**Sunny Udhani : 012457289**

**Repository: https://github.com/sunny-udhani/online-file-sharing**

# PART - 1 *CALCULATOR*

## **Introduction:**

*Goals and purpose of your system:*

To develop a system demonstrating stateless web service.

The application is capable of handling different operations such as

1.Addition

2.Subtraction

3. Multiplication

4. Division

Also, there are error checks in place to validate input.

## **System Design:**

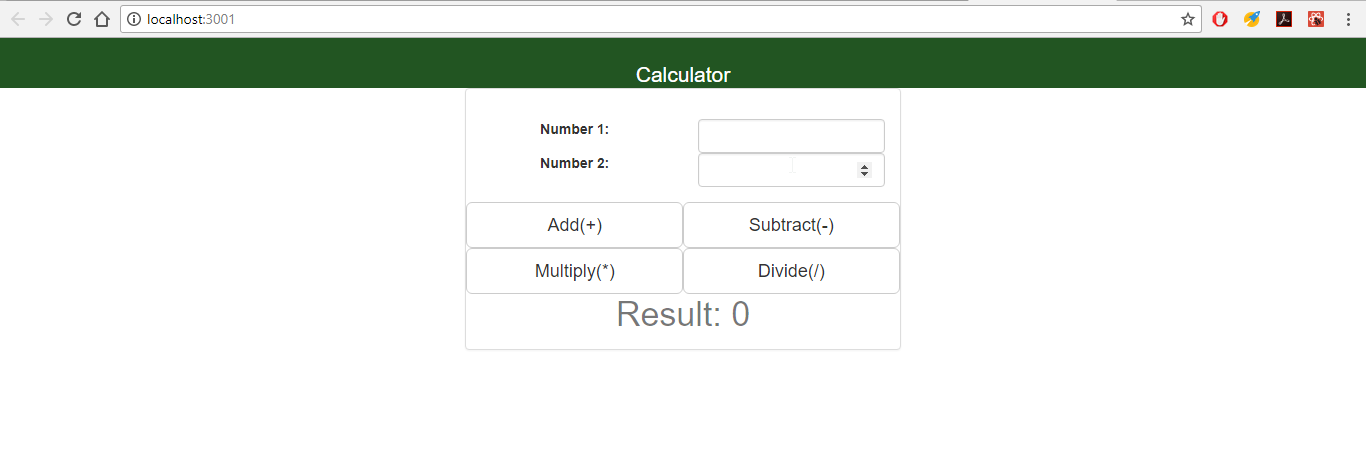
Application stack: NodeJS, ExpressJS, ReactJS, Bootstrap.

* Application has a simple design with a single page and different operations.
* API endpoints: 1. /add 2. /sub 3. /mul 4. /div
* Exceptions taken care of:
  + On dividing number by 0, it will display “Cannot divide by 0” message.
  + If a user tries divides 0 by 0, it will display “Undefined” as result.

