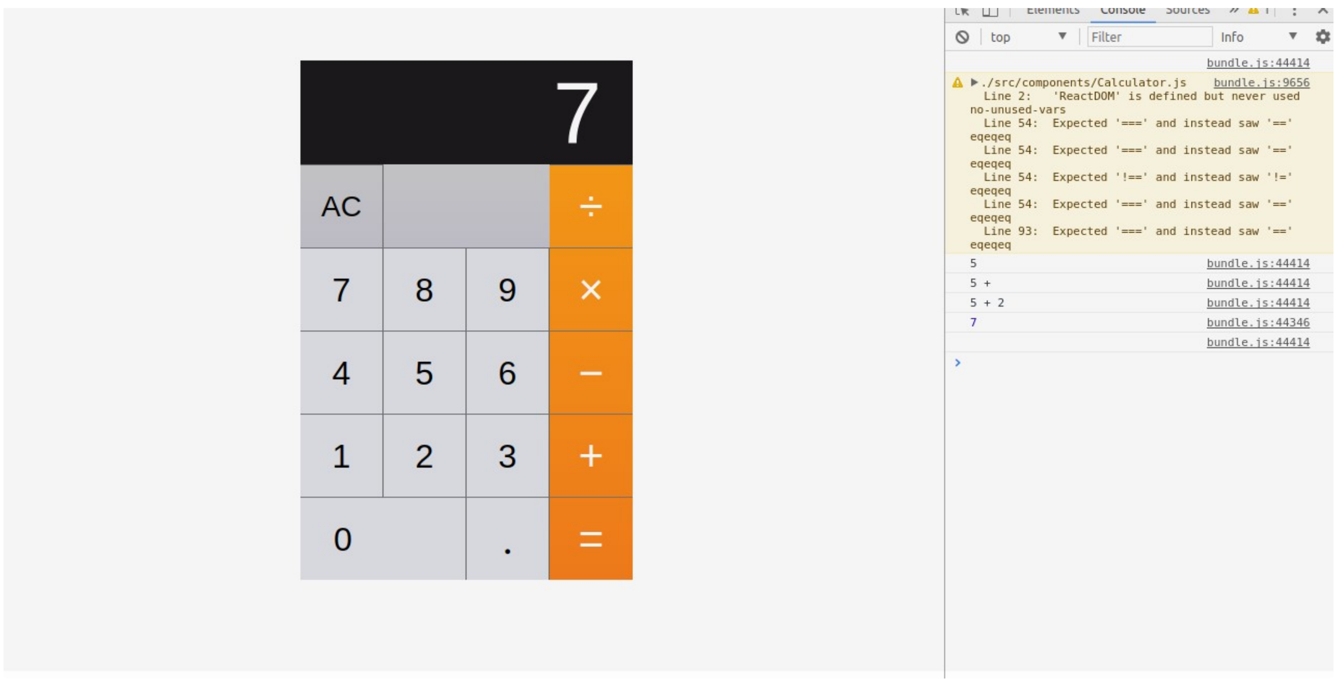
The front-end of the application has been developed using React Js and backed using Node Js. For each computation performed using calculator, React Js server sends a request to Node Js server. Node Js server computes the input string and returns the result back to React Js server which is then displayed on the screen.

API created:

localhost:3000/calcuate

Screenshots :





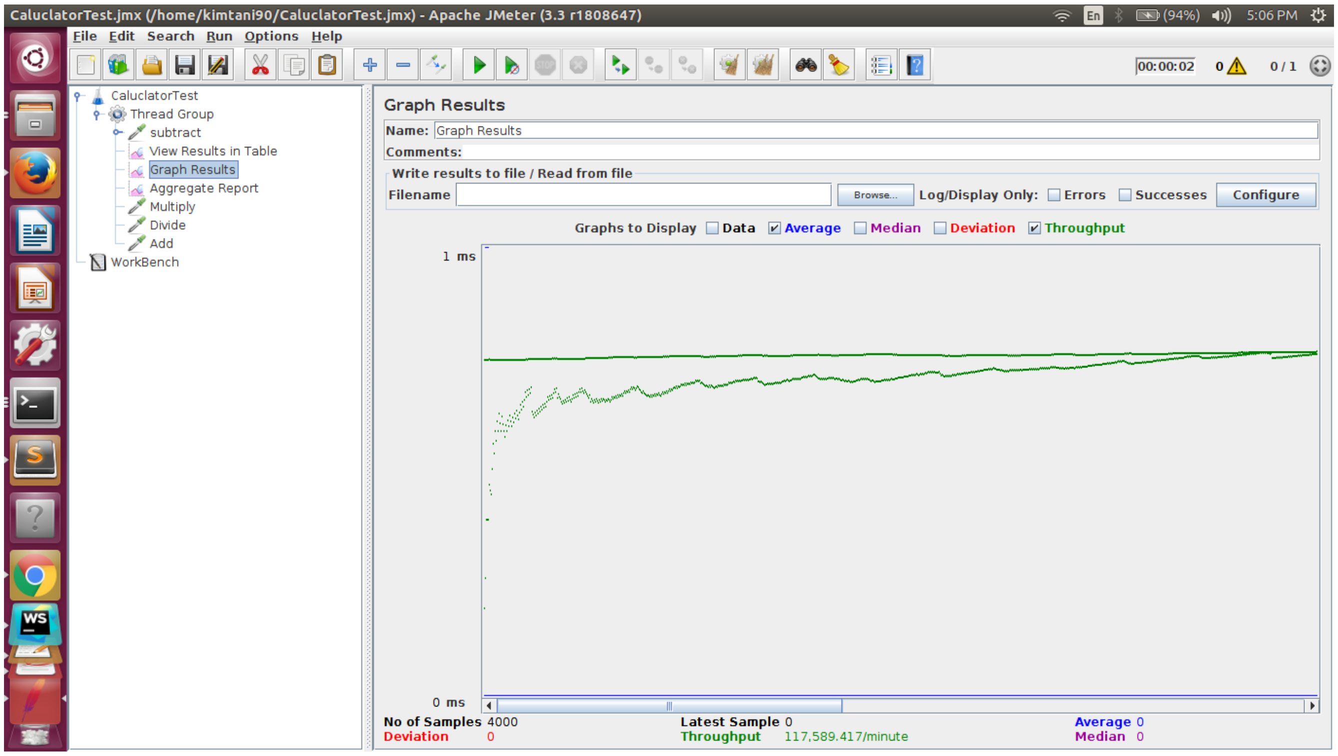
```
                                        bundle.js:44414
▶./src/components/Calculator.js     bundle.js:9656
  Line 2:   'ReactDOM' is defined but never used
no-unused-vars
  Line 54:  Expected '===' and instead saw '=='
eqeqeq
  Line 54:  Expected '===' and instead saw '=='
eqeqeq
  Line 54:  Expected '!==' and instead saw '!='
eqeqeq
  Line 54:  Expected '===' and instead saw '=='
eqeqeq
  Line 93:  Expected '===' and instead saw '=='
eqeqeq
5                                       bundle.js:44414
5 +                                     bundle.js:44414
5 + 2                                   bundle.js:44414
7                                       bundle.js:44346
                                        bundle.js:44414
>
```

Performance :

JMeter Testing :

1. For 1000 requests on Calculator server, the JMeter graph is shown as below.

   The Average time for 1000 concurrent requests is approximately 1ms which is a good performance.



CaluclatorTest.jmx (/home/kimtani90/CaluclatorTest.jmx) - Apache JMeter (3.3 r1808647)

Graph Results

Name: Graph Results

Comments:

Write results to file / Read from file

Filename

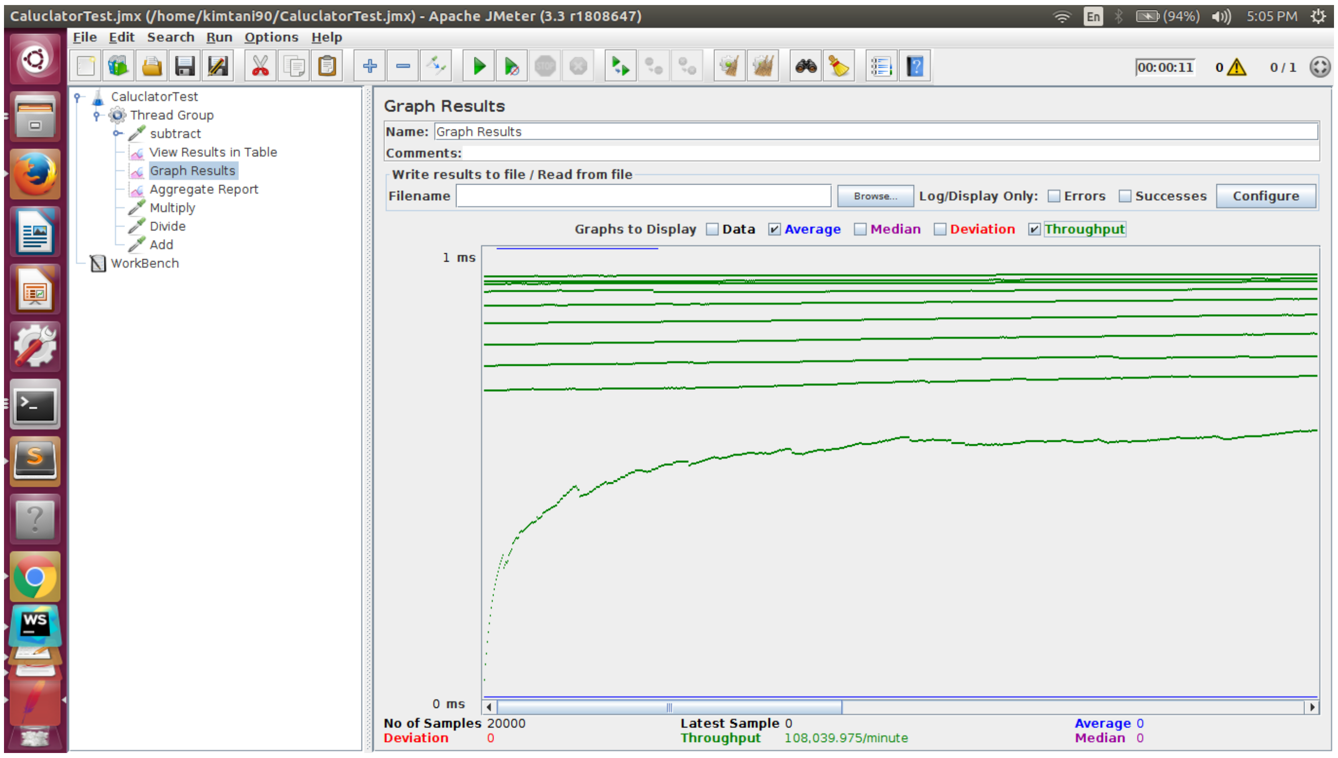
Log/Display Only: ☐ Errors ☐ Successes

Graphs to Display ☐ Data ☑ Average ☐ Median ☐ Deviation ☑ Throughput

```
No of Samples 4000          Latest Sample 0                        Average 0
Deviation     0             Throughput    117,589.417/minute       Median  0
```

2. For 5000 requests on Calculator server, the JMeter graph is shown as below.

   The Average time for 5000 concurrent requests is approximately 1ms which is a good performance.



