San Jose State University Fall 2017

CMPE 273-Lab 2

Dropbox Web Application

SUBMITTED BY -

Dishant Kimtani

SUBMITTED TO-

Prof. Simon Shim

Dropbox Web Application

Goal :

The goal of Lab 2 is to develop a Dropbox web application which has functionalities similar to actual Dropbox, demonstrate how “React JS” client with Redux interacts with a “Node Js” server, implement connection pooling for MongoDB database and use Apache Kafka for making the application scalable, fault tolerant.

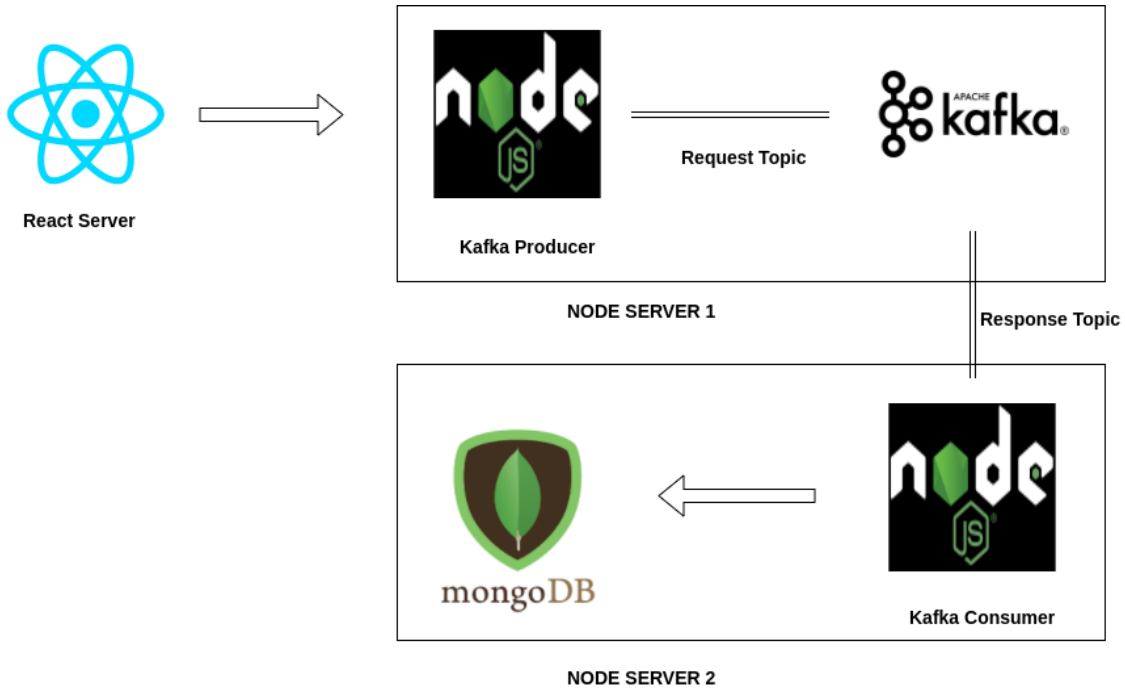
Purpose :

The purpose of this assignment is to understand and implement the concepts of distributed applications including Apache Kafka, MongoDB, connection pooling, encryption using Node Js, React Js with Redux.

The features to be included in the application:

1. Sign up new user
2. Sign in existing user
3. Sign out. Sign Up should have first name, last name, Email and password.
4. Upload a file
5. List a file
6. Create a directory
7. Star a folder/directory.
8. Share a folder/directory by email/name/link.
9. Create, delete, list groups

System Design :



The front-end of the application has been developed using React Js backed with Node Js and MongoDB. For each operation performed by the application, React Js server sends a request to Kafka front-end server (Producer) implemented using NodeJs. The server processes the request and generates a message on the request queue (topic). The Kafka back-end server (Consumer) implemented using Node Js picks up the message from the request queue, processes it and sends a message back on response queue. The response is sent 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 user authentication is performed using Passport Js which is an authentication middleware for Node.js. It is extremely flexible and modular. It abstracts away the complexity of authentication process, which makes the application code more clean and maintainable.

The system also handles session management using Passport session management which is stored in Mongo Store.

The encryption algorithm used for the application is BCRYPT.

APIs created:

1. User Sign Up:

   Method: Post

   URL: localhost:3001/users/signup

2. User Login:

   Method: Post

   URL: localhost:3001/users

3. Get User Details:

   Method: Get

   URL: localhost:3001/users/signup

4. User Update:

   Method: Post

   URL: localhost:3001/users/userupdate

5. File Upload:

   Method: Post

   URL: localhost:3001/files/upload

6. Delete File:

   Method: Post

   URL: localhost:3001/files/delete

7. Make Folder:

   Method: Post

   URL: localhost:3001/files/makefolder

8. Share File:

   Method: Post

   URL: localhost:3001/sharefile

9. Star File:

   Method: Post

   URL: localhost:3001/files/starfile

10. Create Group:

    Method: Post

    URL: localhost:3001/groups/addgroup

11. Get Groups:

    Method: Get

URL: localhost:3001/groups/getgroups

12. Delete Group:

    Method: Post

    URL: localhost:3001/groups/deletegroup

13. Add Member:

    Method: Post

    URL: localhost:3001/groups/addmember

14. Get Members:

    Method: Get

    URL: localhost:3001/groups/getmembers

15. Delete Member:

    Method: Post

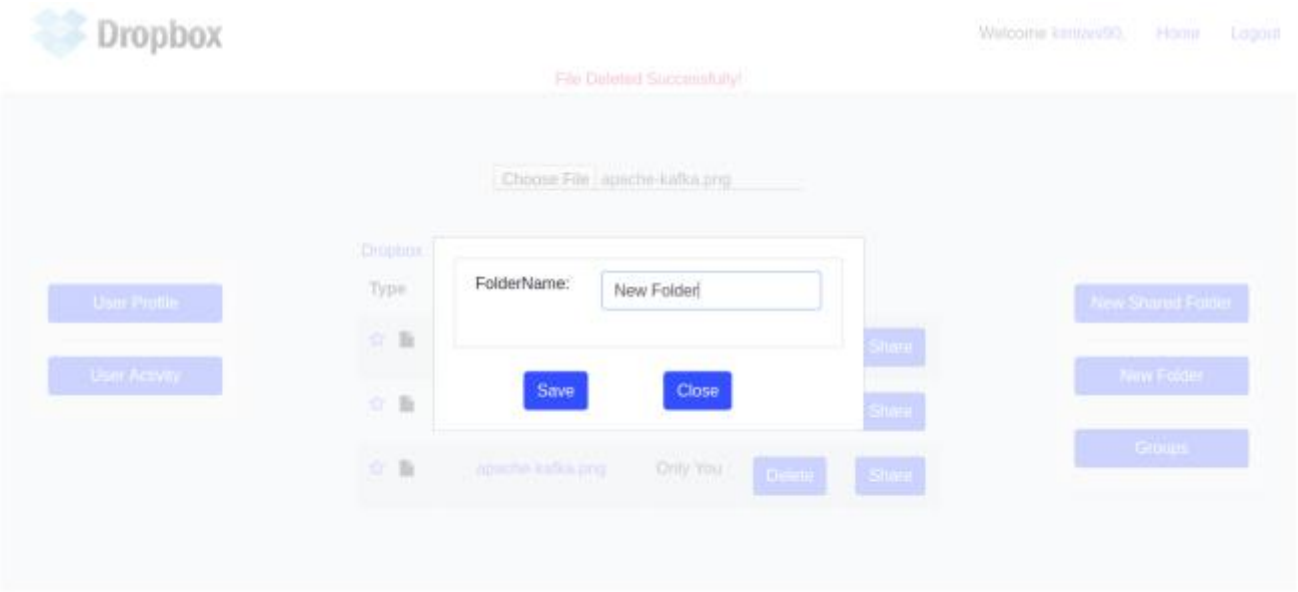
    URL:localhost:3001/groups/deletemember

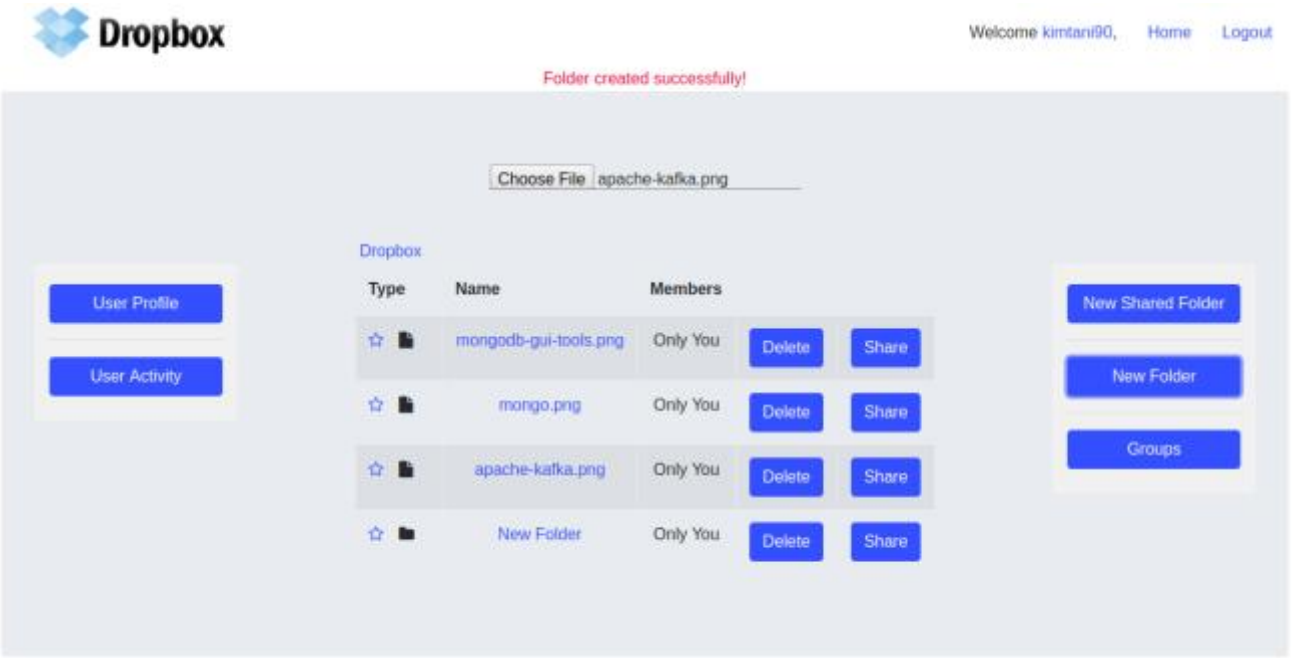
Screenshots :