## **Results**:

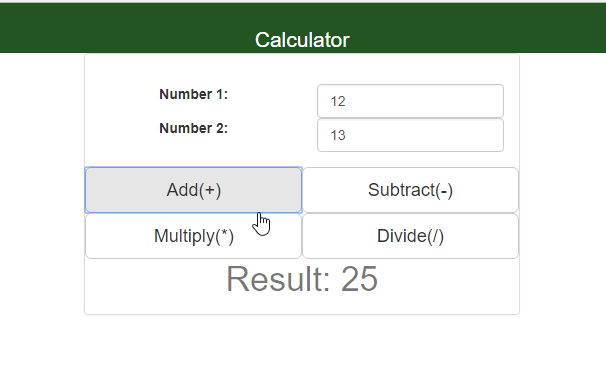
*Screen captures of Calculator:*

**Screen 1:**

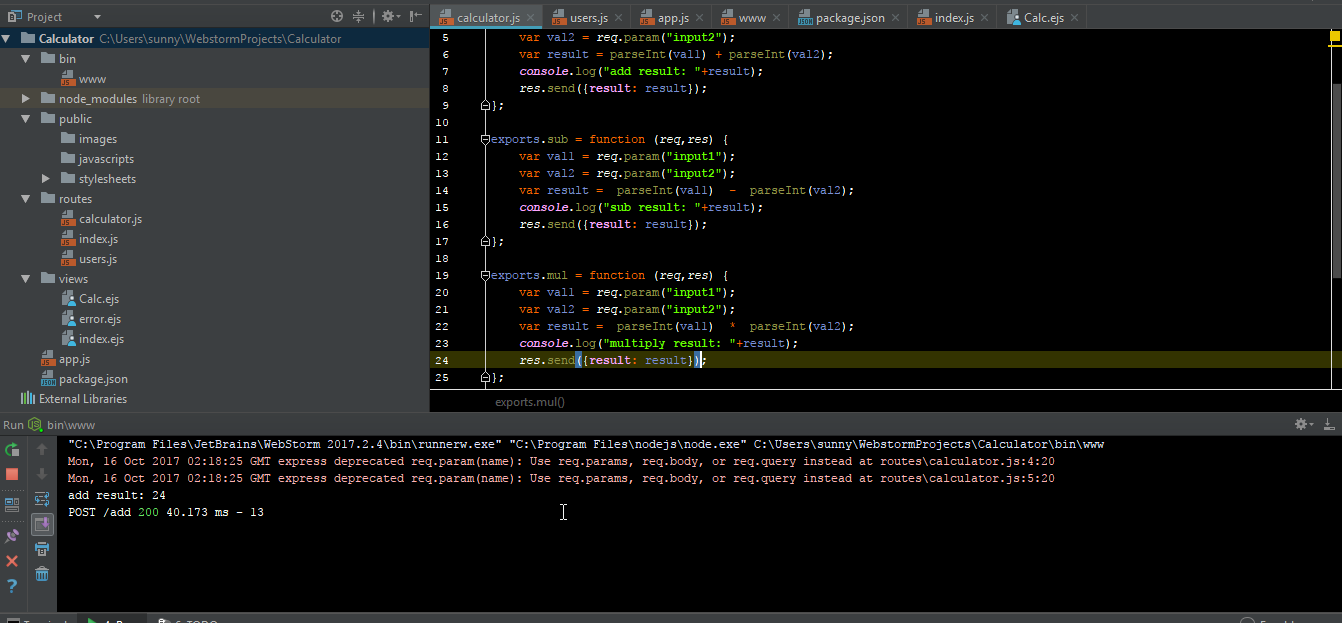


**Screen 2: Addition operation when user clicks on “Add” button.**

Client Side

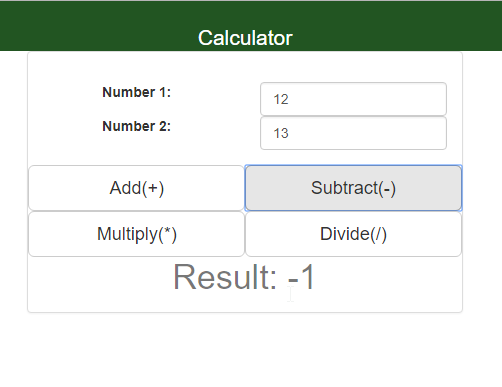


Server Side

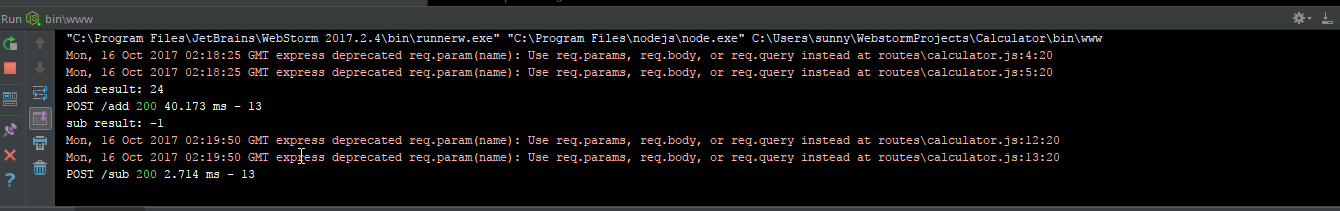


**Screen 3: Subtraction operation when user clicks on “Sub” button.**

**Client Side**

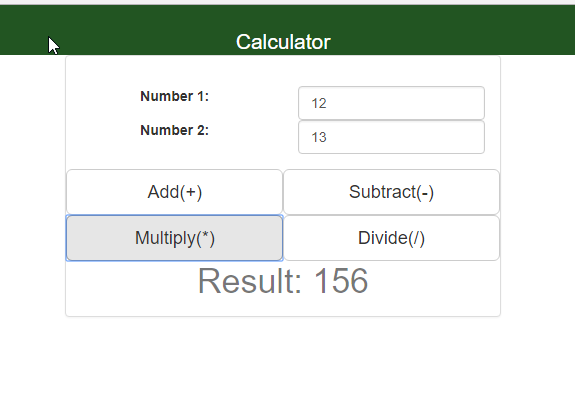


Server Side

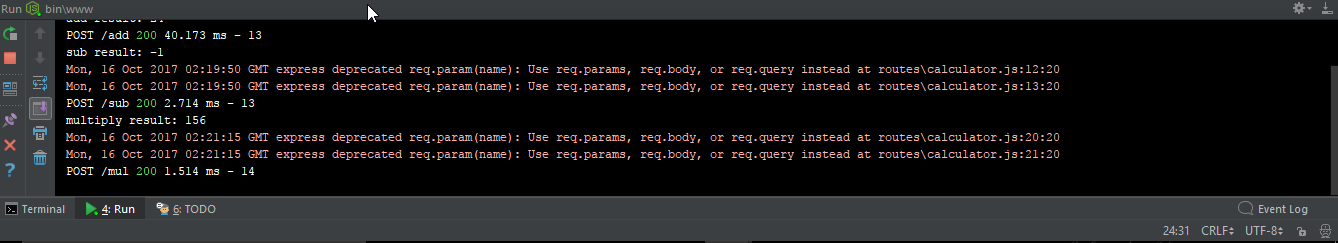