Name: Graph Results

Comments:

Write results to file / Read from file

Filename

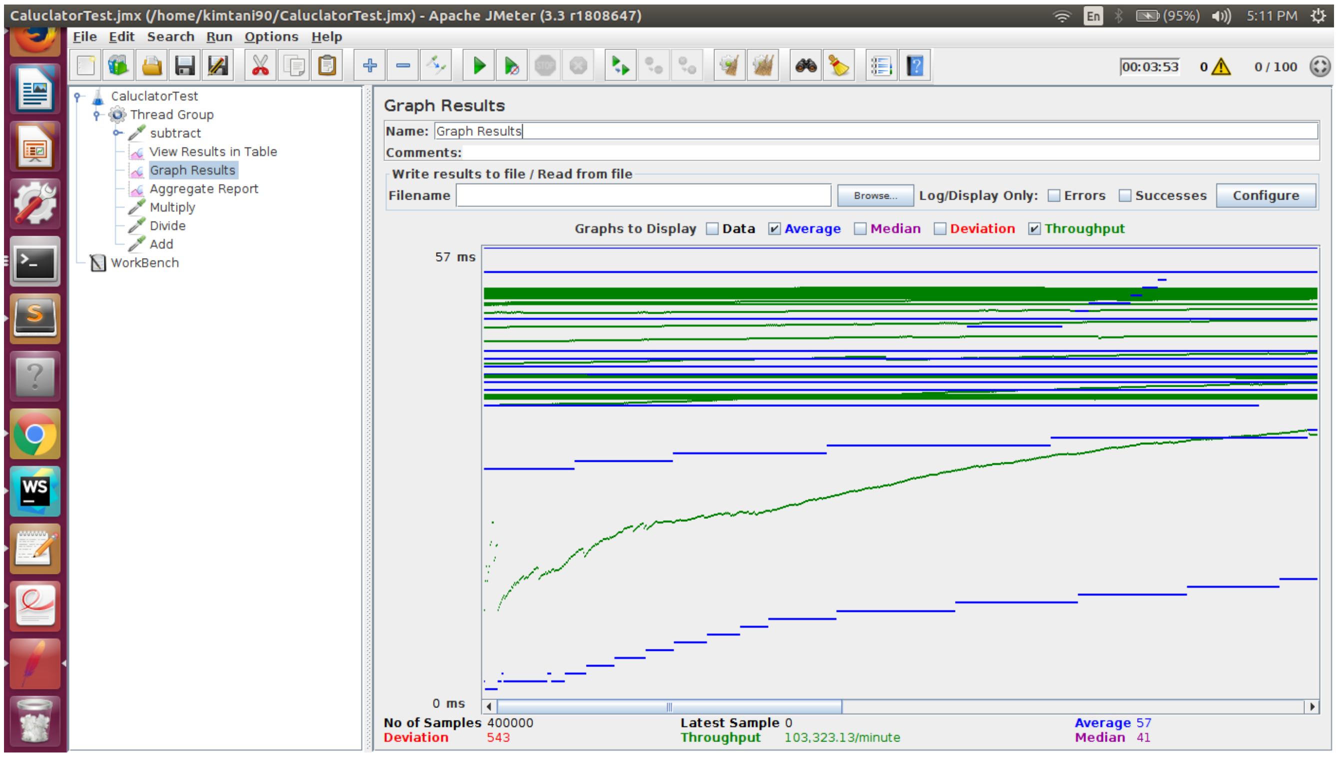
Log/Display Only: ☐ Errors ☐ Successes

Graphs to Display ☐ Data ☑ Average ☐ Median ☐ Deviation ☑ Throughput

No of Samples 20000
Deviation 0
Latest Sample 0
Throughput 108,039.975/minute
Average 0
Median 0

3. For 100 users sending 1000 requests to the Calculator application, the Jmeter graph is shown as below.

The Average time for this scenario is 57 ms approximately which is quite high compared to previous 2 scenarios.



Graph Results

Name: Graph Results

Comments:

Write results to file / Read from file

Filename Log/Display Only: ☐ Errors ☐ Successes

Graphs to Display ☐ Data ☑ Average ☐ Median ☐ Deviation ☑ Throughput

No of Samples 400000
Deviation 543
Latest Sample 0
Throughput 103,323.13/minute
Average 57
Median 41

<u>Dropbox Web Application</u>

Goal :

The goal is to develop a Dropbox web application which has functionalities similar to actual Dropbox and demonstrate how “React Js” client with Redux interacts with a “Node Js” server and MySql.

Purpose :

The purpose of this assignment is to understand and implement the basic concepts of stateless connection , SQL connection pooling, encryption and understanding the concepts of Node Js framework , JMeter , React Js, Redux.

System Design :

The front-end of the application has been developed using React Js and backed using Node Js. For each operation performed by the application, React Js server sends a request to Node Js server. Node Js server performs the action and returns the result back to React Js server which is then displayed on the screen.

In addition, a Redux store is used to store the client state which maintains a centralized repository for the all the React Js components and improves the performance of the application.

The system also handles the session management, for that express-sessions has been installed as the middleware.

The encryption algorithm used for the application is Salt and Hash.

APIs created:

1. User Sign Up:
   Method: Post
   URL: localhost:3000/users/signup

2. User Login:
   Method: Post
   URL: localhost:3000/users

3. Get User Details:
   Method: Get
   URL: localhost:3000/users/signup

4. User Update:
   Method: Post
   URL: localhost:3000/users/userupdate

5. File Upload:
   Method: Post
   URL: localhost:3000/files/upload

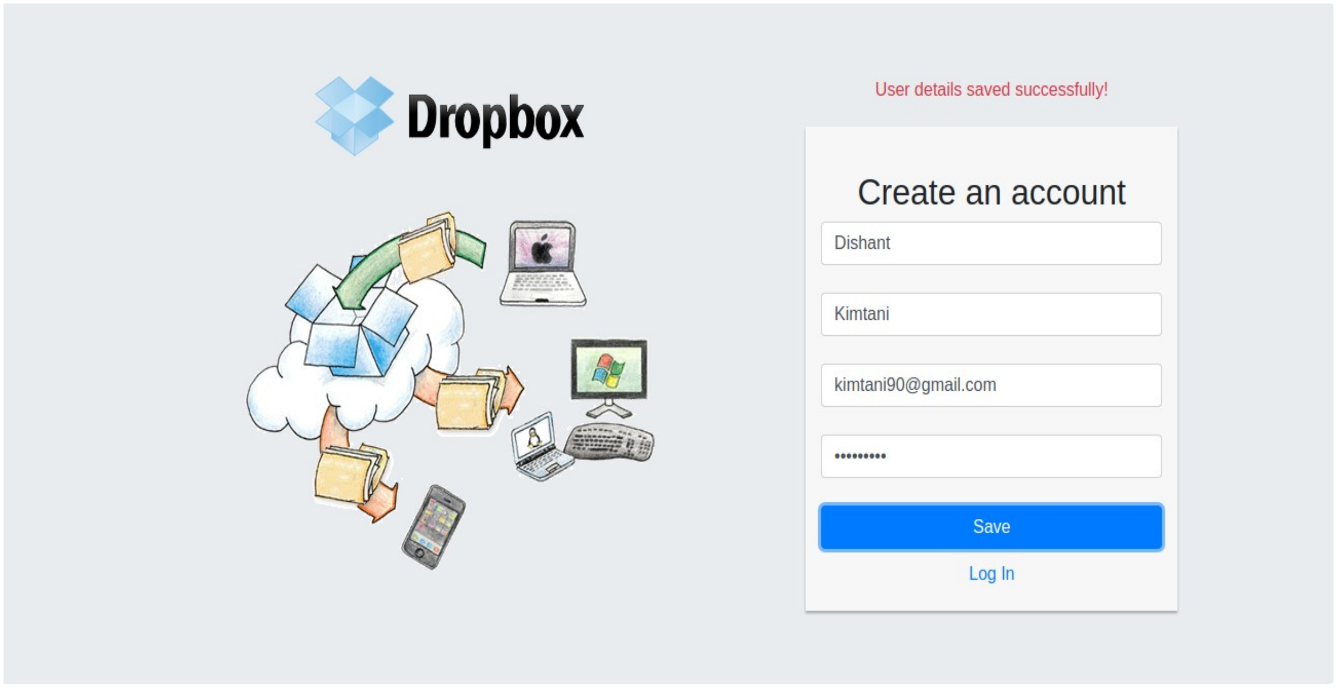
6. Delete File:
   Method: Post
   URL: localhost:3000/files/delete

7. Make Folder:
   Method: Post
   URL: localhost:3000/files/makefolder

8. Share File:
   Method: Post
   URL: localhost:3000/sharefile

Screenshots :

User SignUp Page



User details saved successfully!