User SignUp Page

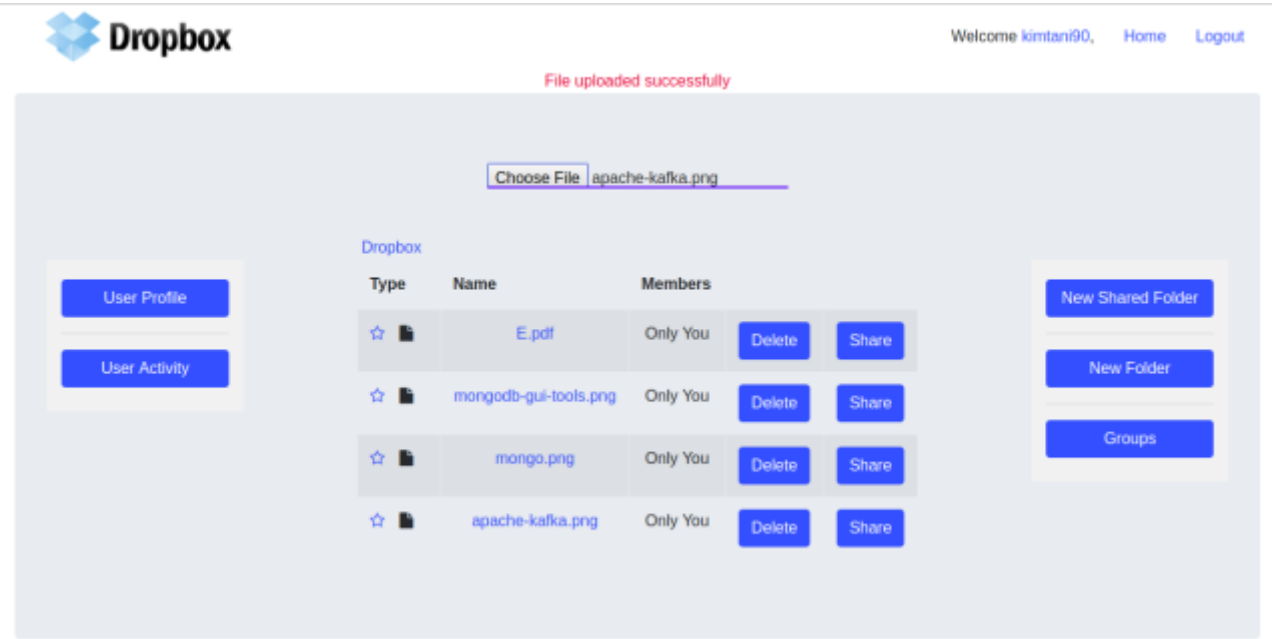
User Login Page

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

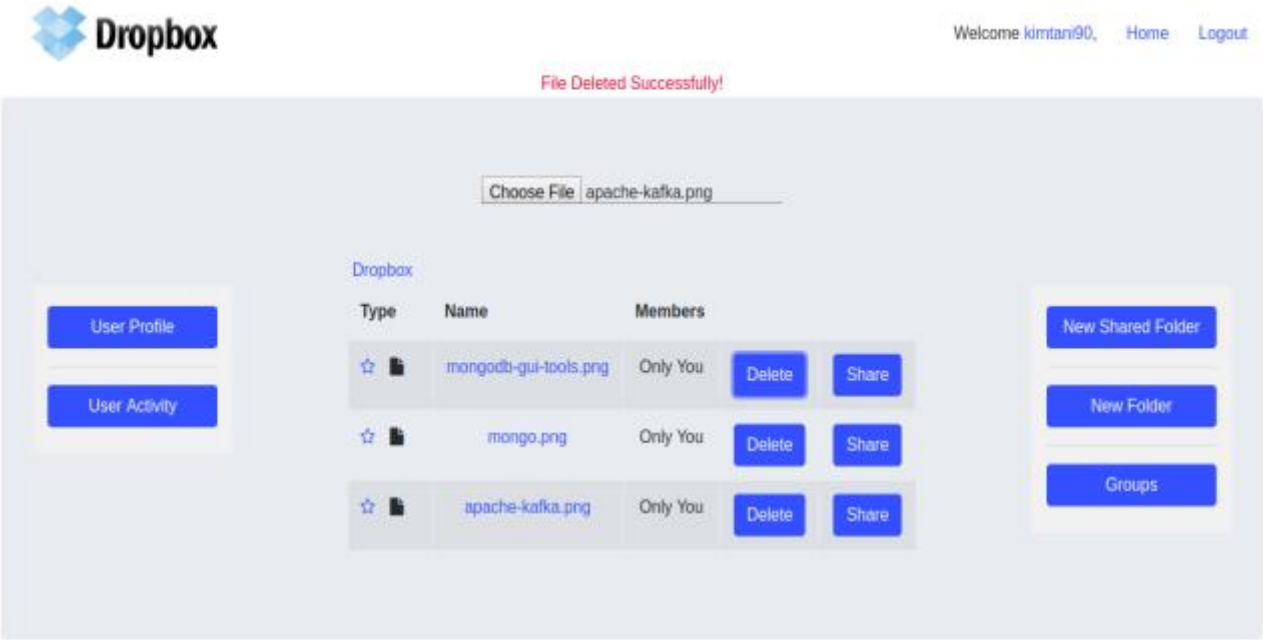




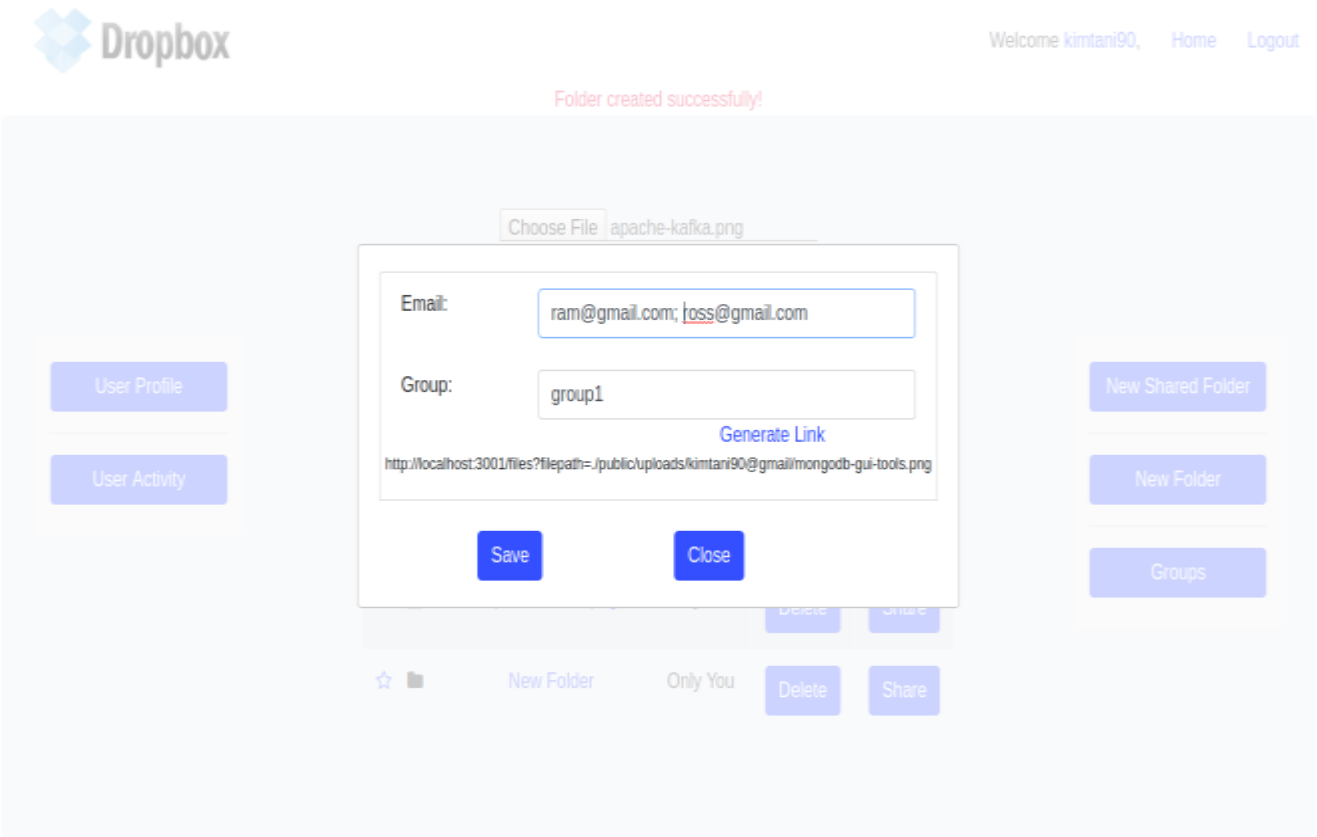
Click on choose file and upload it. The file gets listed in the file list.



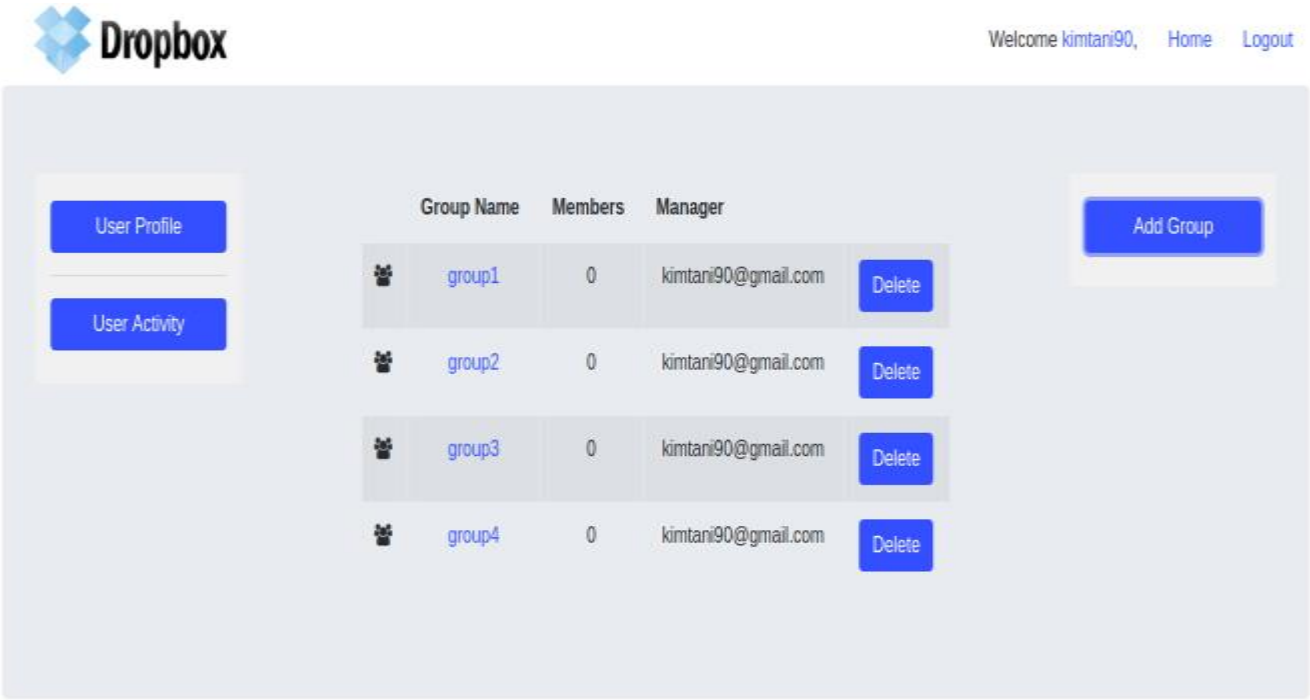
Click on delete button corresponding to a file or folder and the file gets deleted.