**Screen 4: Multiplication operation when user clicks on “Multiply” button.**

Client Side

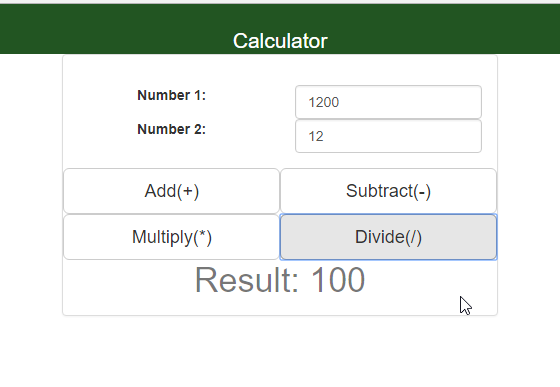


Server side

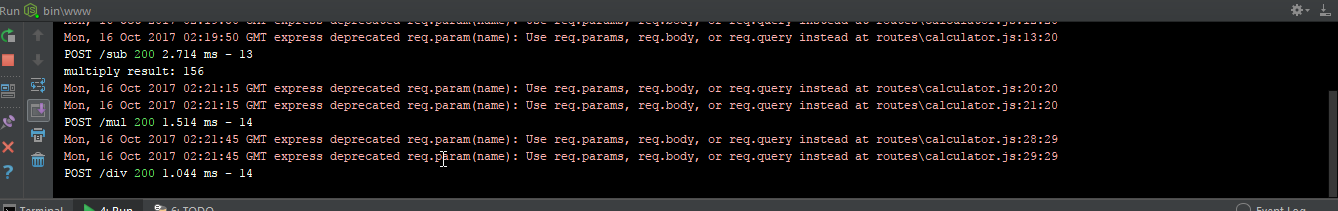


**Screen 5: Division operation when user clicks on “Divide” button.**

Client side

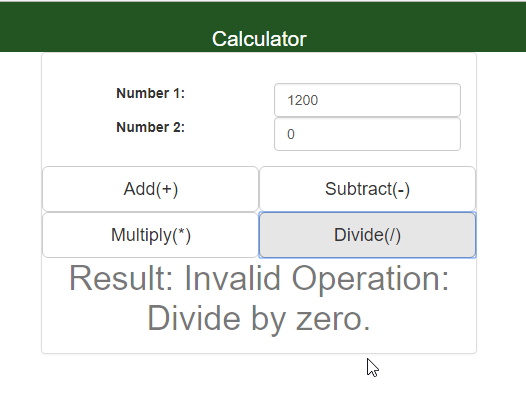


Server side

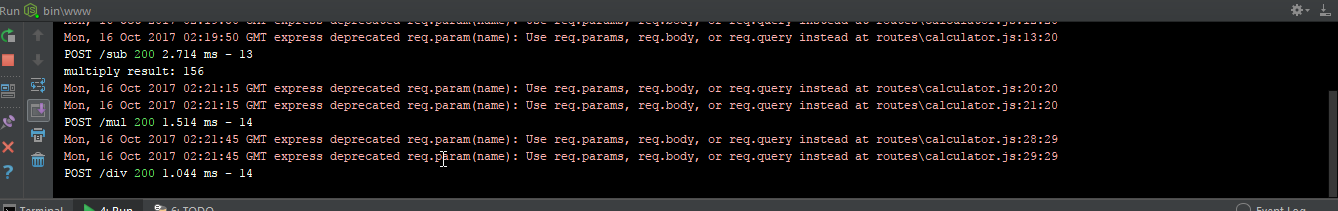


**Screen 6: When user tries to divide a number by 0.**

Client side:



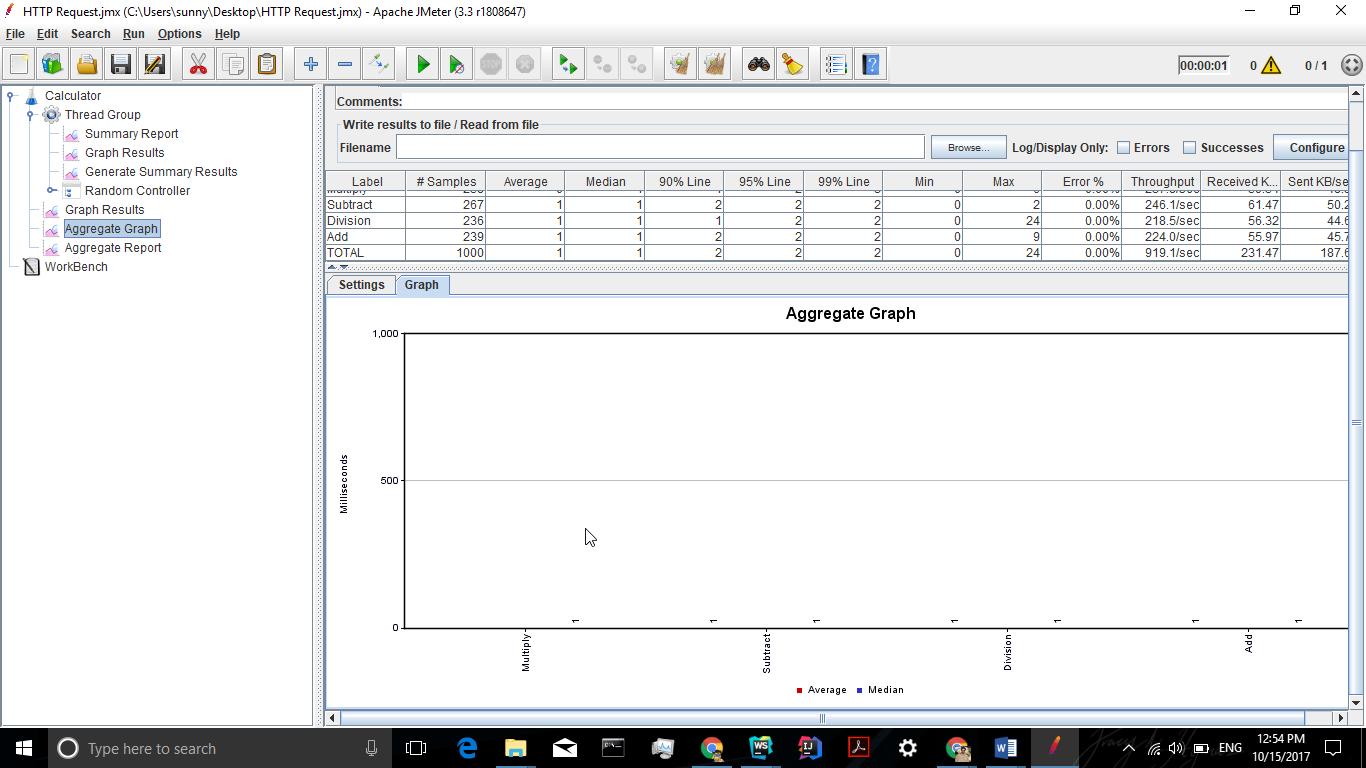
Server side:

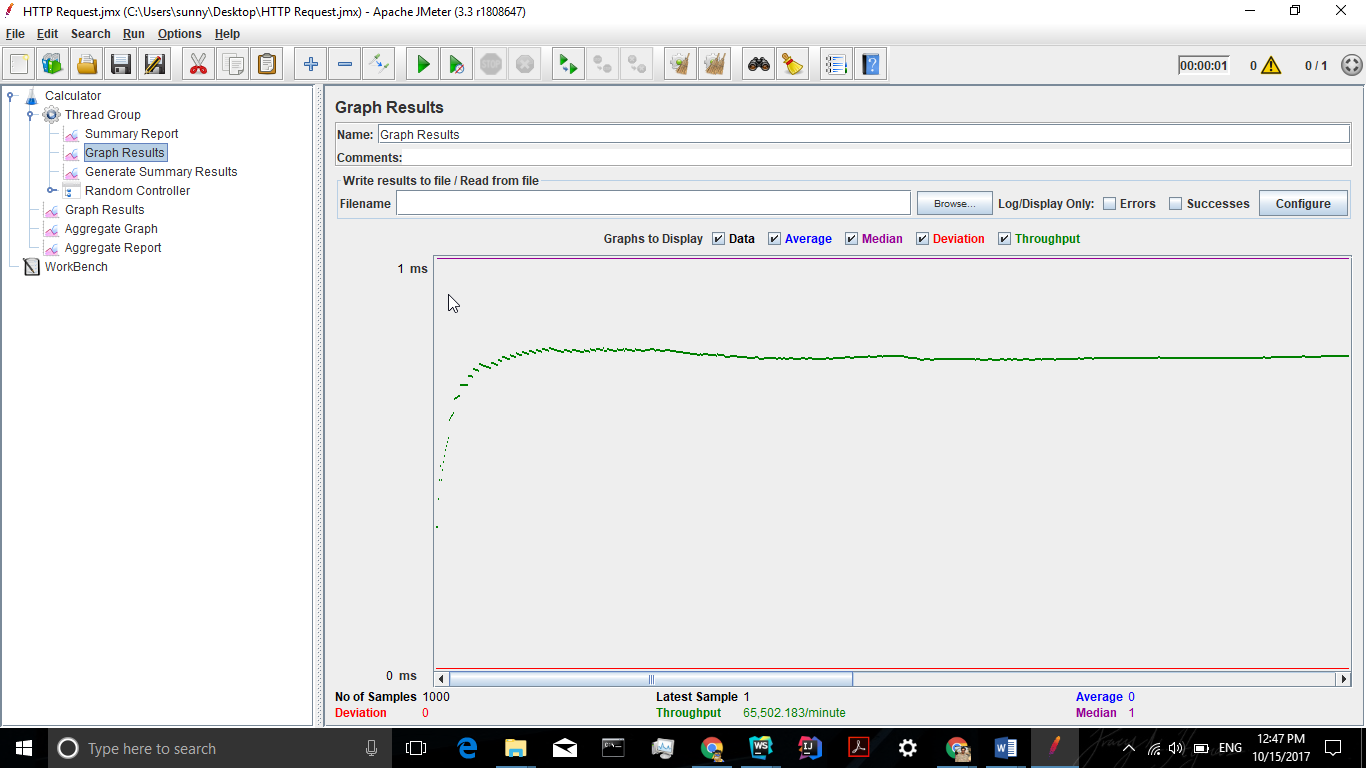


## **Testing and Performance:**

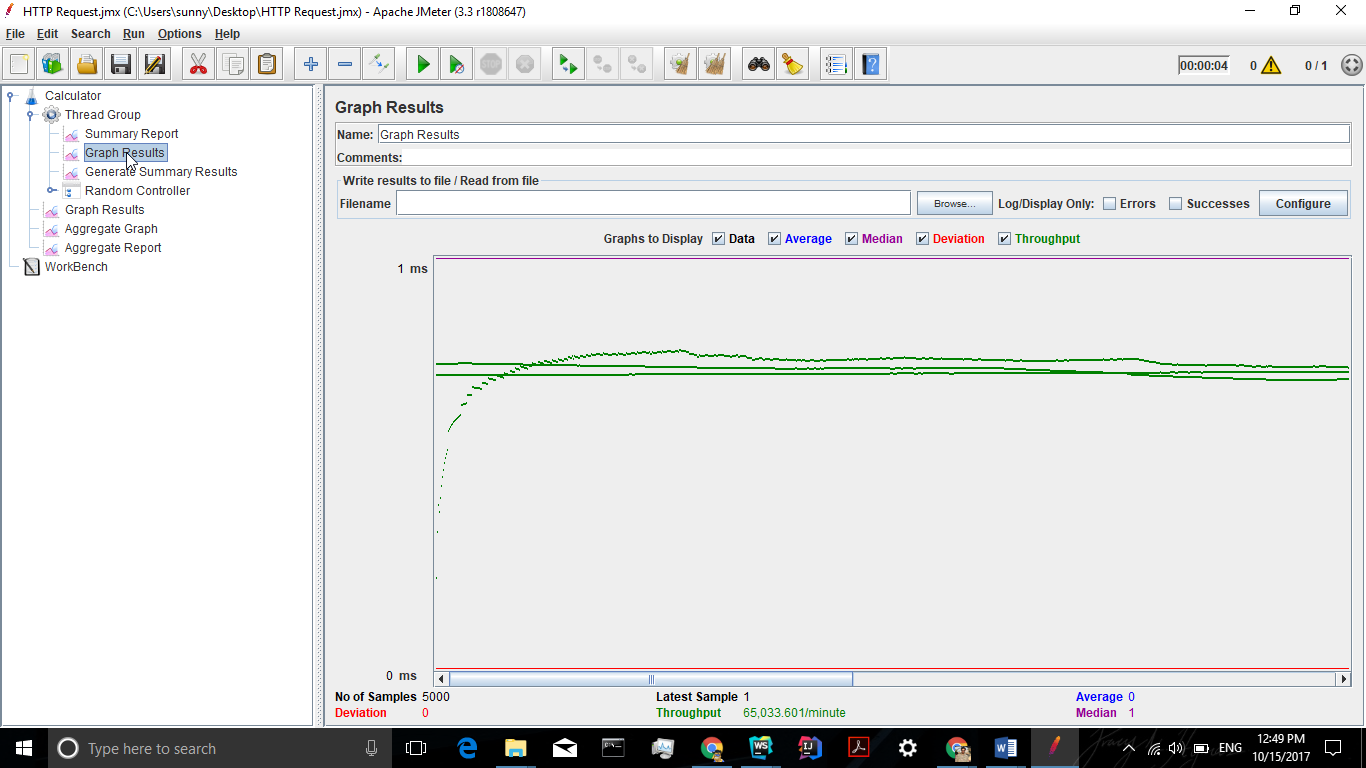