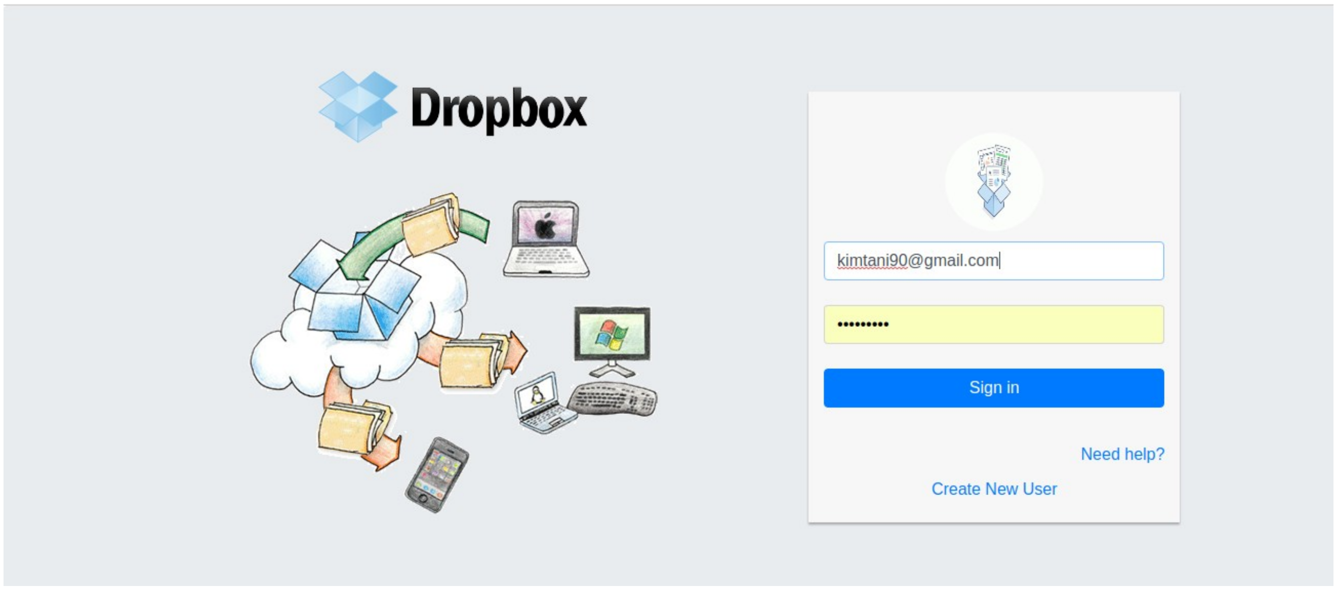
Dishant

Kimtani

kimtani90@gmail.com

•••••••••

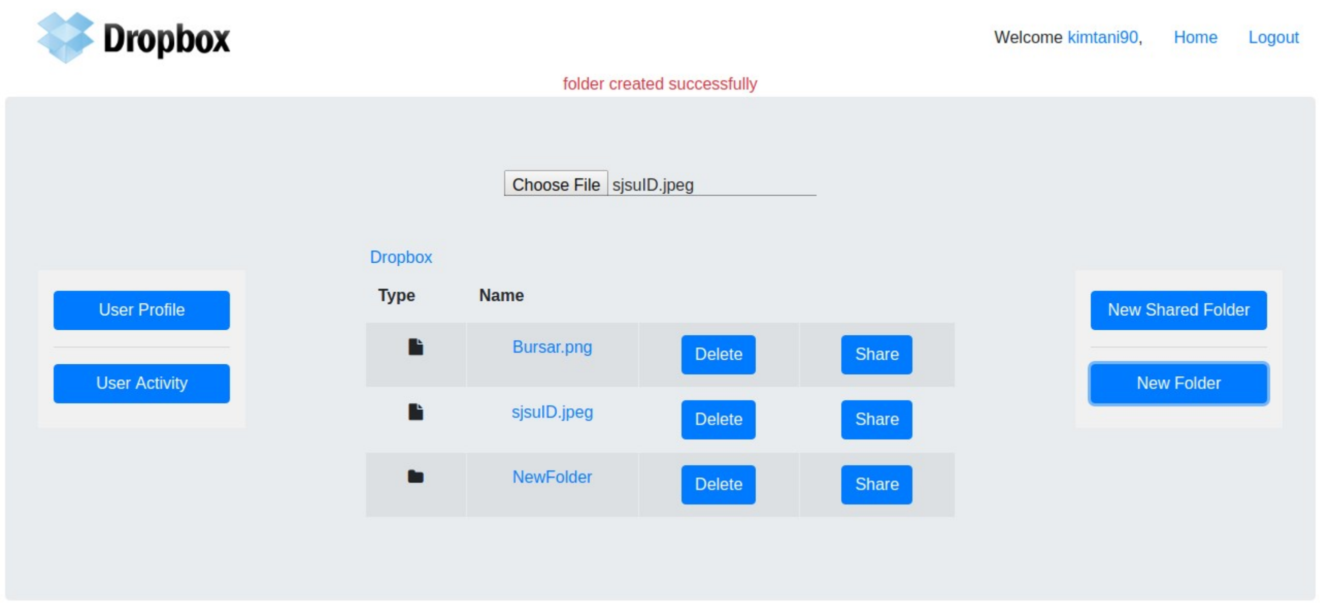
User Login Page



kimtani90@gmail.com

•••••••••

Click on New Folder button, a pop-up opens. Enter folder name and a folder is created and listed.

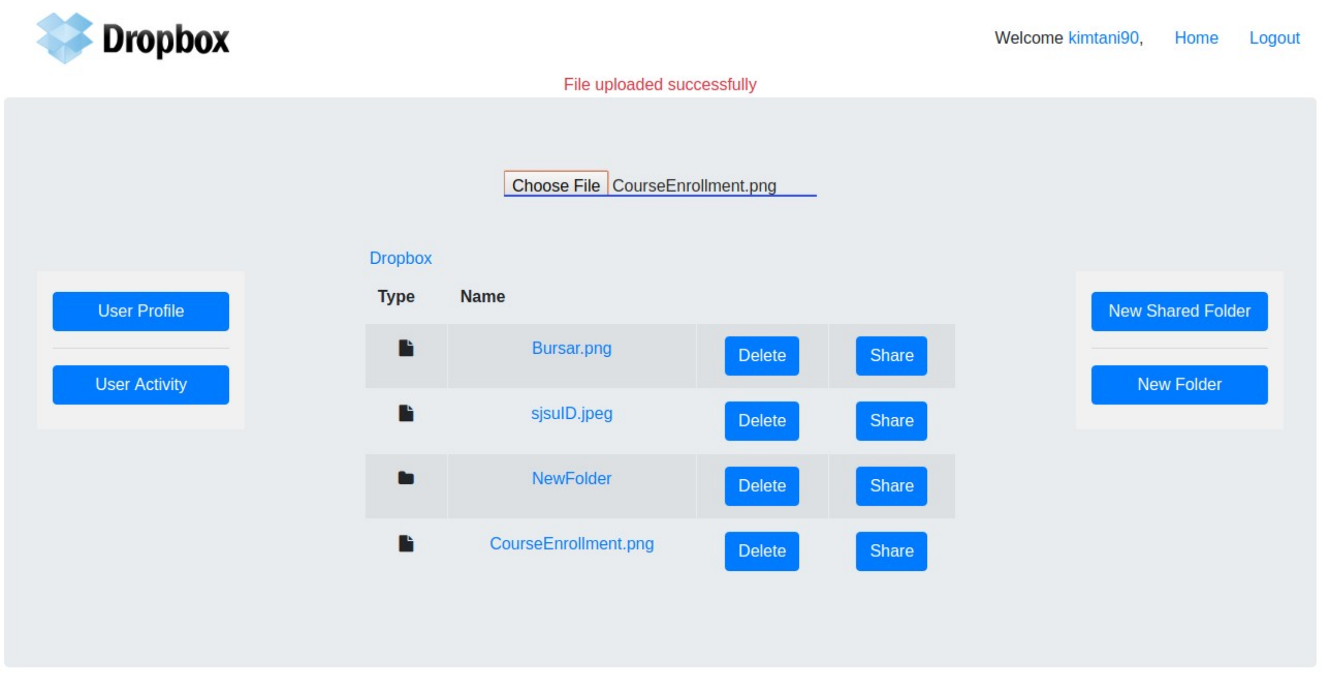


folder created successfully

Dropbox

Type	Name		
🗎	Bursar.png	Delete	Share
🗎	sjsuID.jpeg	Delete	Share
📁	NewFolder	Delete	Share

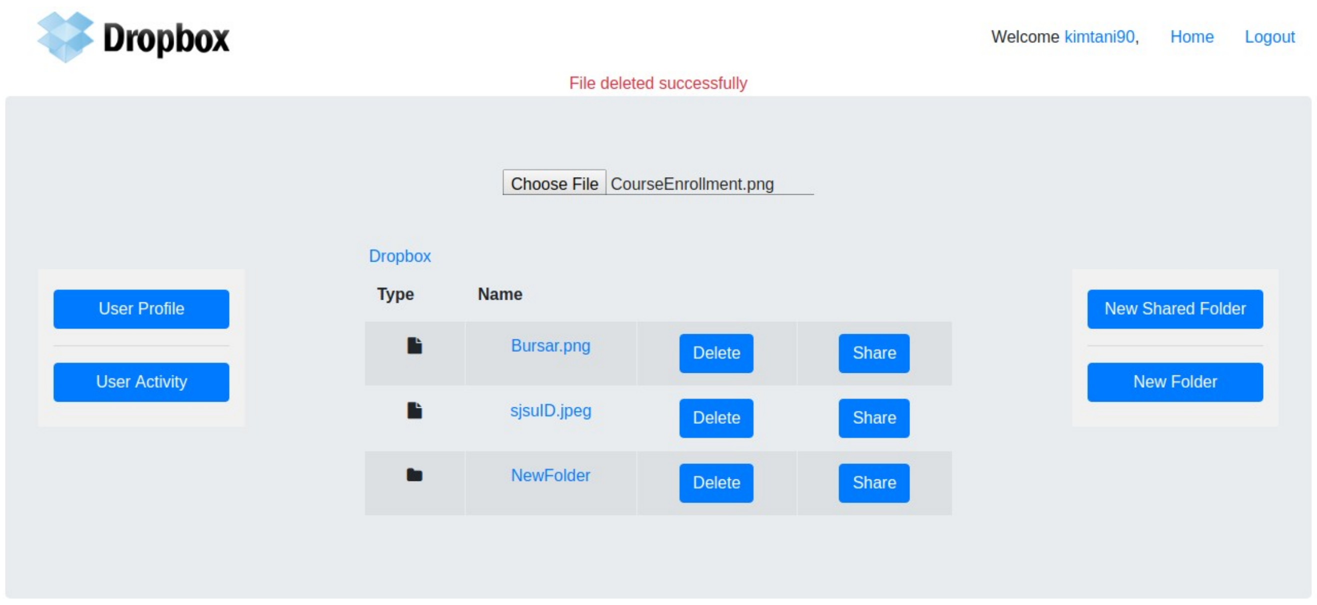
Click on choose file and upload it. The file gets listed in the file list.


Dropbox
Welcome kimtani90, Home Logout
File uploaded successfully
Choose File CourseEnrollment.png
User Profile
User Activity
New Shared Folder
New Folder

Dropbox

Type	Name		
	Bursar.png	Delete	Share
	sjsuID.jpeg	Delete	Share
	NewFolder	Delete	Share
	CourseEnrollment.png	Delete	Share

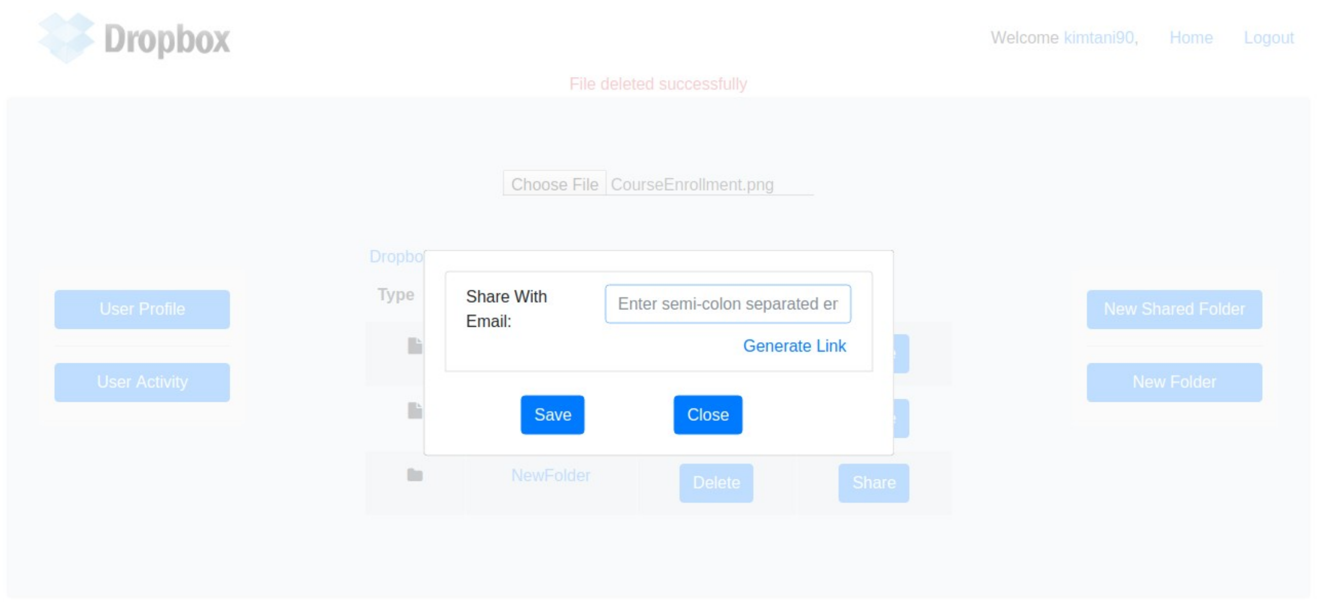
Click on delete button corresponding to a file or folder and the file gets deleted.