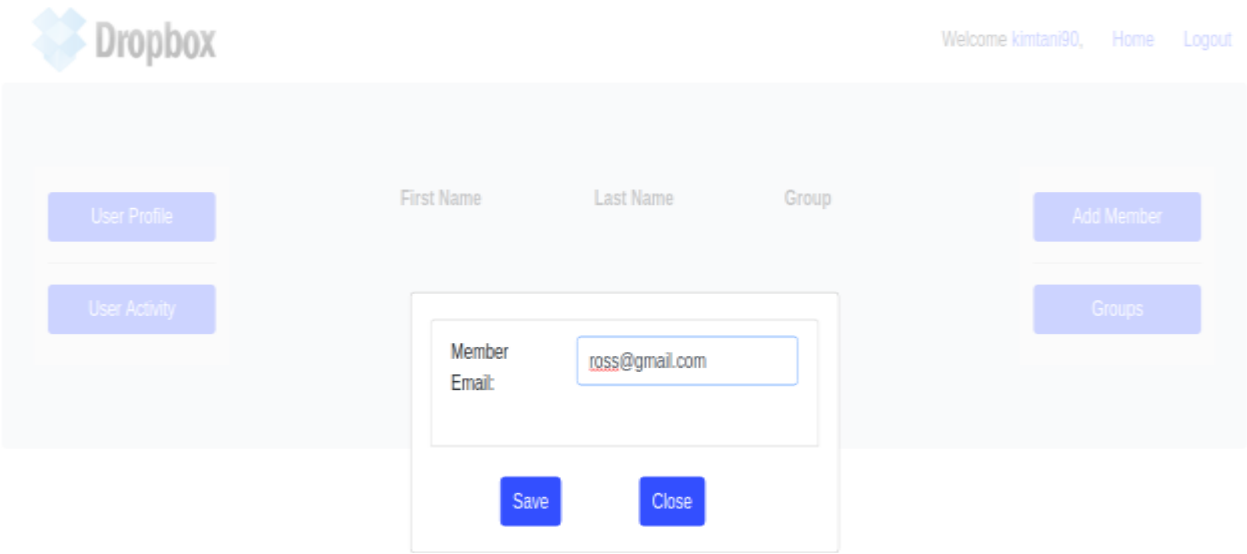
Click on share button corresponding to a file, a pop-up opens. Enter a list of emails separated by semi-colon or input a group name with which user wants to share the file. User can also share the file using generate link button.

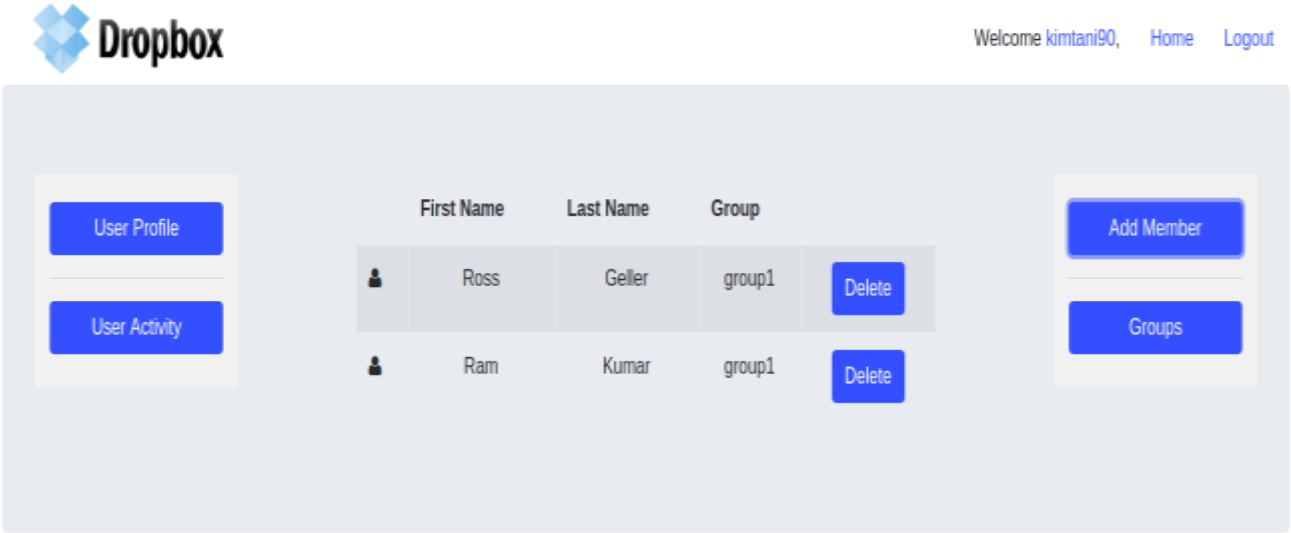


Click on Add Group button to add a group.

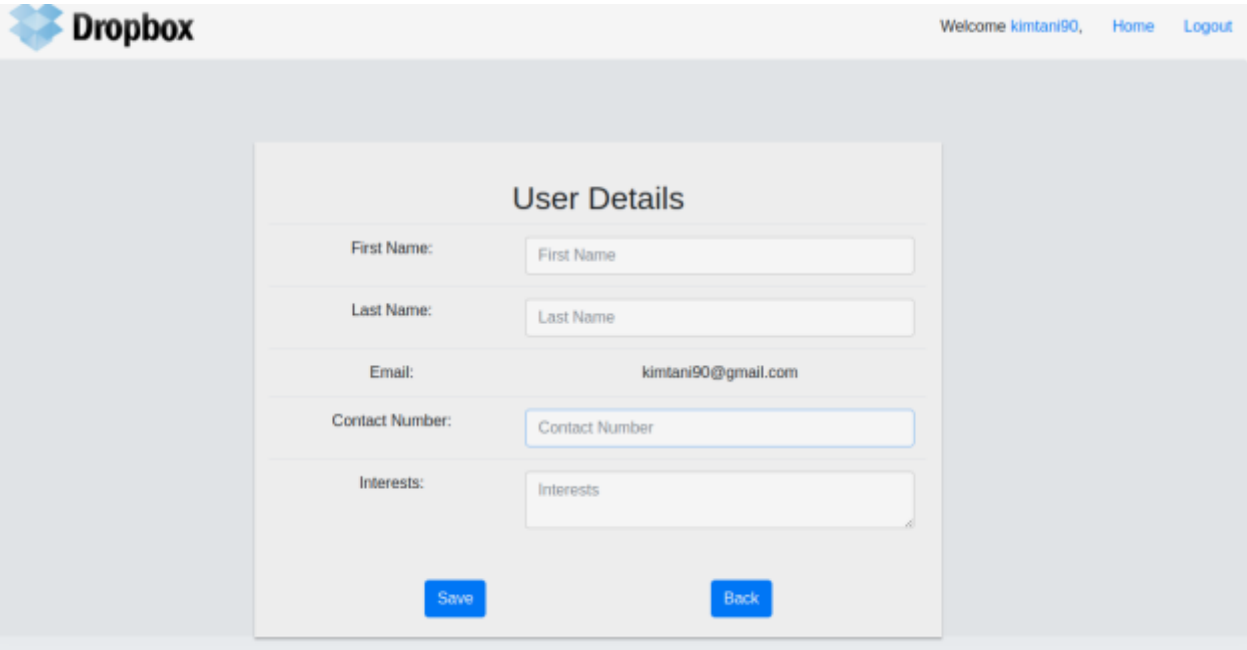


Click on Add Member button to add a member to a group. A pop up appears to enter the email of the member.

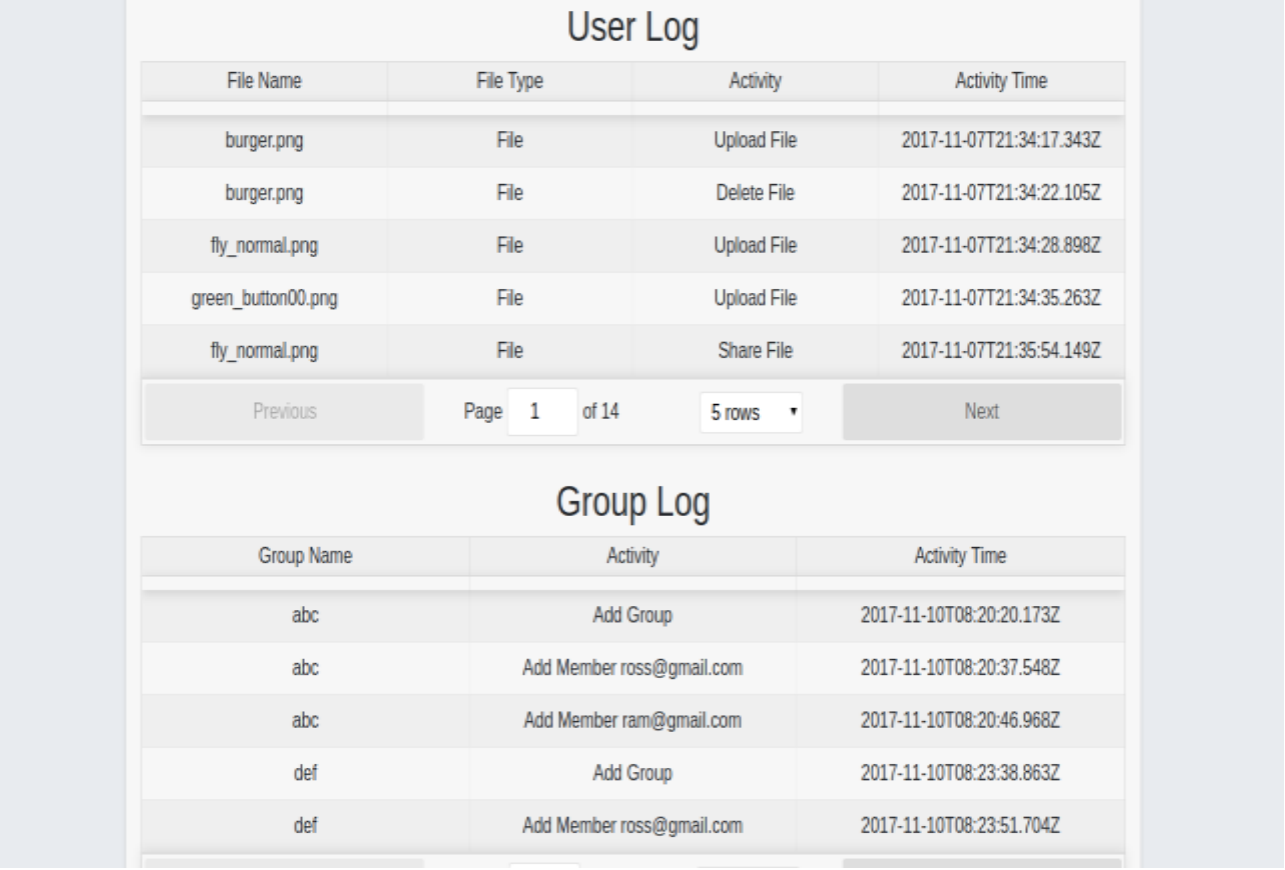

Dropbox
Welcome kimtani90,
Home
Logout
User Profile
User Activity
First Name
Last Name
Group
Add Member
Groups
Member Email:
ross@gmail.com
Save
Close