**Jmeter Load Testing**

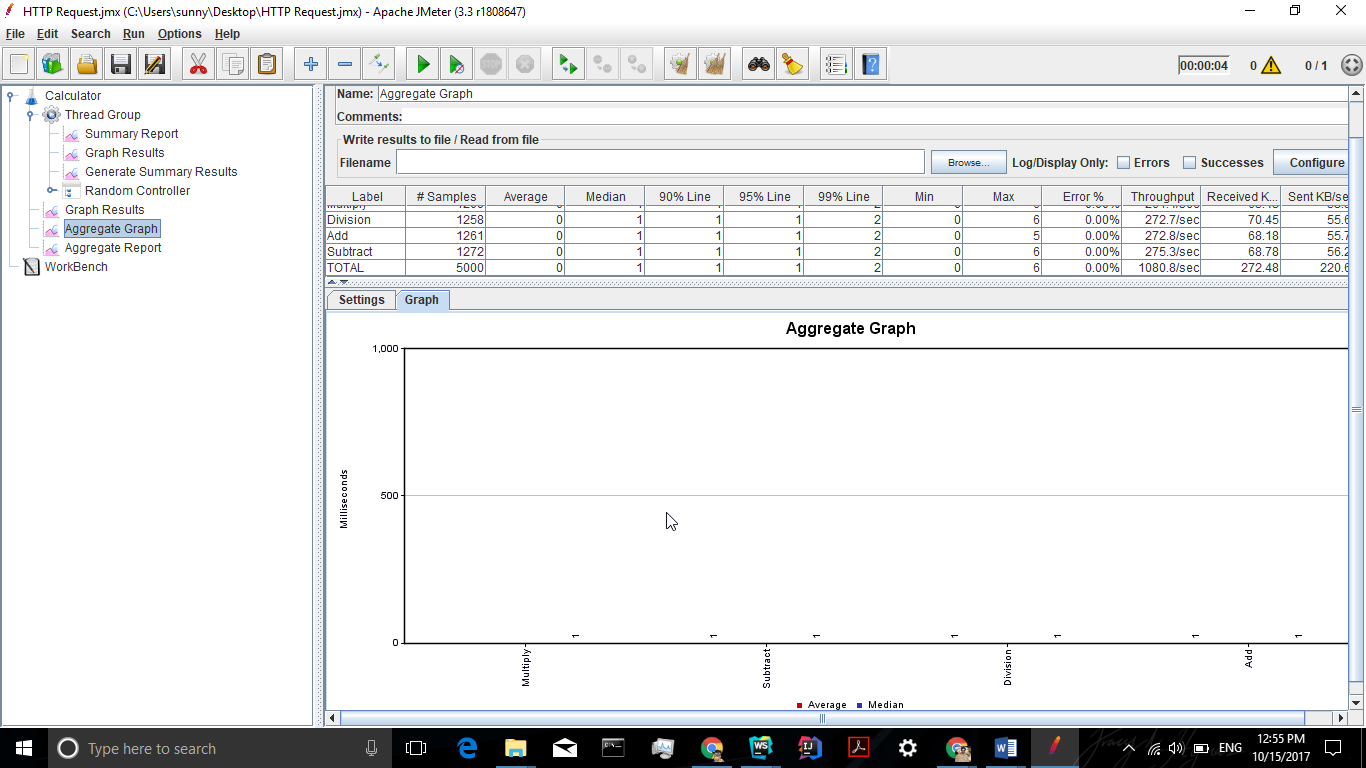
1. ***1000 calls on randomly selected tasks:***