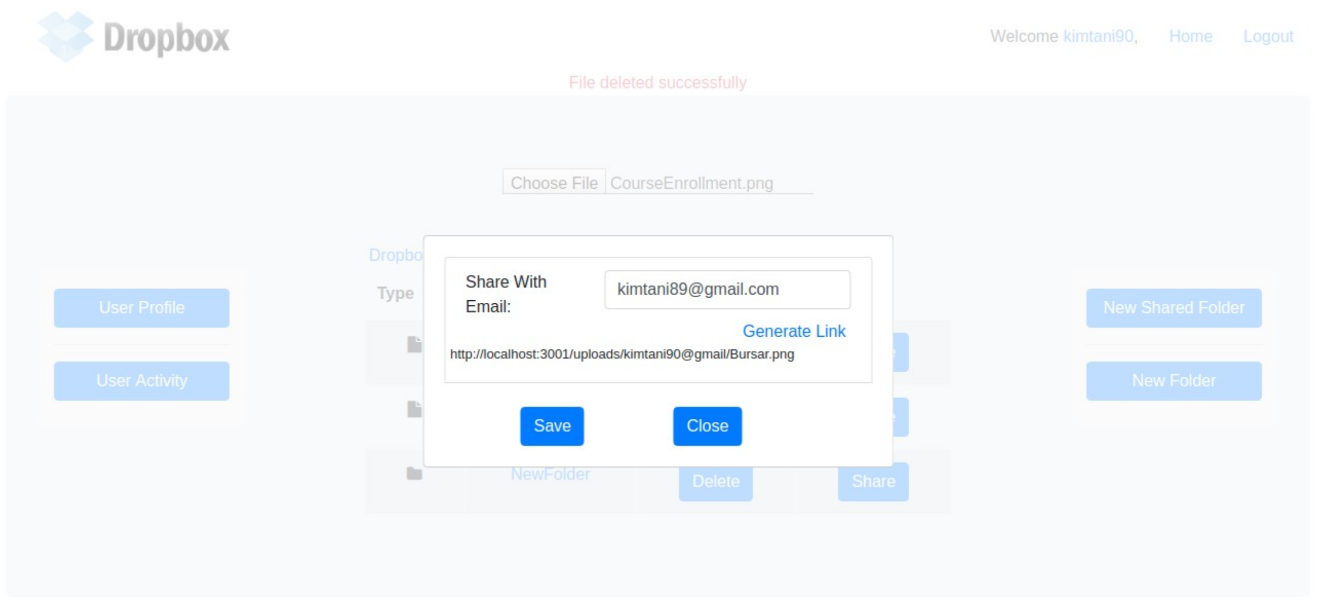

Dropbox
Welcome kimtani90, Home Logout
File deleted successfully
Choose File CourseEnrollment.png
User Profile
User Activity
New Shared Folder
New Folder

Dropbox

Type	Name		
	Bursar.png	Delete	Share
	sjsuID.jpeg	Delete	Share
	NewFolder	Delete	Share

Click on share button corresponding to a file, a pop-up opens. Enter a list of emails separated bu semi-colon with which user wants to share the file.

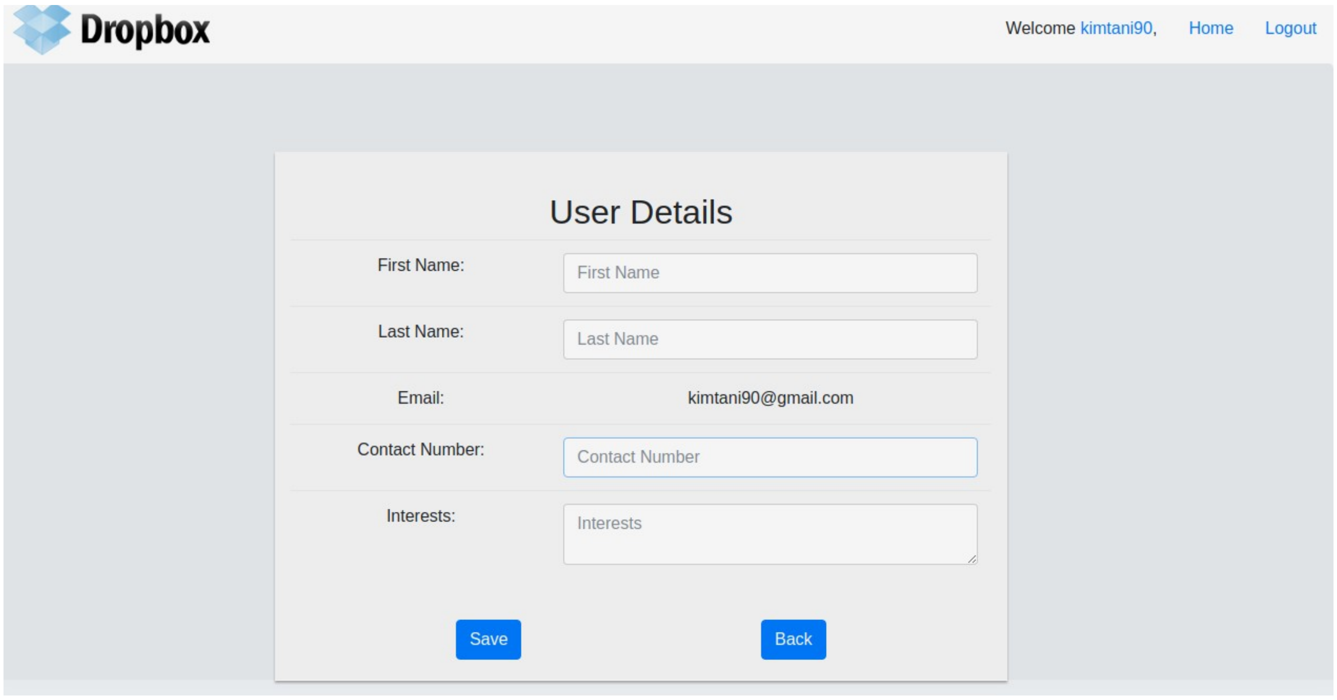




kimtani89@gmail.com

http://localhost:3001/uploads/kimtani90@gmail/Bursar.png

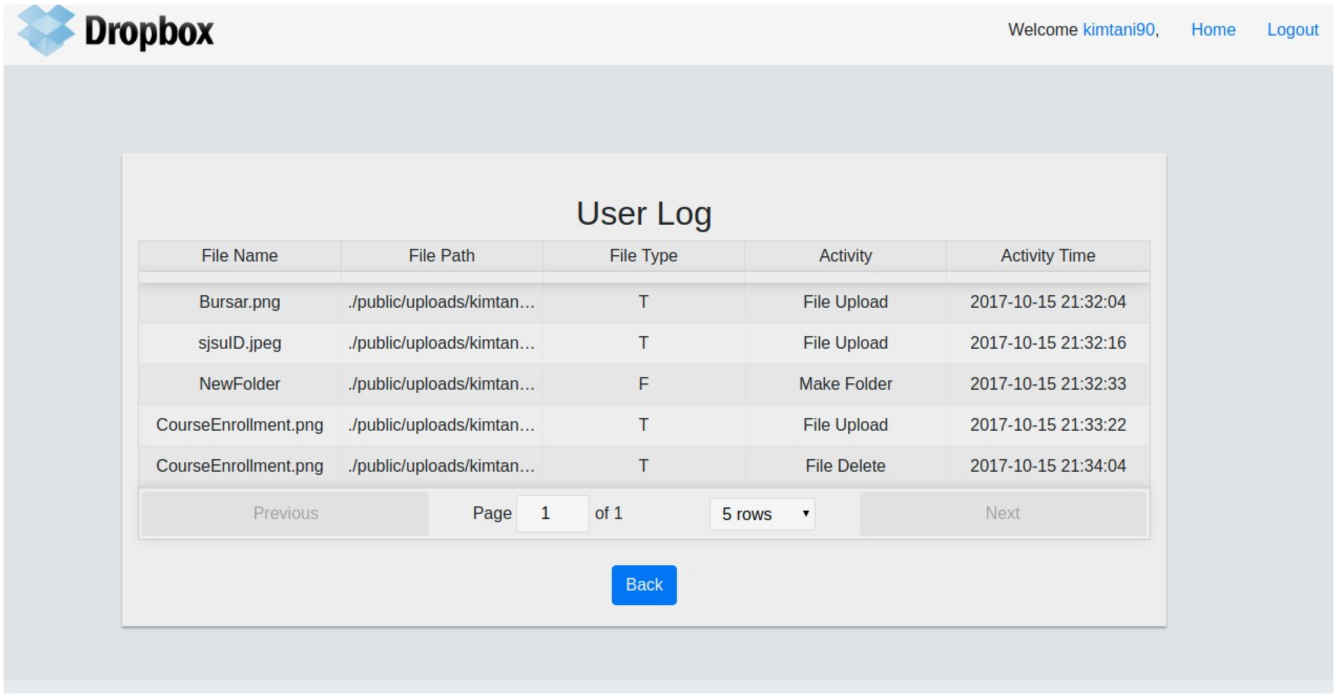
Click on user profile page to navigate to User Details page as shown below



User Details

First Name:	First Name
Last Name:	Last Name
Email:	kimtani90@gmail.com
Contact Number:	Contact Number
Interests:	Interests

Click on User Activity button to navigate to User Log page



User Log

File Name	File Path	File Type	Activity	Activity Time
Bursar.png	./public/uploads/kimtan…	T	File Upload	2017-10-15 21:32:04
sjsuID.jpeg	./public/uploads/kimtan…	T	File Upload	2017-10-15 21:32:16
NewFolder	./public/uploads/kimtan…	F	Make Folder	2017-10-15 21:32:33
CourseEnrollment.png	./public/uploads/kimtan…	T	File Upload	2017-10-15 21:33:22
CourseEnrollment.png	./public/uploads/kimtan…	T	File Delete	2017-10-15 21:34:04

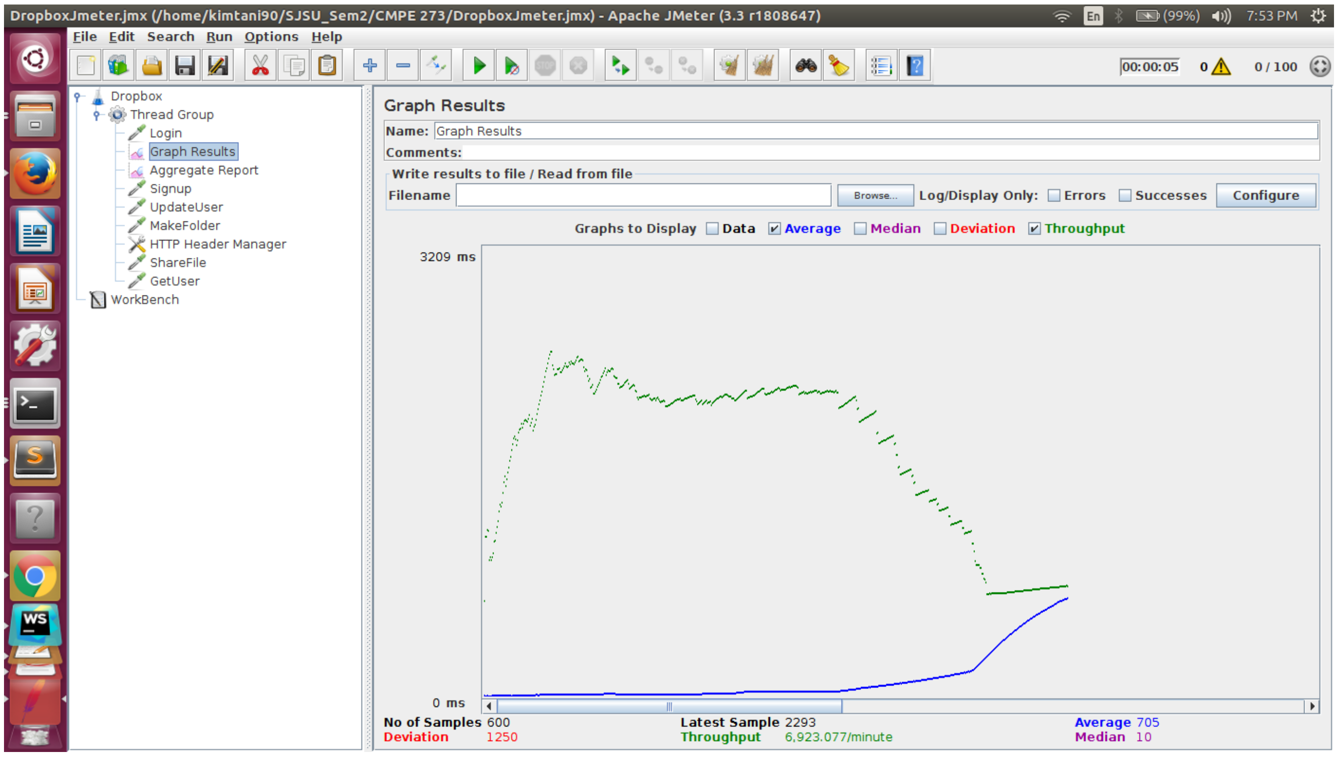
Performance :

JMeter Testing:

The Performance of the system was good considering the amount of concurrent requests it could handle , mysql connection pooling played a significant role in decreasing the average time of serving the request .