Dropbox
Welcome kimtani90,
Home
Logout
User Profile
User Activity
First Name
Last Name
Group
Ross
Geller
group1
Delete
Ram
Kumar
group1
Delete
Add Member
Groups

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



Click on User Activity button to navigate to User Log page



User Log

File Name	File Type	Activity	Activity Time
burger.png	File	Upload File	2017-11-07T21:34:17.343Z
burger.png	File	Delete File	2017-11-07T21:34:22.105Z
fly_normal.png	File	Upload File	2017-11-07T21:34:28.898Z
green_button00.png	File	Upload File	2017-11-07T21:34:35.263Z
fly_normal.png	File	Share File	2017-11-07T21:35:54.149Z

Previous Page 1 of 14 5 rows Next

Group Log

Group Name	Activity	Activity Time
abc	Add Group	2017-11-10T08:20:20.173Z
abc	Add Member ross@gmail.com	2017-11-10T08:20:37.548Z
abc	Add Member ram@gmail.com	2017-11-10T08:20:46.968Z
def	Add Group	2017-11-10T08:23:38.863Z
def	Add Member ross@gmail.com	2017-11-10T08:23:51.704Z

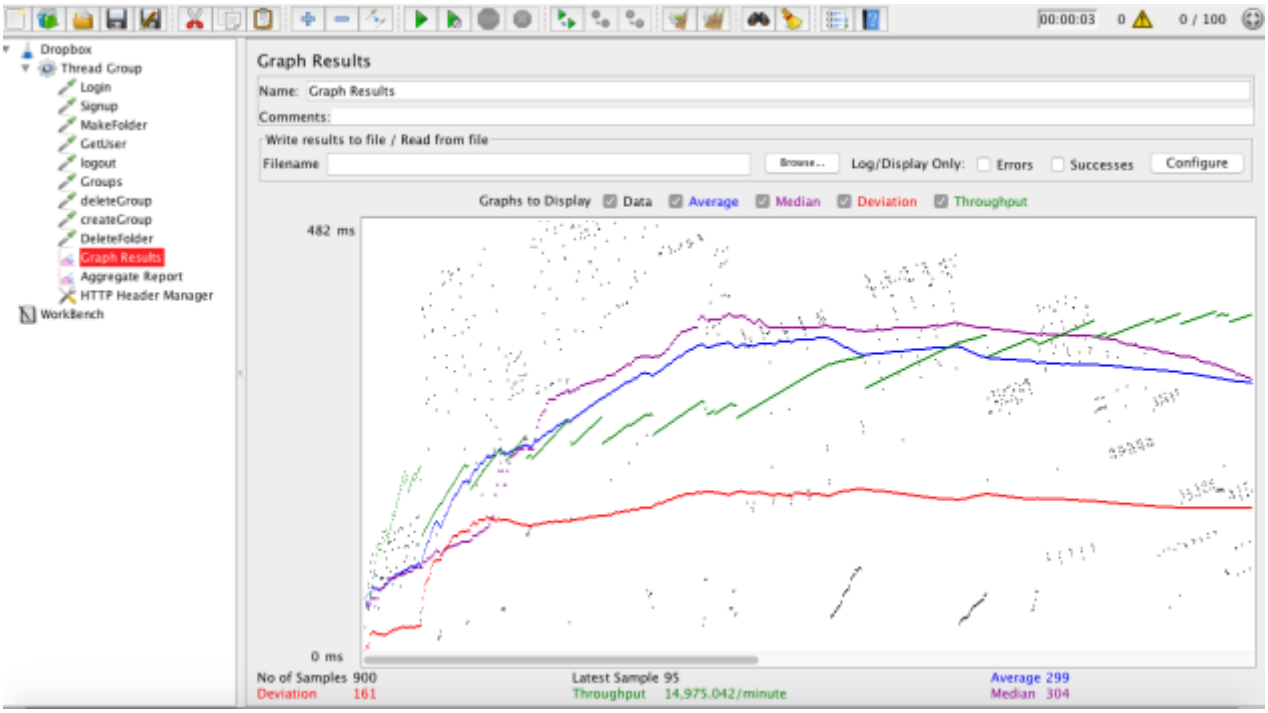
Performance :

JMeter Testing:

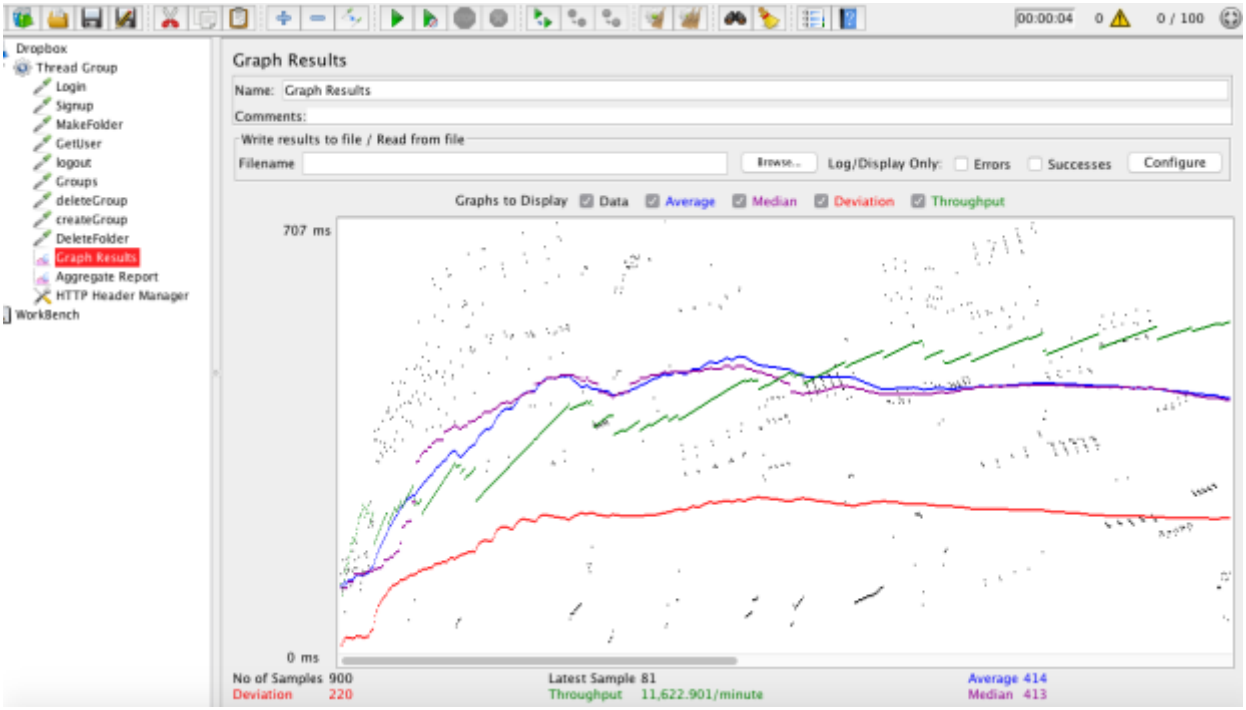
The Performance of the system was good considering the amount of concurrent requests it could handle, mongodb connection pooling played a significant role in decreasing the average time of serving the request .

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

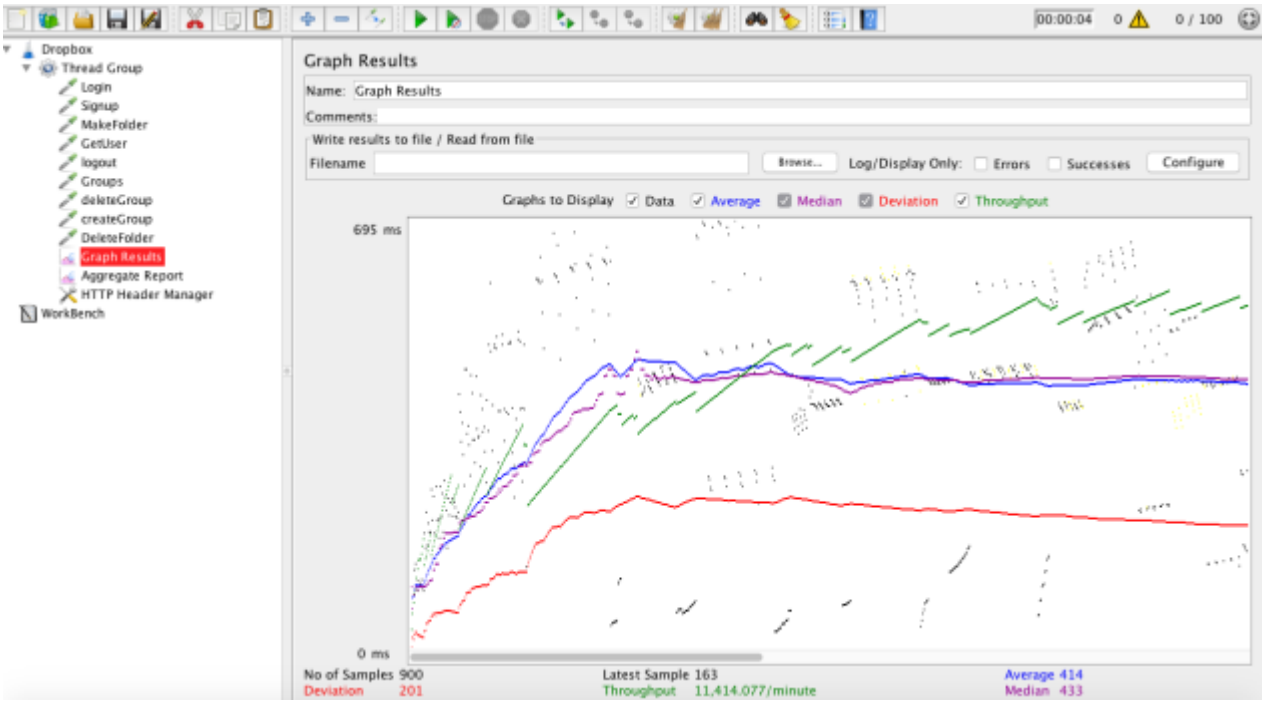
1. For 100 concurrent with self implemented connection pooling.