******

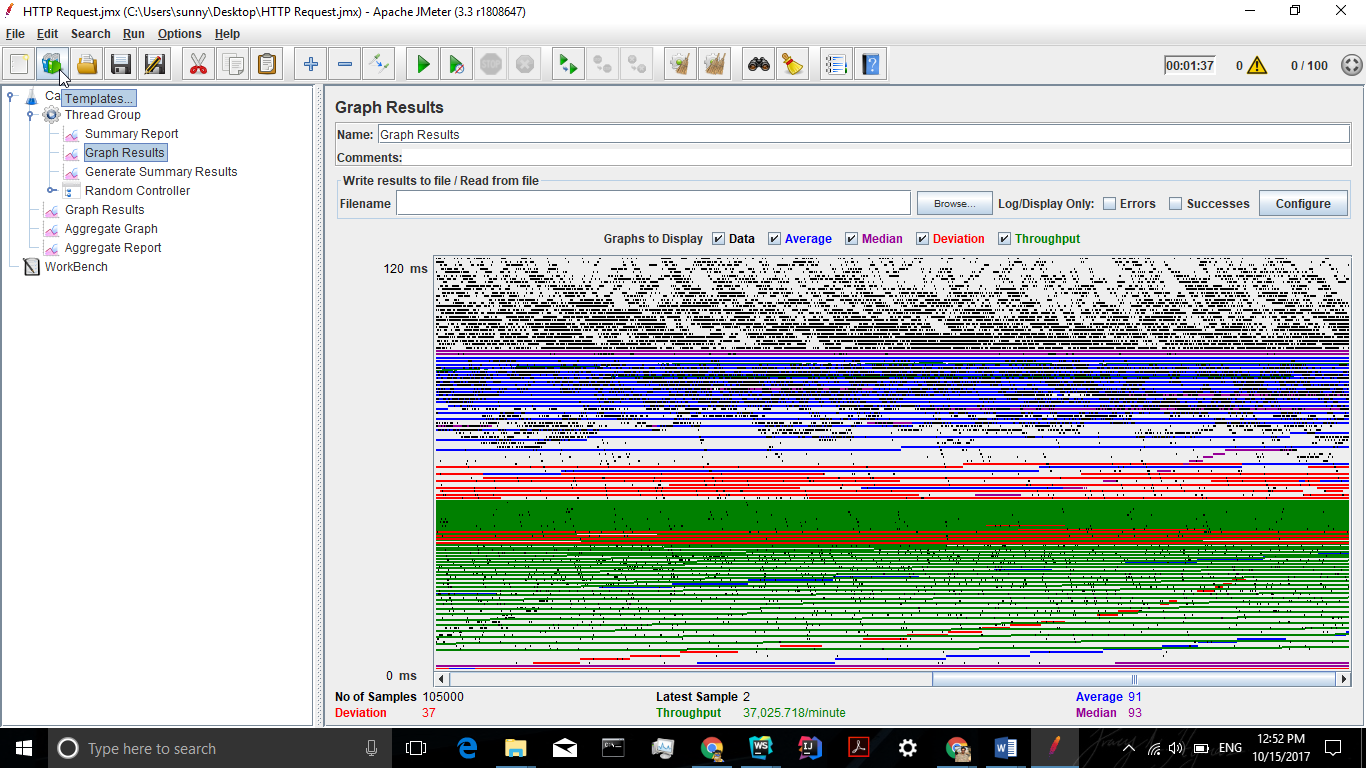
******

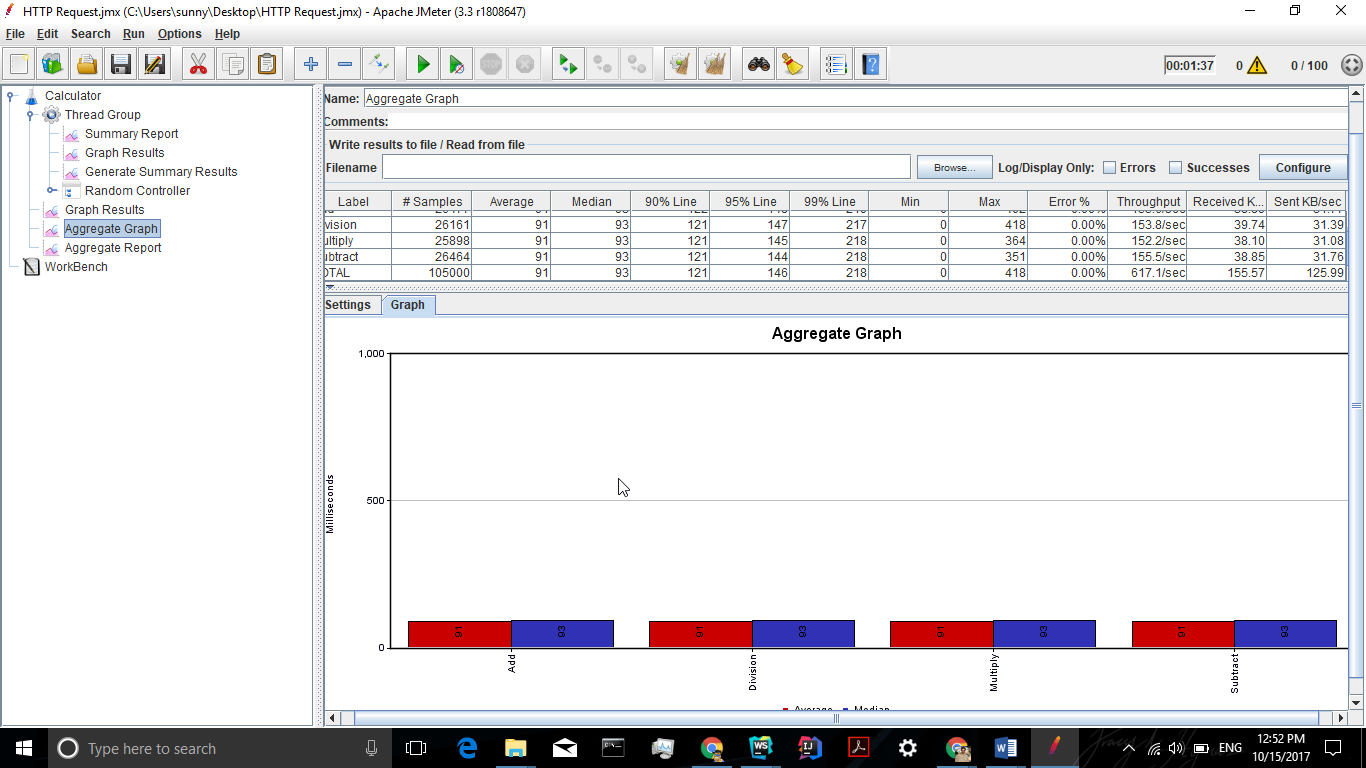
1. ***5000 calls on randomly selected tasks:***

******

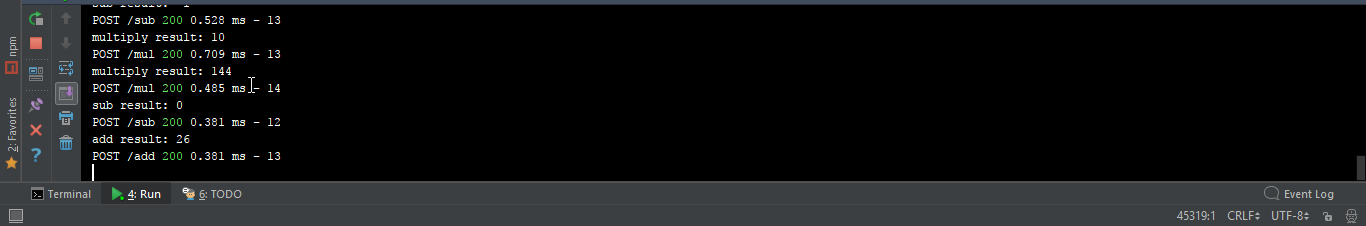
******

1. ***100 concurrent users and 1000 calls on randomly selected tasks:***

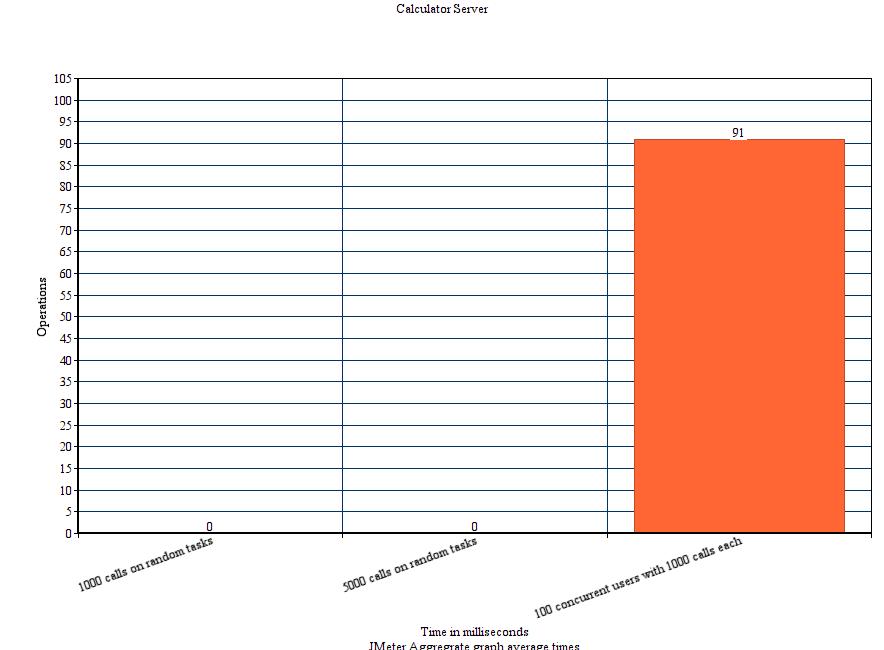
******

******

**Server Screen :**



## **Analysis of performance :**

******

This graph shows that when we test the application with multiple concurrent users performing many concurrent tasks the average response time by server increases.