Below are the graphs that were created by testing the dropbox Api's with Apache Jmeter

1. For 100 concurrent using connection pooling.


DropboxJmeter.jmx (/home/kimtani90/SJSU_Sem2/CMPE 273/DropboxJmeter.jmx) - Apache JMeter (3.3 r1808647)

Graph Results

Name: Graph Results

Comments:

Write results to file / Read from file

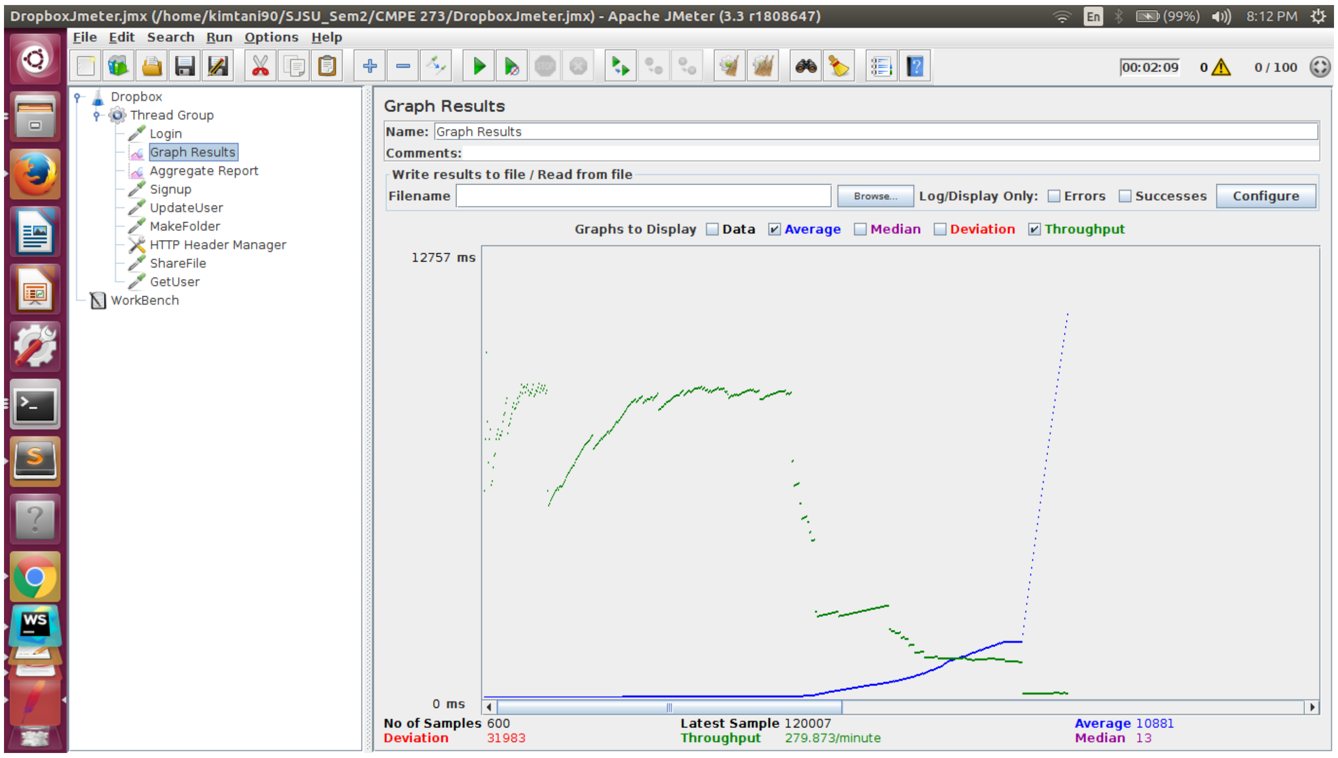
Filename

Log/Display Only: Errors Successes

Graphs to Display: Data [✓ Average] Median Deviation [✓ Throughput]

No of Samples 600
Deviation 1250
Latest Sample 2293
Throughput 6,923.077/minute
Average 705
Median 10

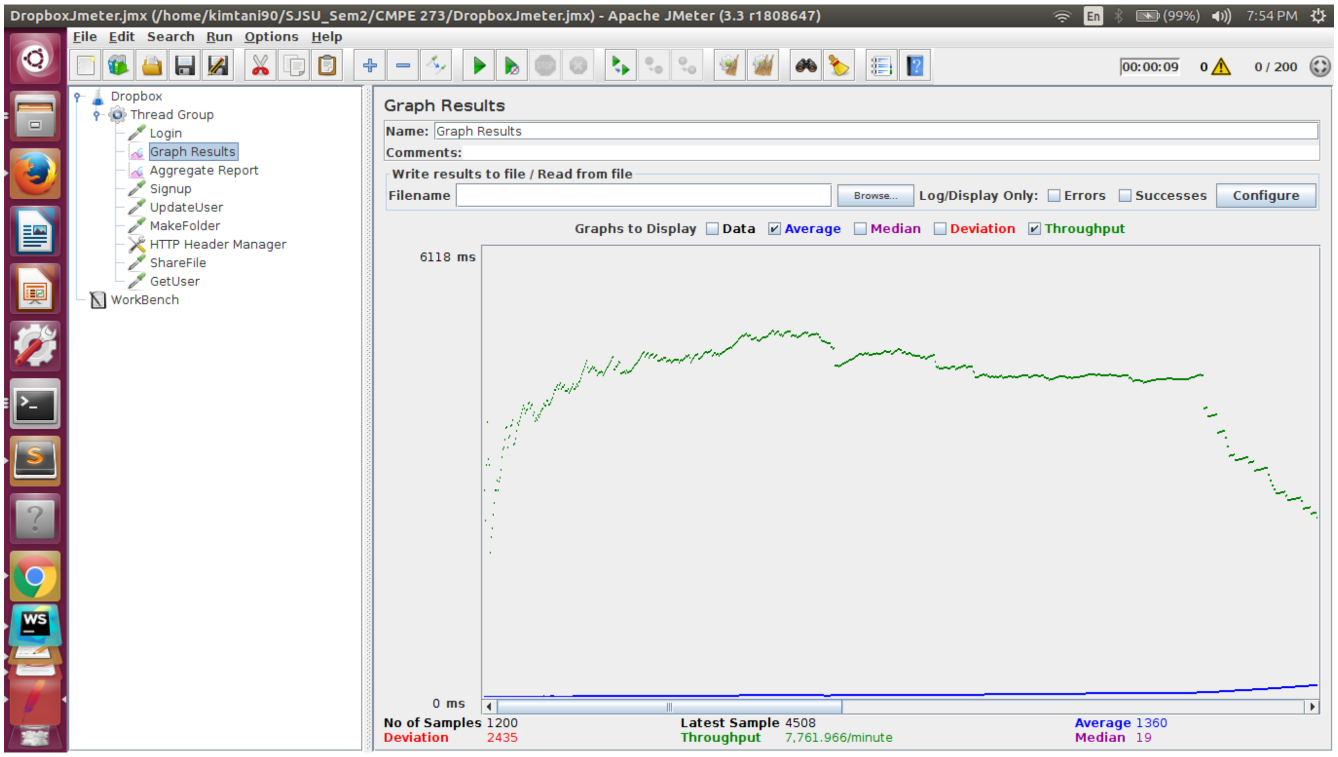
For 100 concurrent users without connection pooling.



Name: Graph Results
Comments:
Write results to file / Read from file
Filename
Log/Display Only: ☐ Errors ☐ Successes
Graphs to Display ☐ Data ☑ Average ☐ Median ☐ Deviation ☑ Throughput

No of Samples 600
Deviation 31983
Latest Sample 120007
Throughput 279.873/minute
Average 10881
Median 13

2. For 200 concurrent users with connection pooling.


DropboxJmeter.jmx (/home/kimtani90/SJSU_Sem2/CMPE 273/DropboxJmeter.jmx) - Apache JMeter (3.3 r1808647)

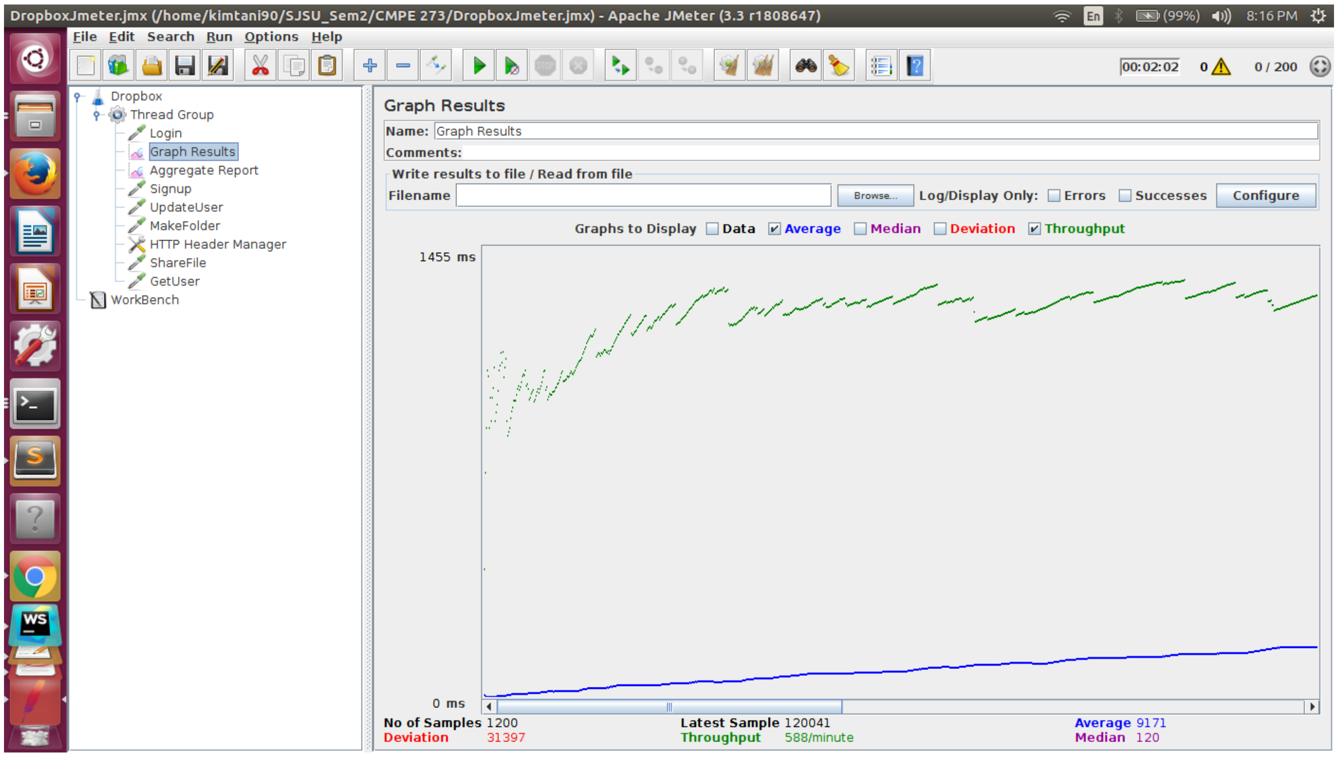
Graph Results

Name: Graph Results

Graphs to Display: ☐ Data ☑ Average ☐ Median ☐ Deviation ☑ Throughput

No of Samples 1200 | Latest Sample 4508 | Average 1360
Deviation 2435 | Throughput 7,761.966/minute | Median 19

For 200 concurrent users without connection pooling.