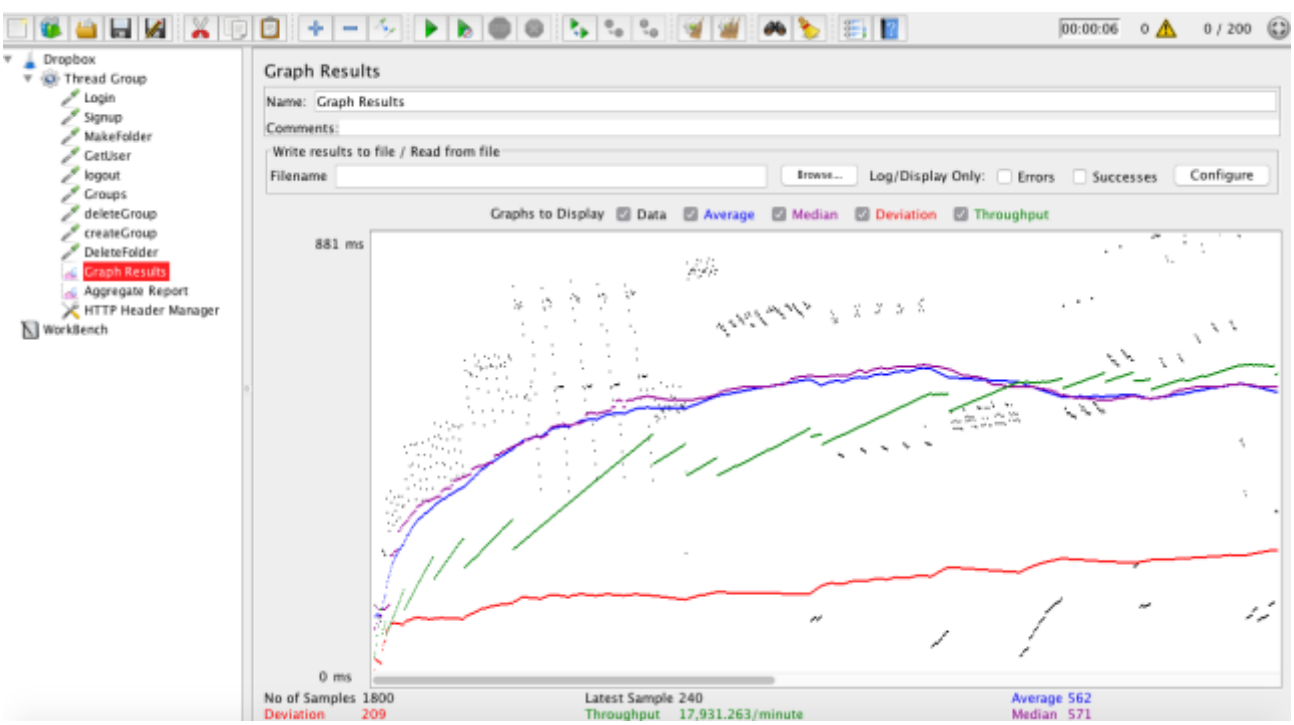
For 100 concurrent users with DB implemented connection pooling.



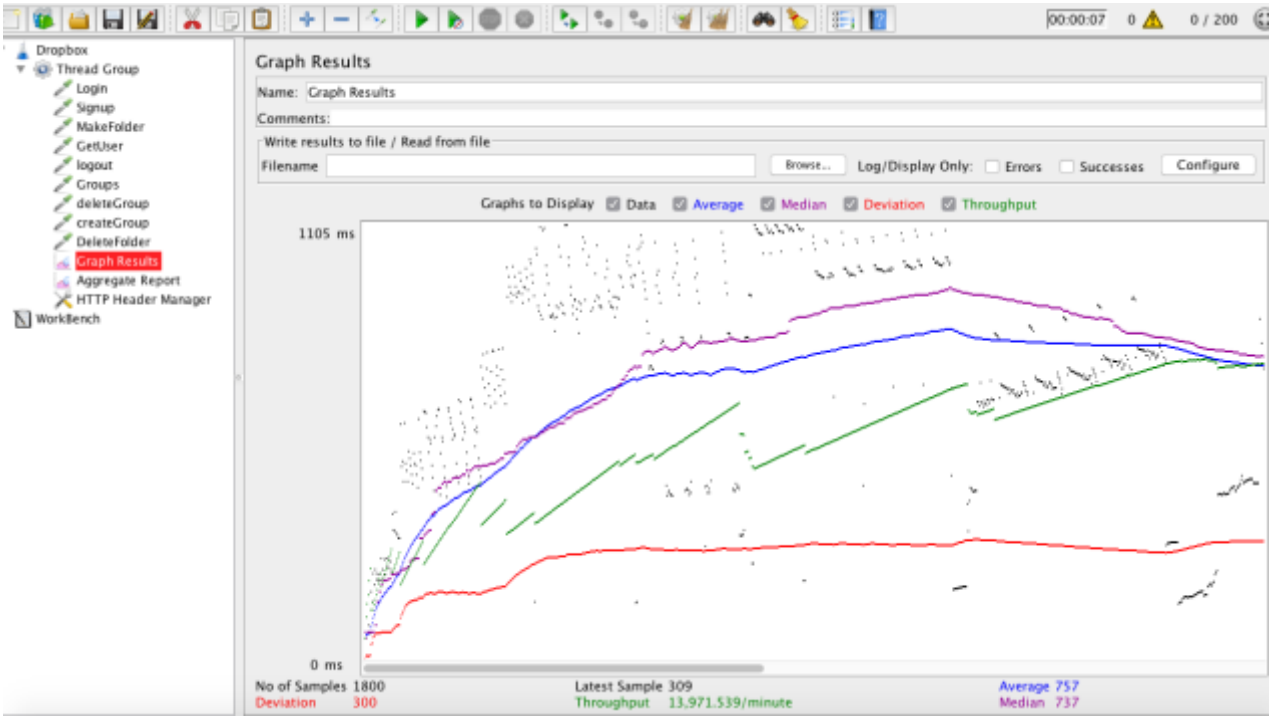
For 100 concurrent users with DB implemented connection pooling.



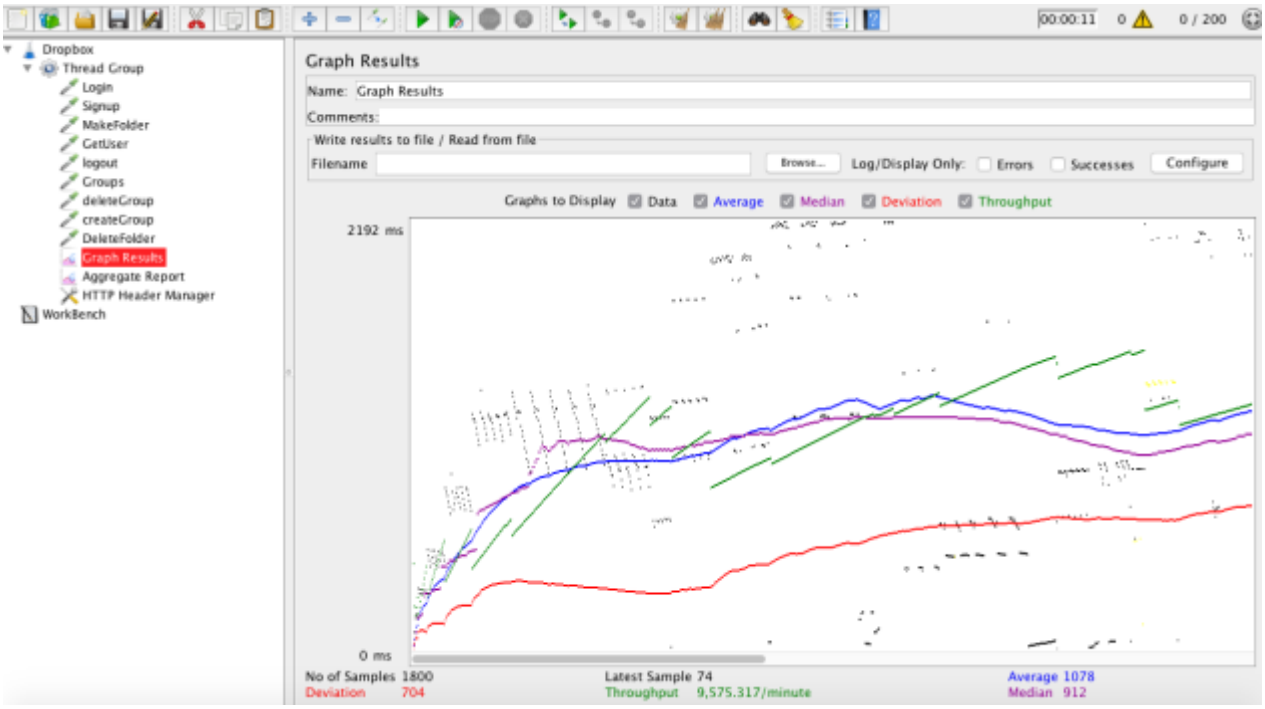
2. For 200 concurrent users with self implemented connection pooling.



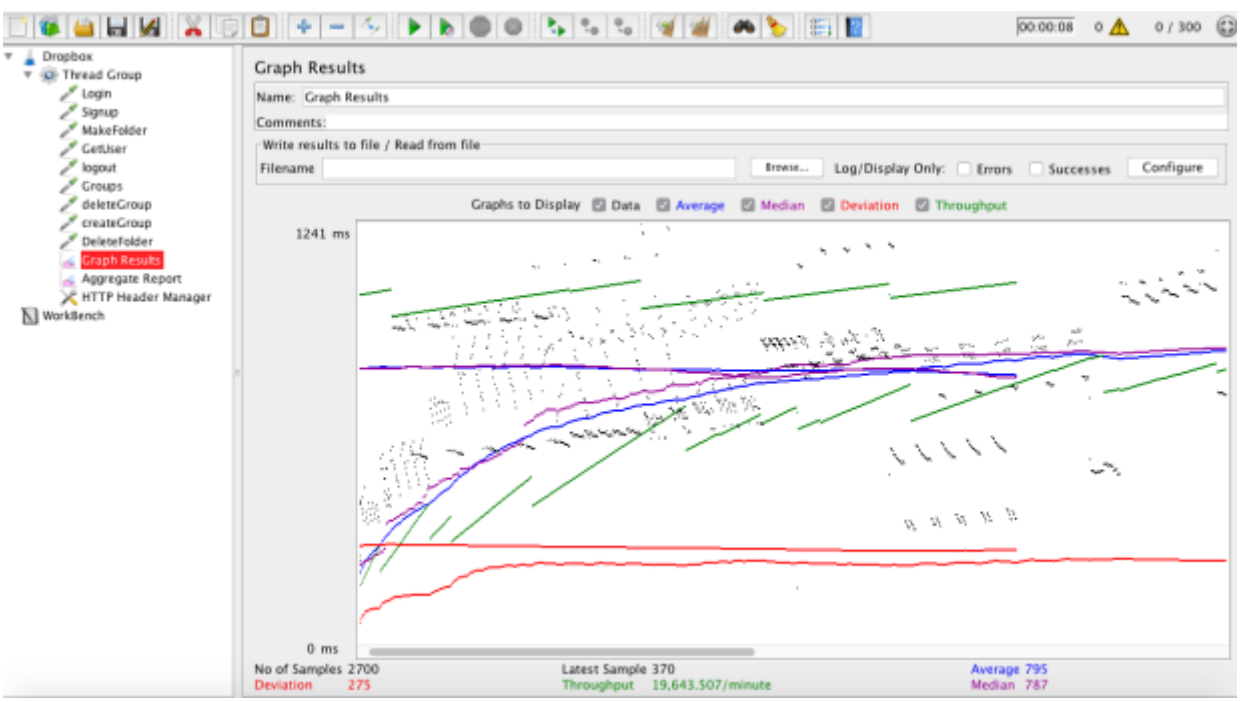
For 200 concurrent users with DB implemented connection pooling.