**Reason** :

Average Response time refers to the average amount of time Server takes to return the results of a request to the user.

The Average response time is directly proportional to the number of users and number of requests submitted. As a result, the average response time increases when we test the application with multiple requests by multiple concurrent users.

Also, as single and simple server is used to test the application it performs accordingly and performance can be increased by hosting application in a multiple server architecture.

# PART-2 ***DropBox\_Prototype***

**Introduction:**

*Goals and purpose of your system:*

Designed and developed a prototype of dropbox to demonstrate REST web services. The application provides features like,

* File sharing and file uploading like DropBox.
* It provides the functionality of allowing users to Signup, Login, upload files, share files, star files and logout.
* On successful login it shows a list of files shared with user, files uploaded by user and files starred by user.
* It provides an overview of users’ profile

## **System Design:**

Applications uses a simple Client-Server architecture where there are as many as 10 React components and 8 API’s to support different functionalities.

**Different pages and their functionalities:**

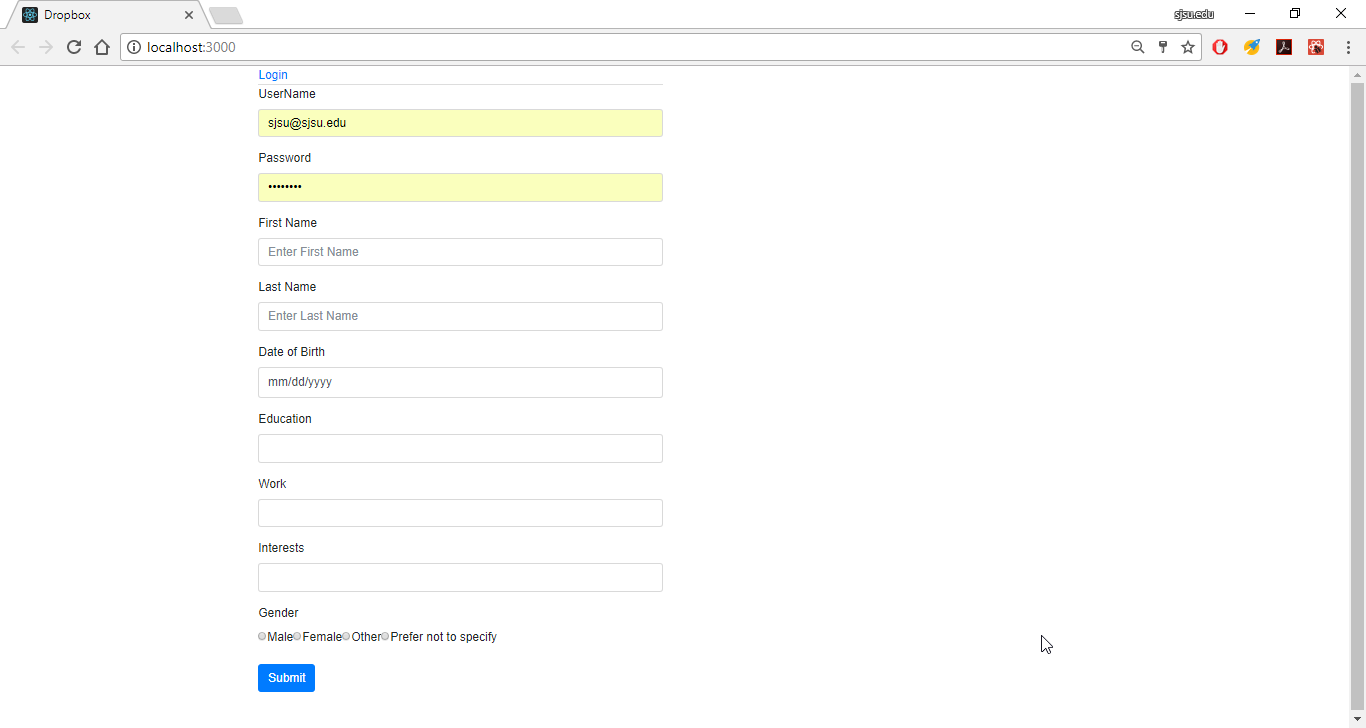
1. **Signup page** : Allows the user to sign up for the application so that they can use it to share and upload their files. It takes a variety of user input to provide a personalized experience on login.
2. **Login page**: Allows the user to login to the application and create user session. The user gets redirected to the home page on successful login or shown a validation message on incorrect inputs.
3. **Home page**: It serves multiple functionalities of listing user’s uploaded and starred files. It also shows the files/folders shared by others with the user. Also allows the user to upload multiple files and create folders.
4. **Profile page**: Shows details regarding user profile.

Stack:

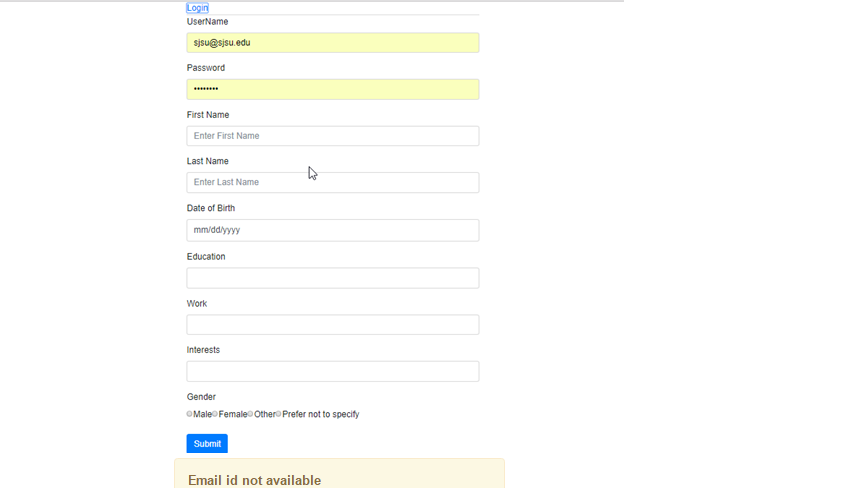
* Database used: MySQL
* Front-End: HTML, Bootstrap and ReactJS
* Server-side: Node JS and Express JS
* Package used for encrypting password: bcrypt (<https://www.npmjs.com/package/bcrypt>)
* Testing: MochaJs and J-Meter.