DropboxJmeter.jmx (/home/kimtani90/SJSU_Sem2/CMPE 273/DropboxJmeter.jmx) - Apache JMeter (3.3 r1808647)

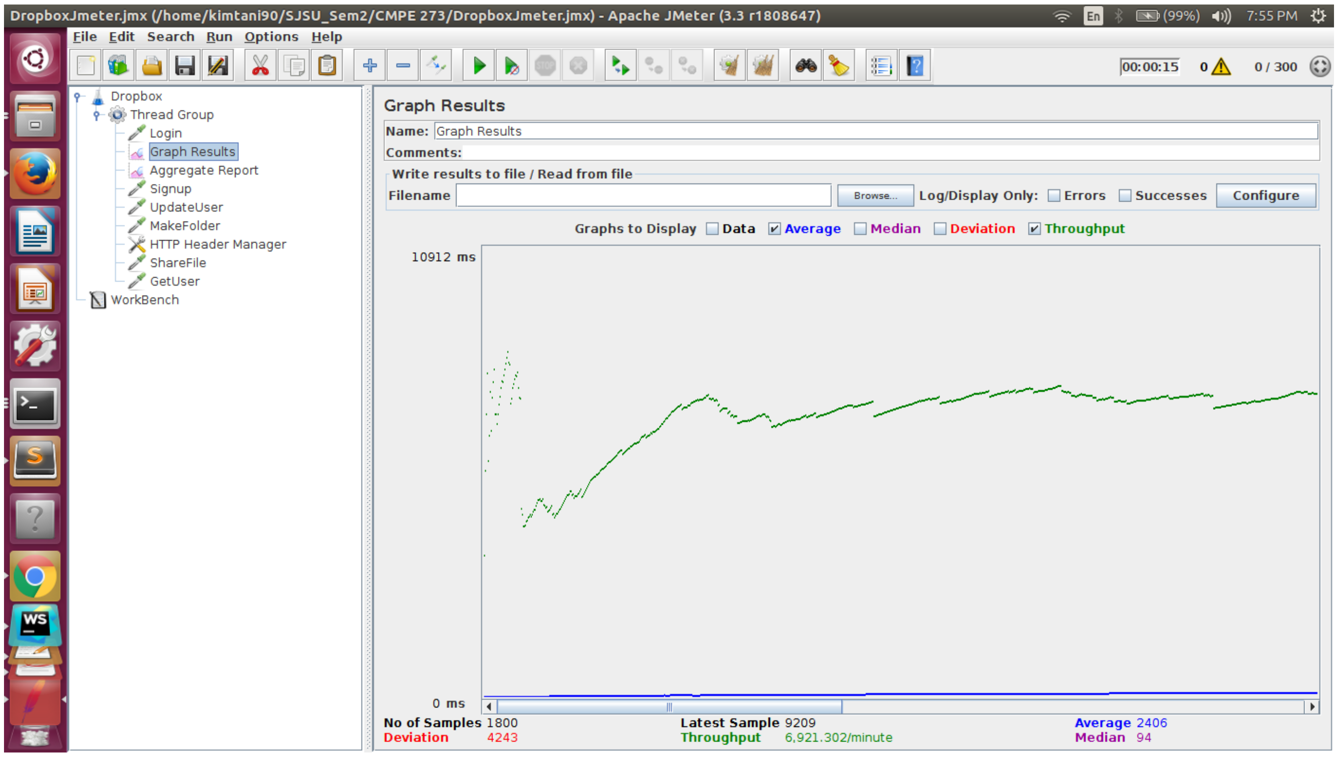
Graph Results

Name: Graph Results

Graphs to Display: ☐ Data ☑ Average ☐ Median ☐ Deviation ☑ Throughput

No of Samples 1200 | Latest Sample 120041 | Average 9171
Deviation 31397 | Throughput 588/minute | Median 120

3. For 300 concurrent users with connection pooling.



DropboxJmeter.jmx (/home/kimtani90/SJSU_Sem2/CMPE 273/DropboxJmeter.jmx) - Apache JMeter (3.3 r1808647)

00:00:15 0 0 / 300

Graph Results

Name: Graph Results

Comments:

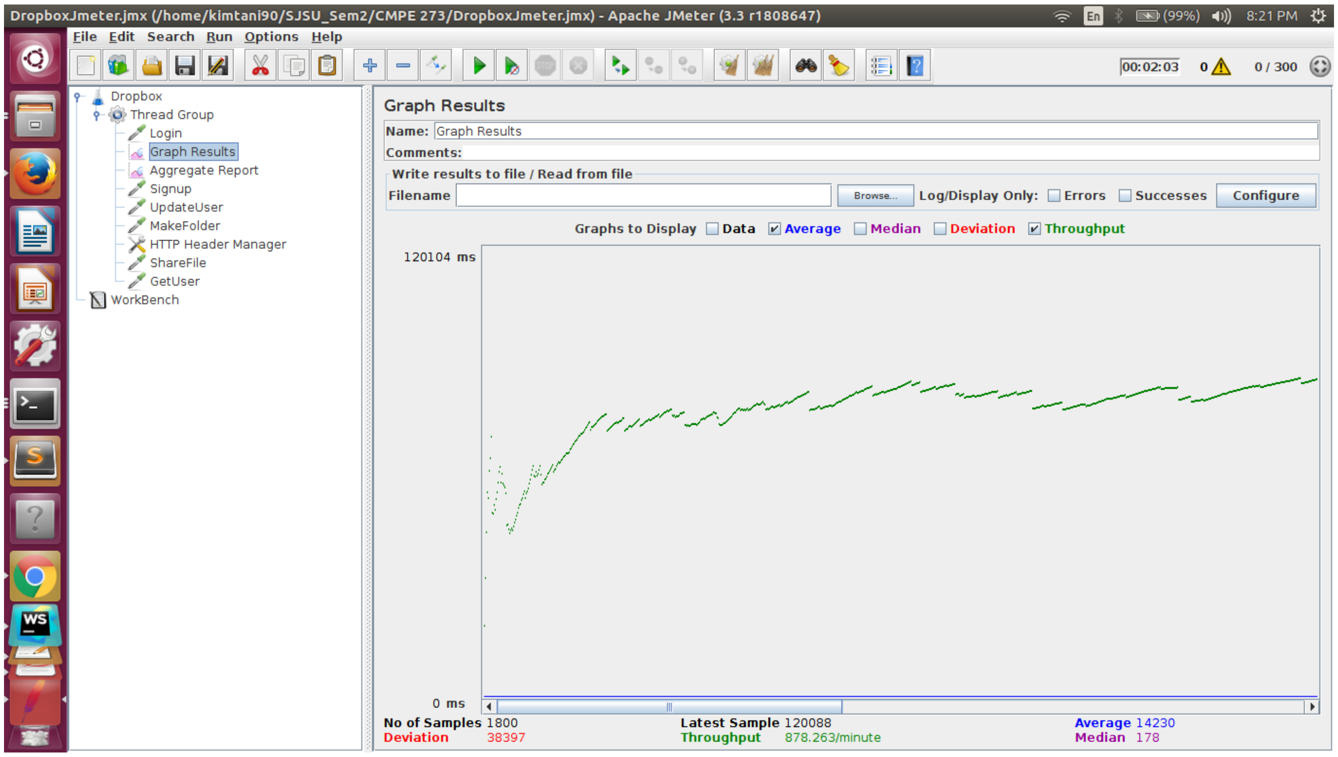
Write results to file / Read from file

Filename Log/Display Only: ☐ Errors ☐ Successes

Graphs to Display ☐ Data ☑ Average ☐ Median ☐ Deviation ☑ Throughput

No of Samples 1800 Latest Sample 9209 Average 2406
Deviation 4243 Throughput 6,921.302/minute Median 94

For 300 concurrent users without connection pooling.



DropboxJmeter.jmx (/home/kimtani90/SJSU_Sem2/CMPE 273/DropboxJmeter.jmx) - Apache JMeter (3.3 r1808647)

00:02:03 0 0 / 300

Graph Results

Name: Graph Results

Comments:

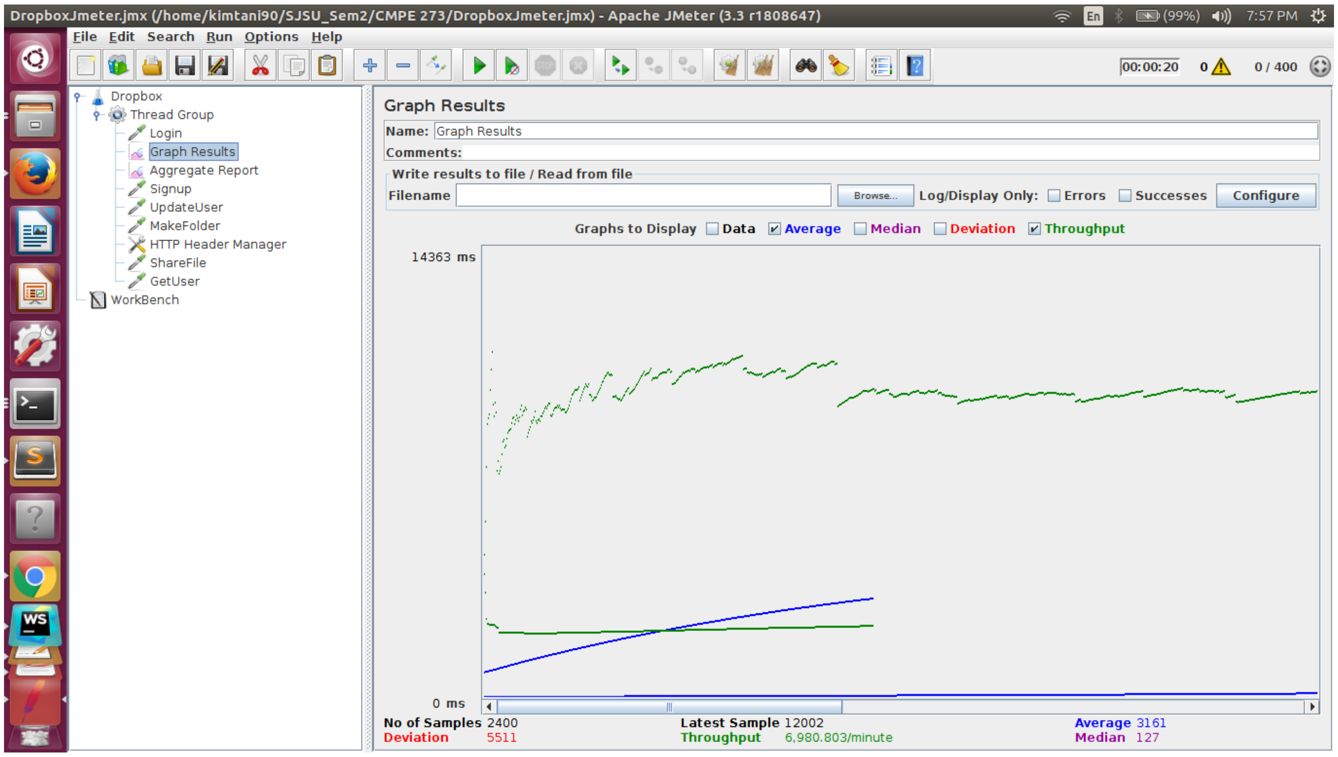
Write results to file / Read from file

Filename Log/Display Only: ☐ Errors ☐ Successes

Graphs to Display ☐ Data ☑ Average ☐ Median ☐ Deviation ☑ Throughput

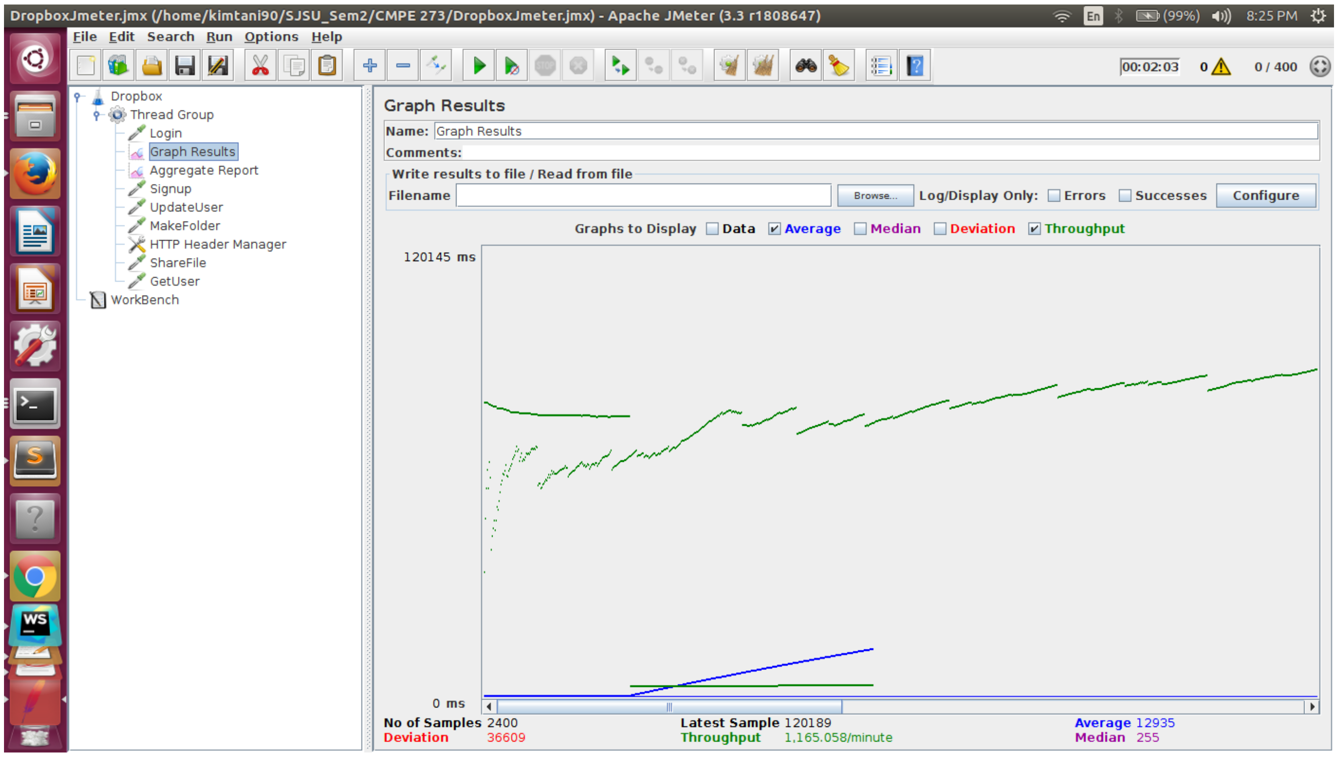
No of Samples 1800 Latest Sample 120088 Average 14230
Deviation 38397 Throughput 878.263/minute Median 178

4. For 400 concurrent users with connection pooling.


DropboxJmeter.jmx (/home/kimtani90/SJSU_Sem2/CMPE 273/DropboxJmeter.jmx) - Apache JMeter (3.3 r1808647)
File Edit Search Run Options Help
00:00:20 0 0 / 400
Dropbox
Thread Group
Login
Graph Results
Aggregate Report
Signup
UpdateUser
MakeFolder
HTTP Header Manager
ShareFile
GetUser
WorkBench
Graph Results
Name: Graph Results
Comments:
Write results to file / Read from file
Filename
Browse...
Log/Display Only: Errors Successes
Configure
Graphs to Display Data Average Median Deviation Throughput
14363 ms
0 ms

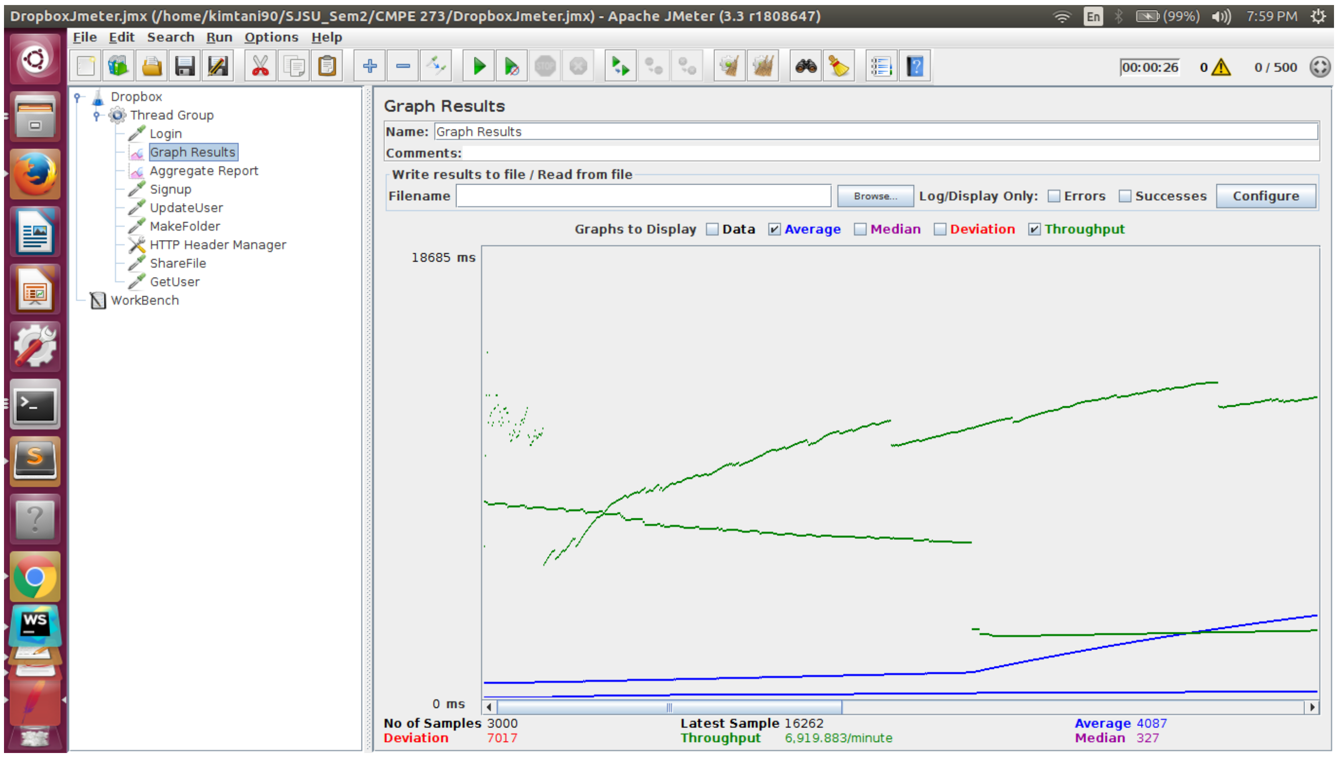
No of Samples 2400
Deviation 5511
Latest Sample 12002
Throughput 6,980.803/minute
Average 3161
Median 127

For 400 concurrent users without connection pooling.


DropboxJmeter.jmx (/home/kimtani90/SJSU_Sem2/CMPE 273/DropboxJmeter.jmx) - Apache JMeter (3.3 r1808647)
File Edit Search Run Options Help
00:02:03 0 0 / 400
Dropbox
Thread Group
Login
Graph Results
Aggregate Report
Signup
UpdateUser
MakeFolder
HTTP Header Manager
ShareFile
GetUser
WorkBench
Graph Results
Name: Graph Results
Comments:
Write results to file / Read from file
Filename
Browse...
Log/Display Only: Errors Successes
Configure
Graphs to Display Data Average Median Deviation Throughput
120145 ms
0 ms

No of Samples 2400
Deviation 36609
Latest Sample 120189
Throughput 1,165.058/minute
Average 12935
Median 255

5. For 500 concurrent users with connection pooling.


DropboxJmeter.jmx (/home/kimtani90/SJSU_Sem2/CMPE 273/DropboxJmeter.jmx) - Apache JMeter (3.3 r1808647)

Graph Results

Name: Graph Results

Comments:

Write results to file / Read from file

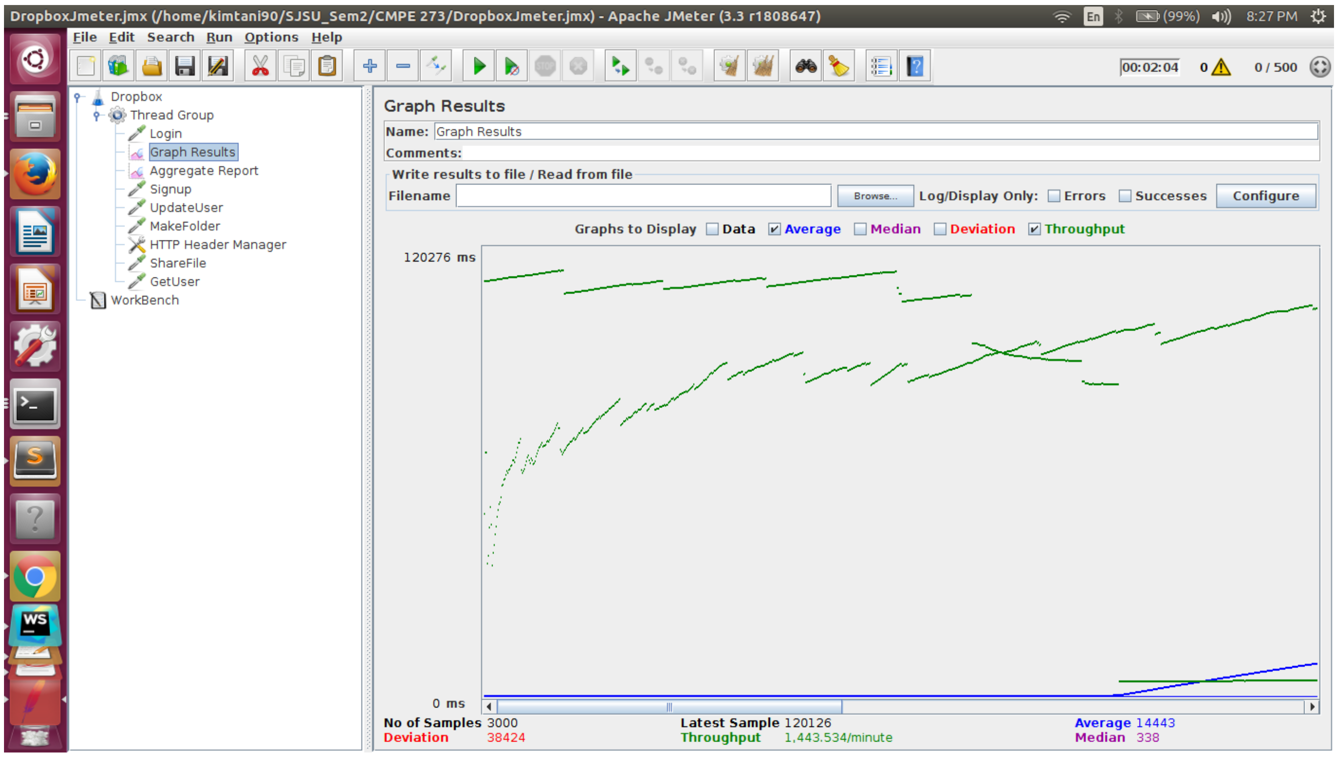
Filename

Log/Display Only: Errors Successes

Graphs to Display: Data [✓ Average] Median Deviation [✓ Throughput]

No of Samples 3000
Deviation 7017
Latest Sample 16262
Throughput 6,919.883/minute
Average 4087
Median 327

For 500 concurrent users without connection pooling.