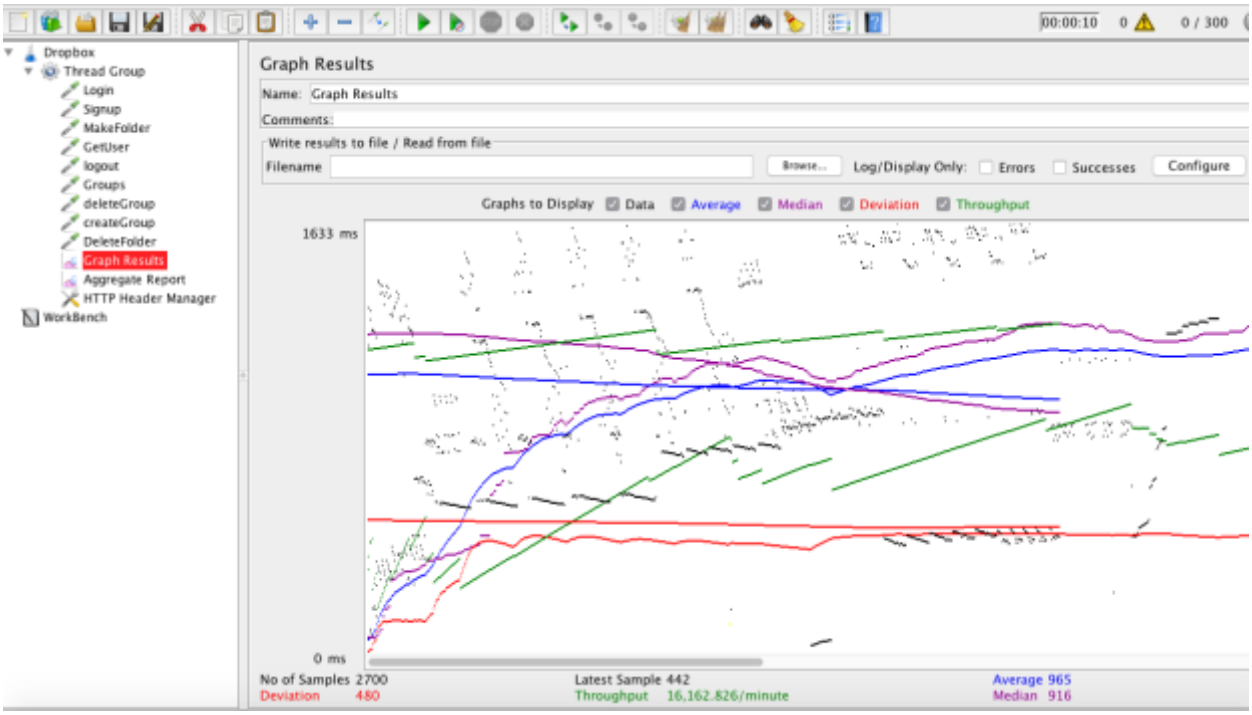
For 200 concurrent users without connection pooling.



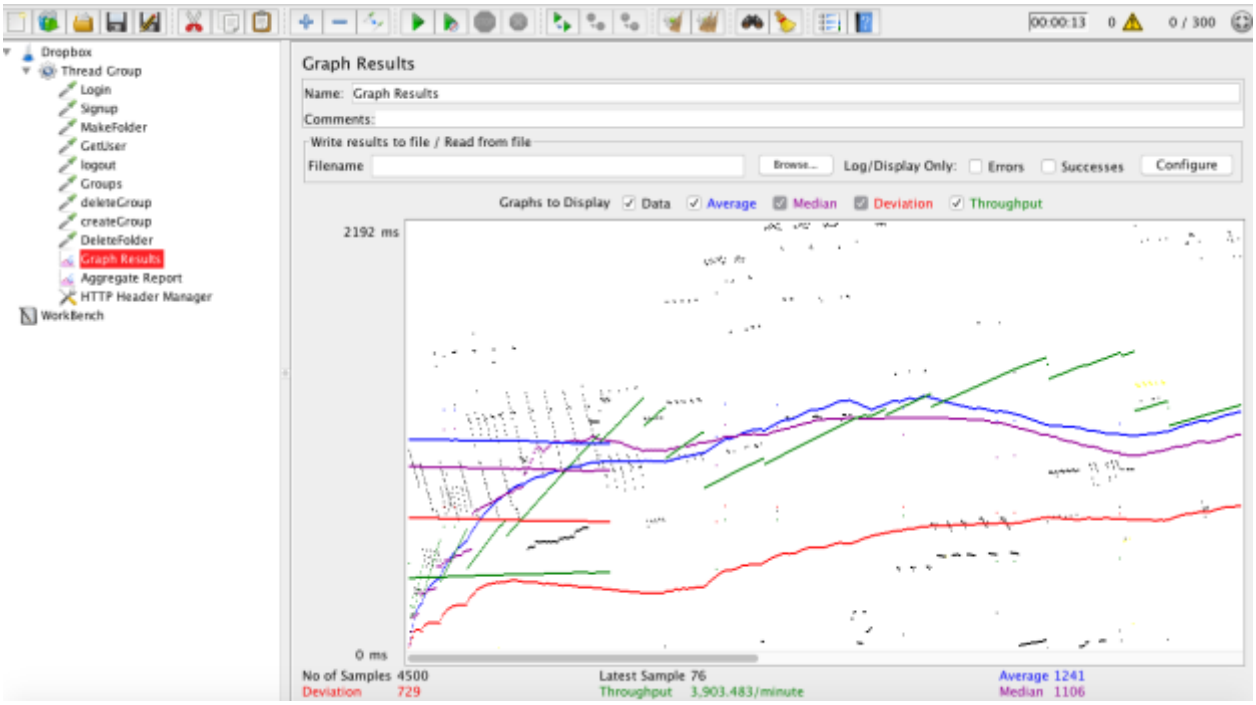
3. For 300 concurrent users with self implemented connection pooling.



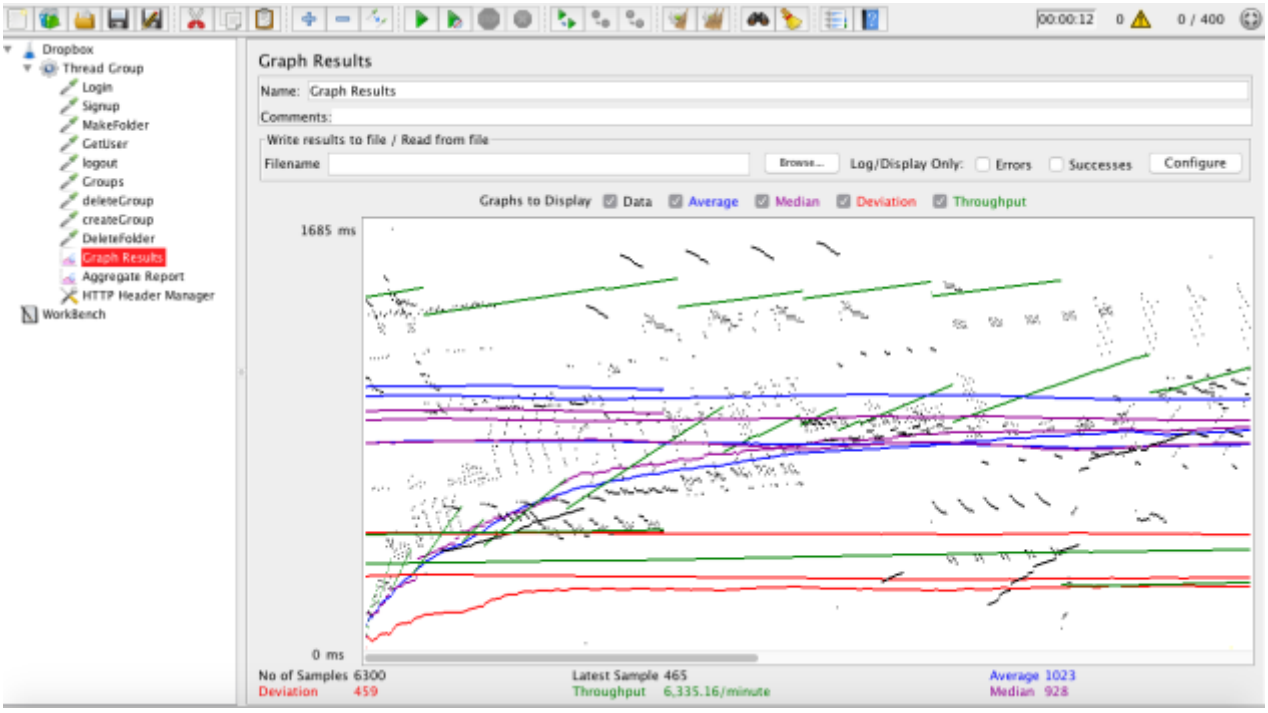
For 300 concurrent users with DB implemented connection pooling.



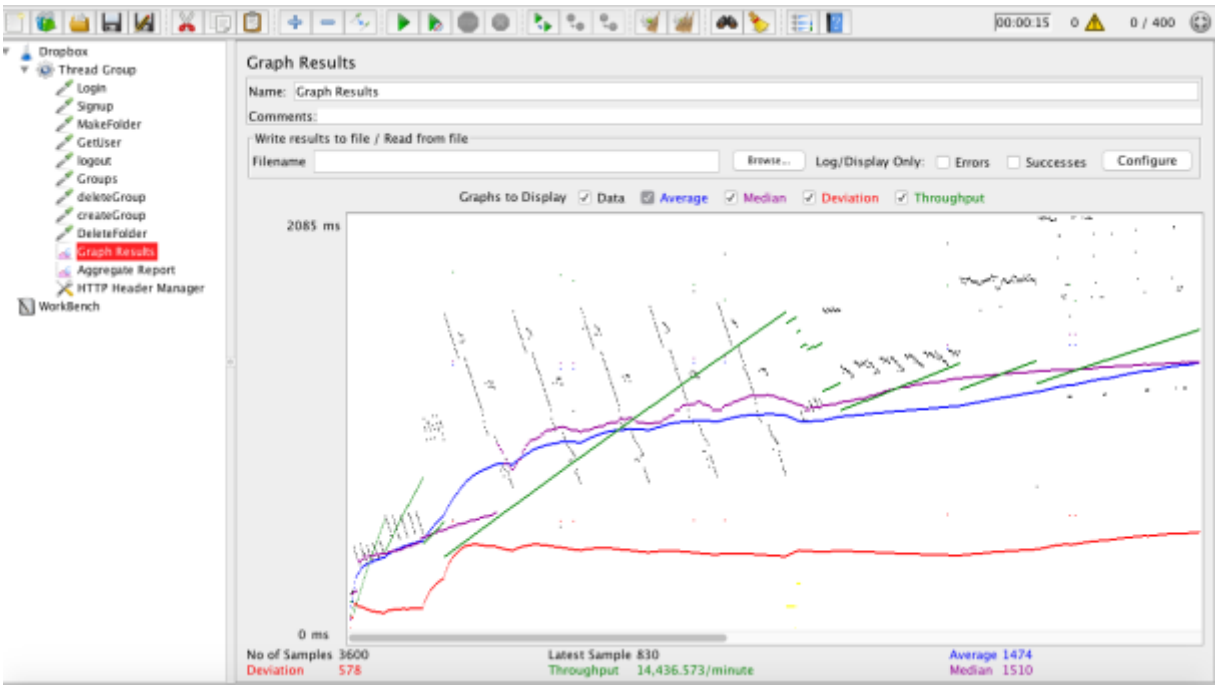
For 300 concurrent users without connection pooling.



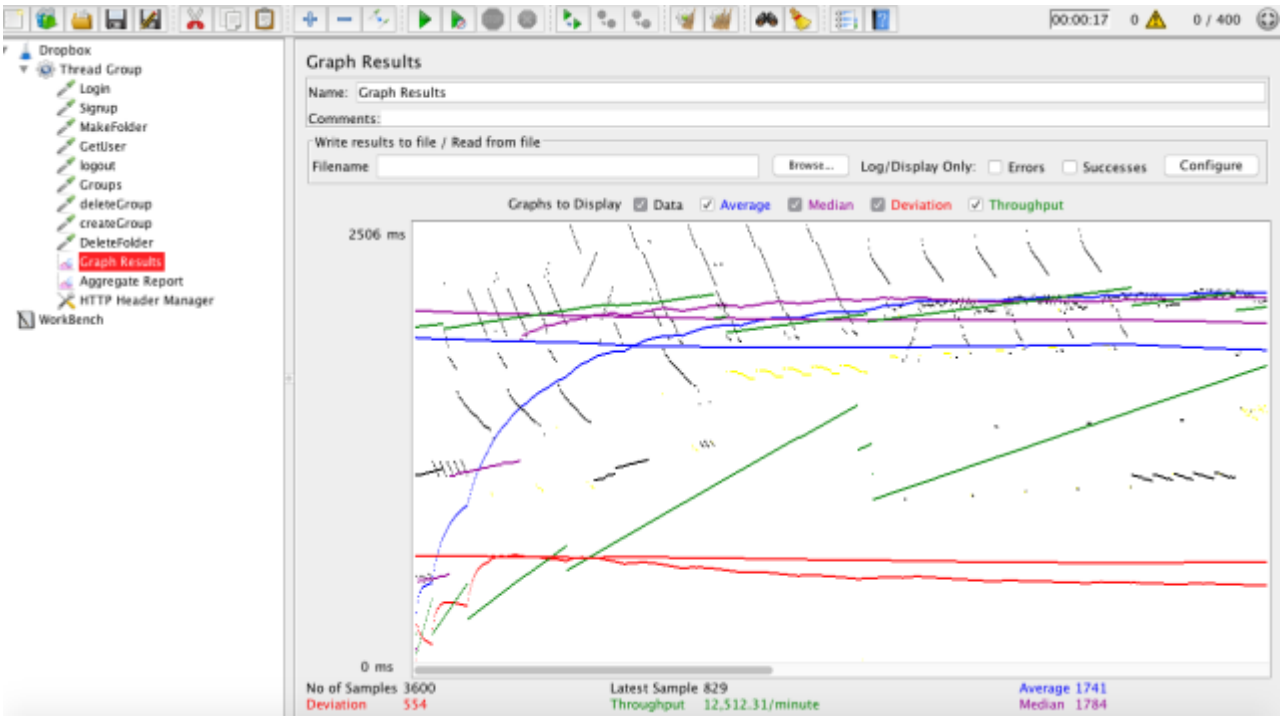
4. For 400 concurrent users with self implemented connection pooling.



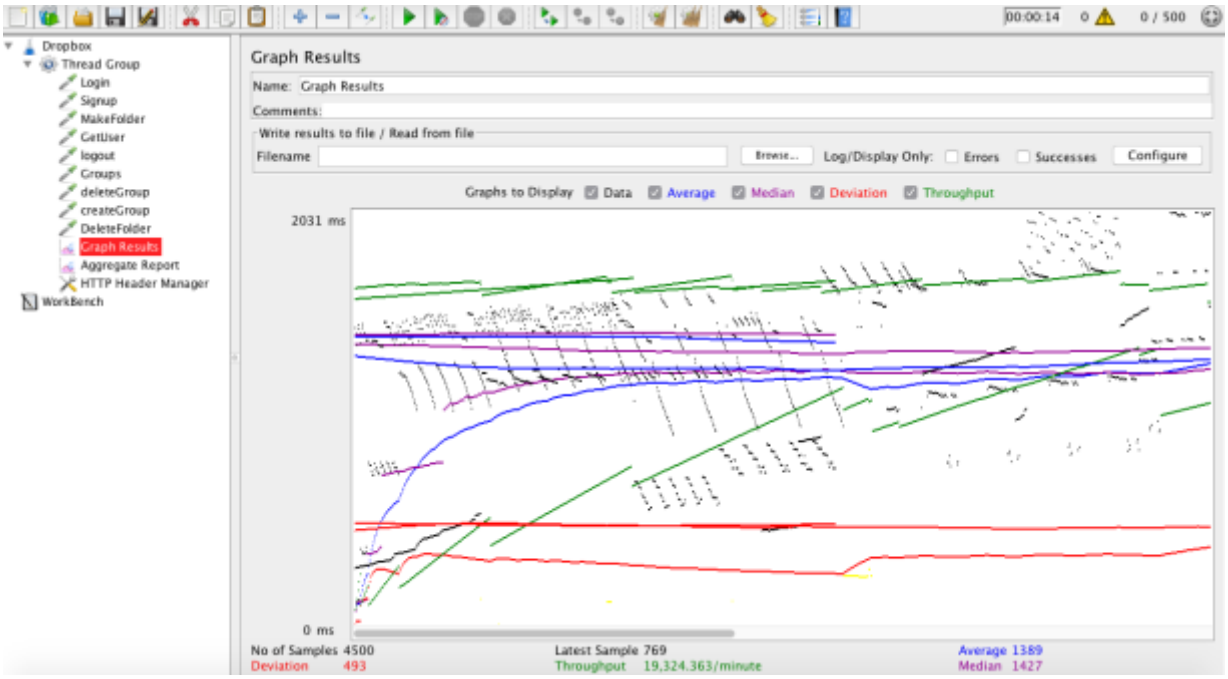
For 400 concurrent users with DB implemented connection pooling.



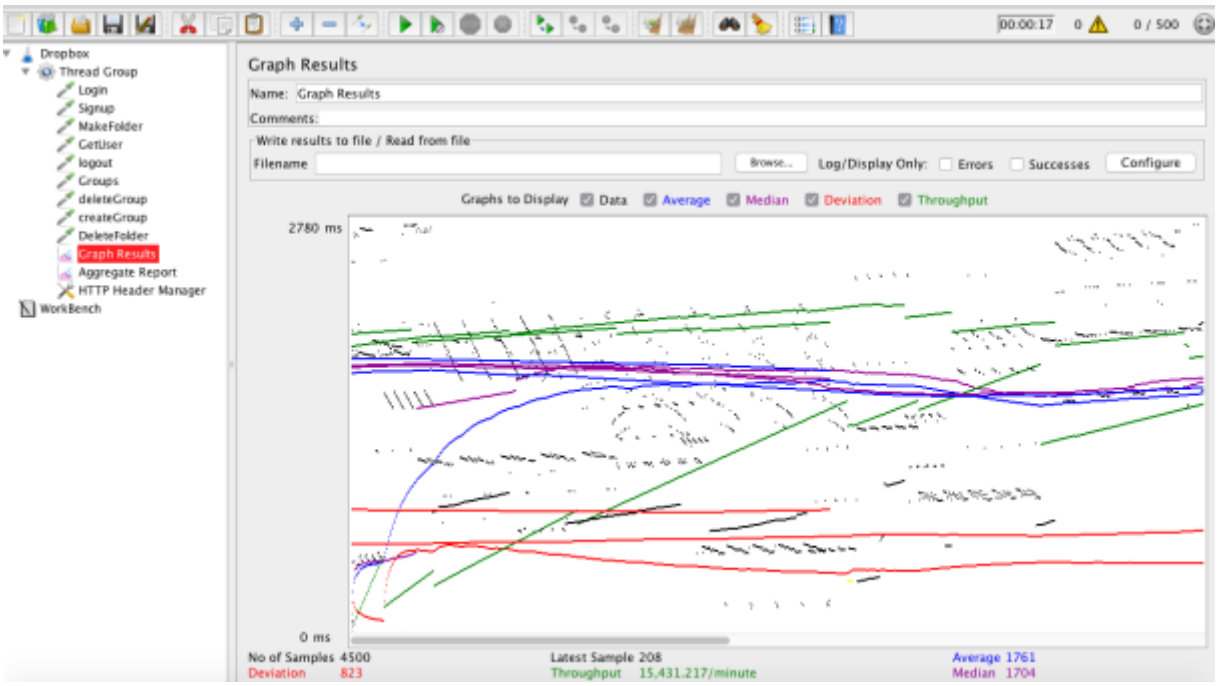
For 400 concurrent users without connection pooling.