DropboxJmeter.jmx (/home/kimtani90/SJSU_Sem2/CMPE 273/DropboxJmeter.jmx) - Apache JMeter (3.3 r1808647)
File Edit Search Run Options Help
00:02:04 0 0 / 500
Dropbox
Thread Group
Login
Graph Results
Aggregate Report
Signup
UpdateUser
MakeFolder
HTTP Header Manager
ShareFile
GetUser
WorkBench
Graph Results
Write results to file / Read from file
Browse...
Log/Display Only: Errors Successes
Configure
Graphs to Display Data Average Median Deviation Throughput
120276 ms
0 ms

Name: Graph Results
Comments:
Filename
Graphs to Display: Data [ ] Average [✓] Median [ ] Deviation [ ] Throughput [✓]

No of Samples 3000
Deviation 38424
Latest Sample 120126
Throughput 1,443.534/minute
Average 14443
Median 338

As seen from the above screenshots the average time is significantly lower, approximately 10 times using connection pooling when compared to testing done using without connection pooling.

Questions:

1. Explain the encryption algorithm used in your application. Mention different encryption algorithms available and the reason for your selection of the algorithm used.

The encryption algorithm used for dropbox application is “Salt and Hash”.

A salt is random data that is used as an additional input to a one-way function that "hashes" a password.. The primary function of salts is to defend against dictionary attacks or against its hashed equivalent, a pre-computed rainbow table attack.

Different encryption algorithms are:

Triple DES
RSA
Blowfish
Twofish

AES
Salt and Hash

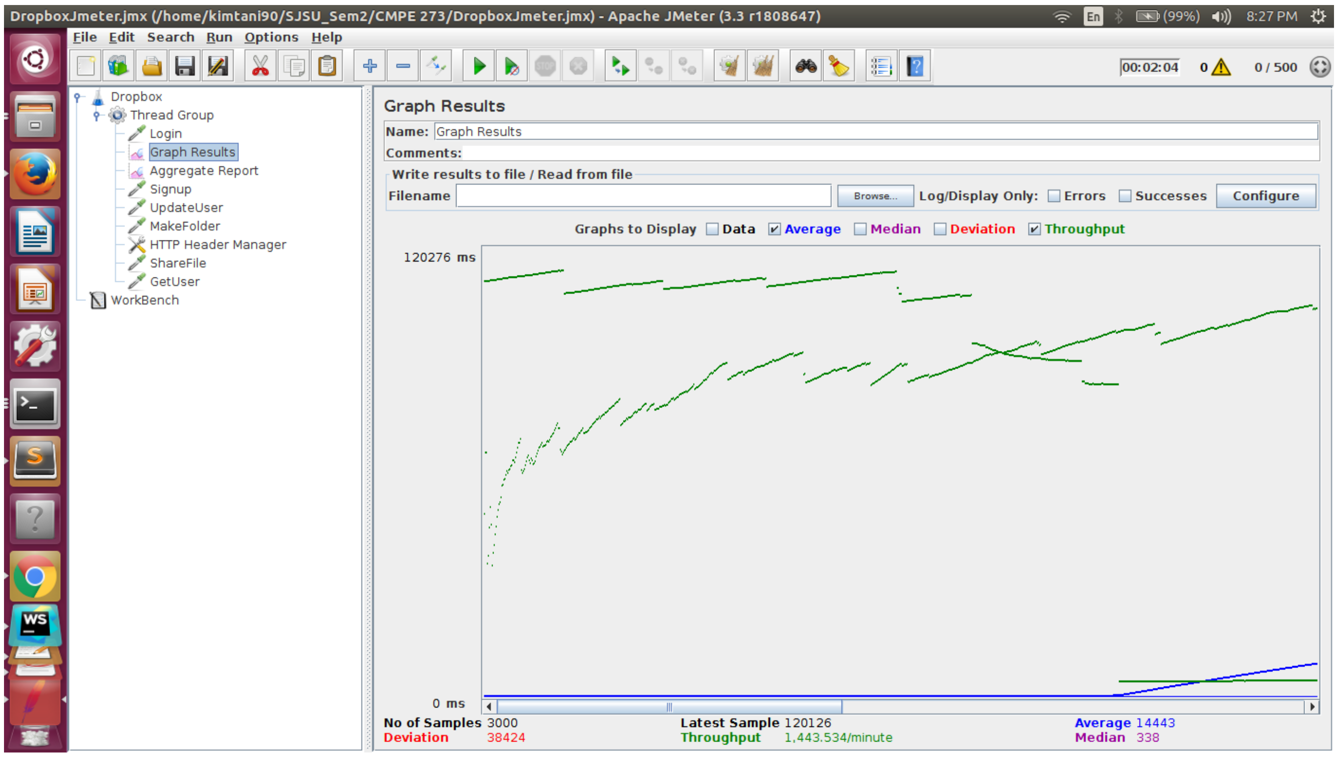
The primary reason to use Salt and Hash is to defend against dictionary attacks or against its hashed equivalent, a pre-computed rainbow table attack.

2. Compare the results of graphs with and without connection pooling of database. Explain the result in detail and describe the connection pooling algorithm used in your code.

MySql inbuilt feature for connection pooling has been used for dropbox application.

```
mysql.createPool({
    host     : 'localhost',
    user     : 'root',
    password : 'kimtani02',
    database : 'cmpe273',
    port    : 3306
});
```

Graphical results when dropbox APIs tested without connection pooling using Jmeter.



Graph Results

Name: Graph Results

Comments:

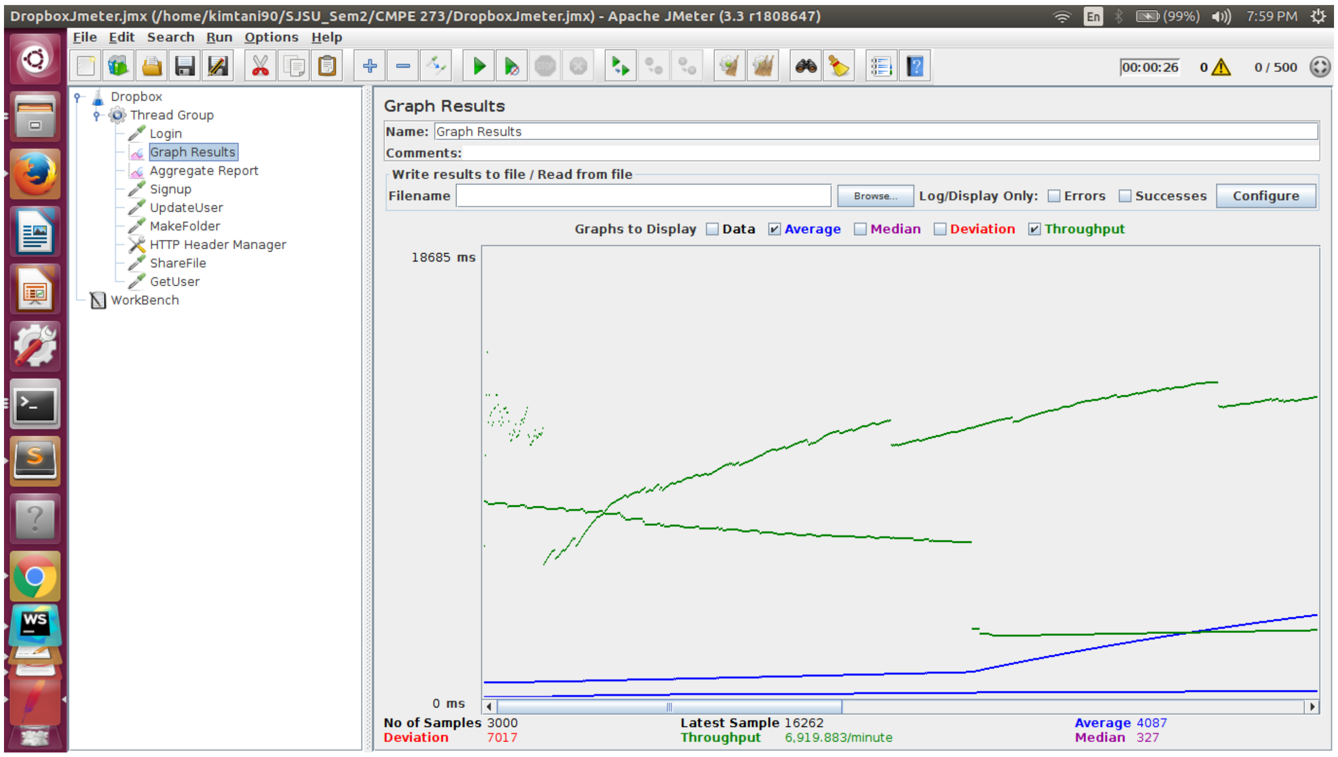
Write results to file / Read from file

Filename | Browse... | Log/Display Only: ☐ Errors ☐ Successes | Configure

Graphs to Display ☐ Data ☑ Average ☐ Median ☐ Deviation ☑ Throughput

```
No of Samples 3000          Latest Sample 120126                  Average 14443
Deviation     38424         Throughput    1,443.534/minute        Median  338
```

Graphical results when dropbox APIs tested without connection pooling using Jmeter.



Graph Results

Name: Graph Results

Comments:

Write results to file / Read from file

Filename | Browse... | Log/Display Only: ☐ Errors ☐ Successes | Configure

Graphs to Display ☐ Data ☑ Average ☐ Median ☐ Deviation ☑ Throughput

```
No of Samples 3000          Latest Sample 16262                   Average 4087
Deviation     7017          Throughput    6,919.883/minute        Median  327
```

The average time when tested without connection pooling is 4007 ms while when tested without connection pooling is 14443 ms. The performance will be about 3 times better when connection pooling is used.

3. What is SQL caching? What all types of SQL caching is available and which suits your code the most. You don’t need to implement the caching, write pseudo code or explain in detail.

SQL caching is a very important task to assure that the server resources are not heavily consumed by a bad performing query. One of the important tasks when SQL caching is to clean the SQL Server Cache so that the results are not influenced by the caching mechanism.

Type of SQL caching:

1. Adhoc query caching
2. Autoparameterization
3. Prepared queries, using either sp_executesql or the prepare and execute method invoked through the API
4. Stored procedures or other compiled objects (triggers, TVFs, etc.)

Prepared queries, using either sp_executesql or the prepare and execute method invoked through the API suits my current structure of application.

4. Is your session strategy horizontally scalable? If YES, explain your session handling strategy. If NO, then explain how can you achieve it.

No , The application is currently not horizontally scalable. Even though there is session management implemented, but there is no database at the back-end to support horizontal scaling.

In order to achieve horizontal scaling all that needs to be done is to store the session data back in the shared database instead of the server memory, by doing so we can achieve horizontal scaling, so if a new server is added all that needs to be done is to connect it to that common database and the session data can be retrieved from there which will handle the statelessness as well as session management. To increase the session management, all that needs to be done is to enable the stickiness on the load balancer to route the requests so that the consecutive requests will be routed to the same server.