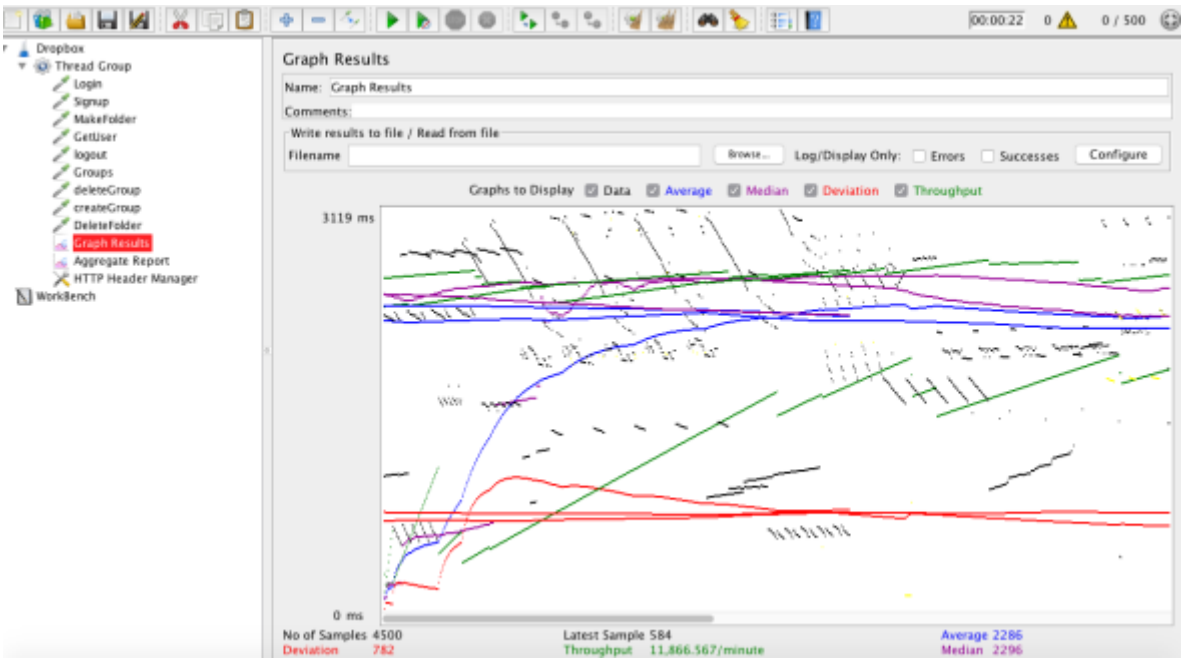
5. For 500 concurrent users with self implemented connection pooling.



For 500 concurrent users with DB implemented connection pooling.



For 500 concurrent users without connection pooling.



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

Connection Pooling improves the performance of the system and increases the through put of the application, it takes less time to process the request using the connection pooling because a majority of time is spent in establishing the connection but with connection pooling a pool of connection is maintained which can be used by application for processing the requests.

The average time on all the three cases for 100 concurrent users is not much different but as the number of concurrent users increases the connection pooling with higher number of open connections increases the performance.

Mocha Js Testing Screenshot:

```
dishant@dishant-Inspiron-3537:~/Lab2/CMPE273-Lab2/dropbox/kafka-front-end$ npm test

> untitled@0.0.0 test /home/dishant/Lab2/CMPE273-Lab2/dropbox/kafka-front-end
> mocha



  Login
    ✓ should login (116ms)

  Sign Up
    ✓ should signup (113ms)

  Get User Details
    ✓ should get user details

  Delete File
    ✓ should delete file

  Share File
    ✓ should share a file

  Create Group
    ✓ should create a group

  Get Groups
    ✓ should get a list of groups

  Get Members
    ✓ should get a list of members

  Delete Group
    ✓ should delete a group

  Delete Member
    ✓ should delete a member


  10 passing (441ms)
```

Questions:

1. Compare passport authentication process with the authentication process used in Lab1.

In lab 1 we used basic authentication mechanism to authenticate the user where we manually check the username and password to authenticate the user. This is a less efficient process when compared to passport Js in-build authentication mechanism

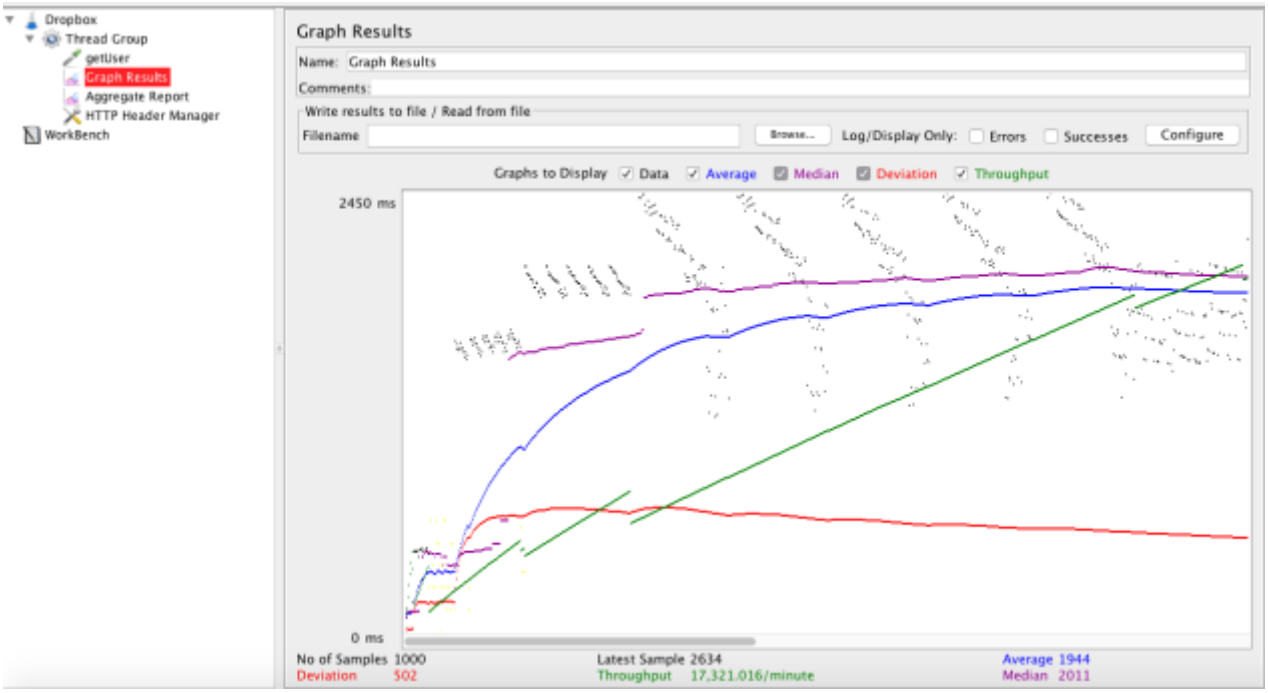
In Lab 2 we have used local strategy in Passport Js for user authentication. Passport is authentication middleware for Node.js. It is extremely flexible and modular. It abstracts away the complexity of authentication process, which makes the application code more clean and maintainable.

The local authentication strategy authenticates users using a username and password. The strategy requires a verify callback, which accepts these credentials and calls done providing a user.

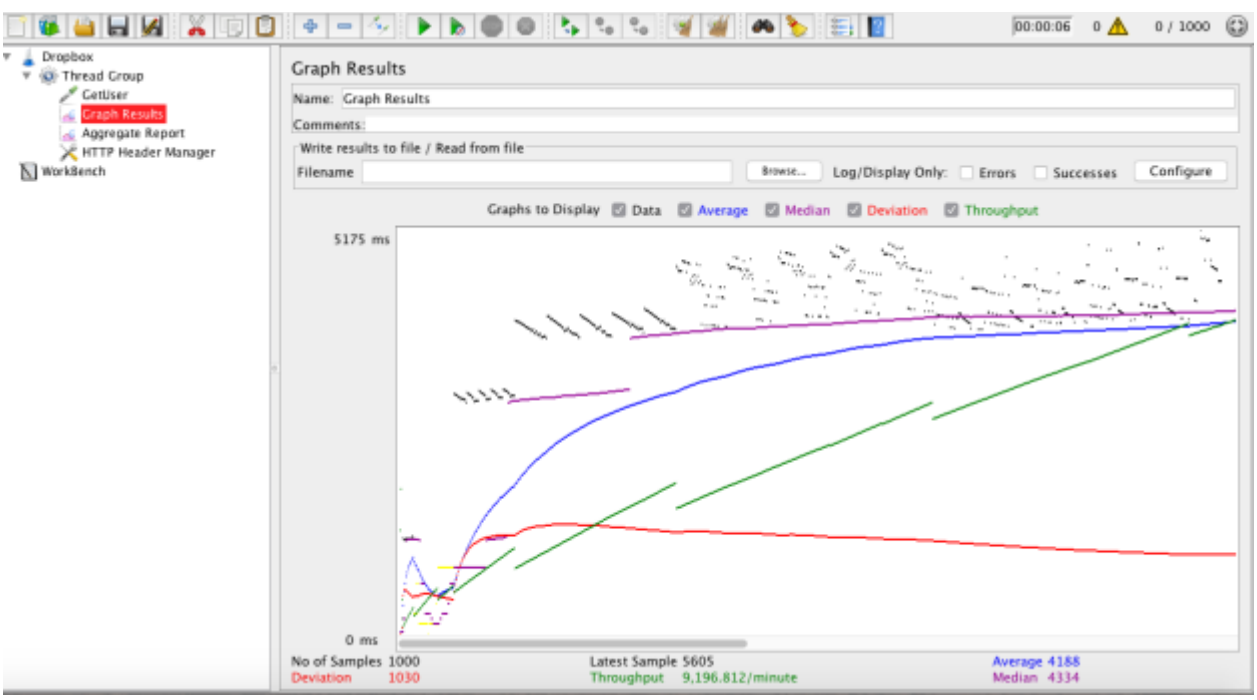
In addition, passport supports plugins for authentication using google, twitter, facebook etc.

2. Compare performance with and without Kafka. Explain in detail the reason for difference in performance.

JMeter testing for 1000 concurrent users for API "GetUsers" with Kafka.



JMeter testing for 1000 concurrent users for API "GetUsers" without Kafka.



The average time when tested without kafka is 4188 ms while when tested without kafka is 1944 ms. The performance is about 2 times better when kafka is used.

As the number of concurrent requests increases, the difference becomes more visible since whenever an API call is performed the majority of time is spent in serving the request and is wasted while fetching the data from the database, using kafka we reduce that time. The node server's (kafka frontend) job is to process the request and send only the revelant data to the backend server so the backend server does not waste any time unpacking the request and can directly work on getting the response from the database. Another major advantage of using kafka is that the application can become distributed, we can replicate the backend code into more number of servers and can make them consumers which will increase the performance of the system.

3. If given an option to implement MySQL and MongoDB both in your application, specify which data of the applications will you store in MongoDB and MySQL respectively.

MySQL stores data in tables and uses structured query language (SQL) for database access, while MongoDB stores data in JSON-like documents that can vary in structure.

If given an option, we will store user data and user group details in MySQL since MySQL offers high performance in data access and user schema will not require frequent changes.

MongoDb will be used to store file data due to its flexible data model which allows us to change database schema with change in business requirements. User logs which include user activity data for user group and file changes can be stored in MongoDB since it will be a large amount of data and not frequently accessed.

MongoDB can also be scaled within and across multiple distributed data centers, providing new levels of availability and scalability.