## **Results**

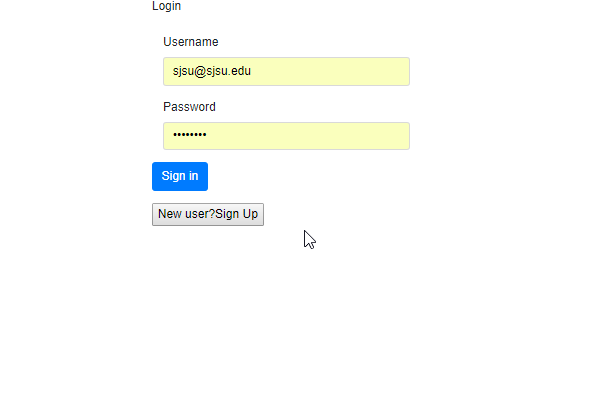
**Screen captures of dropbox prototype:**

**Register**: 

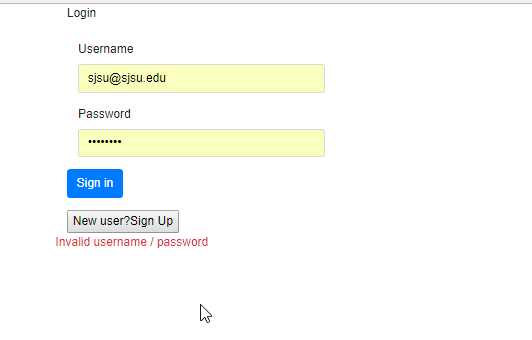
**Validation**: If user tries to register with already existing email-address, it will display message as show below:



**Login:**

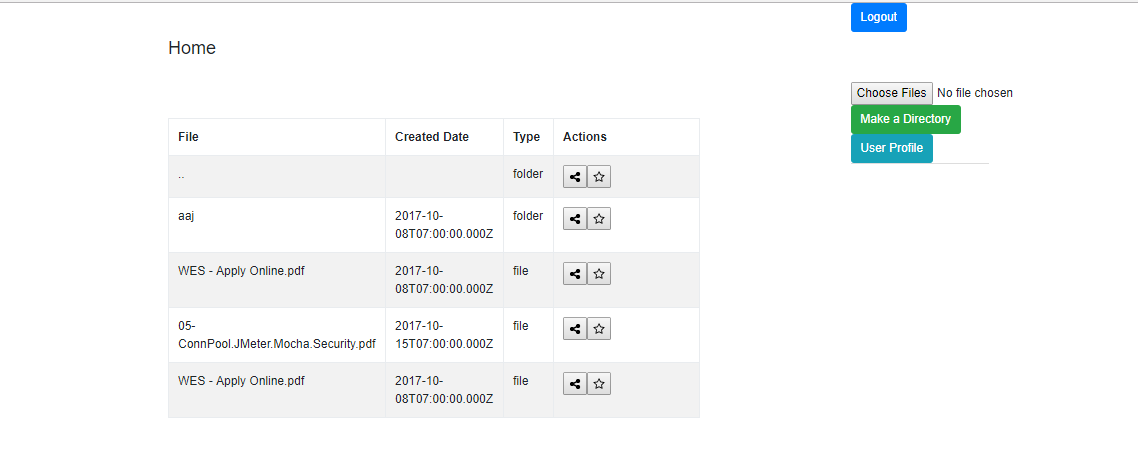
****

**Validation: If incorrect username or password**

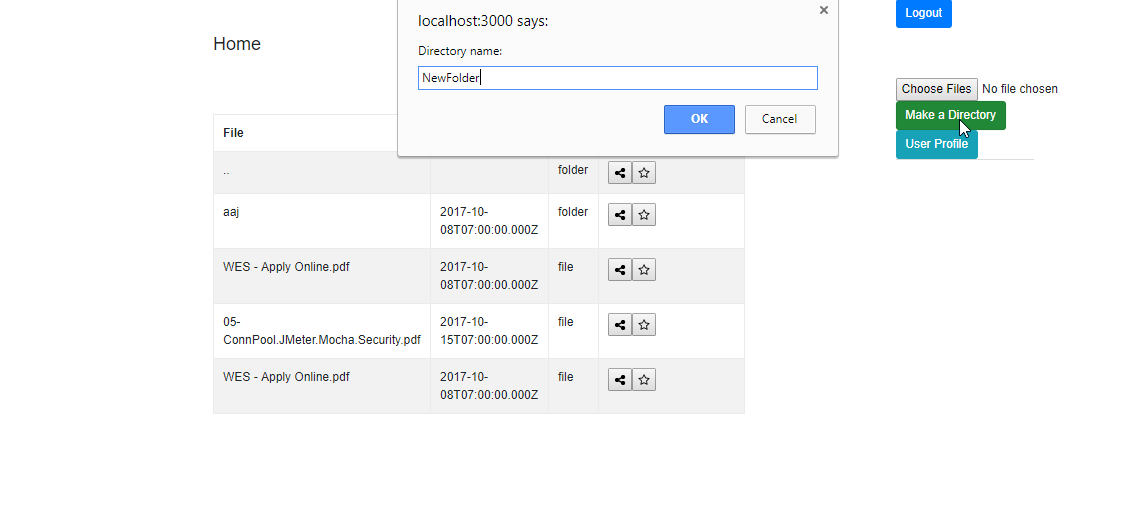
****

**Home:**

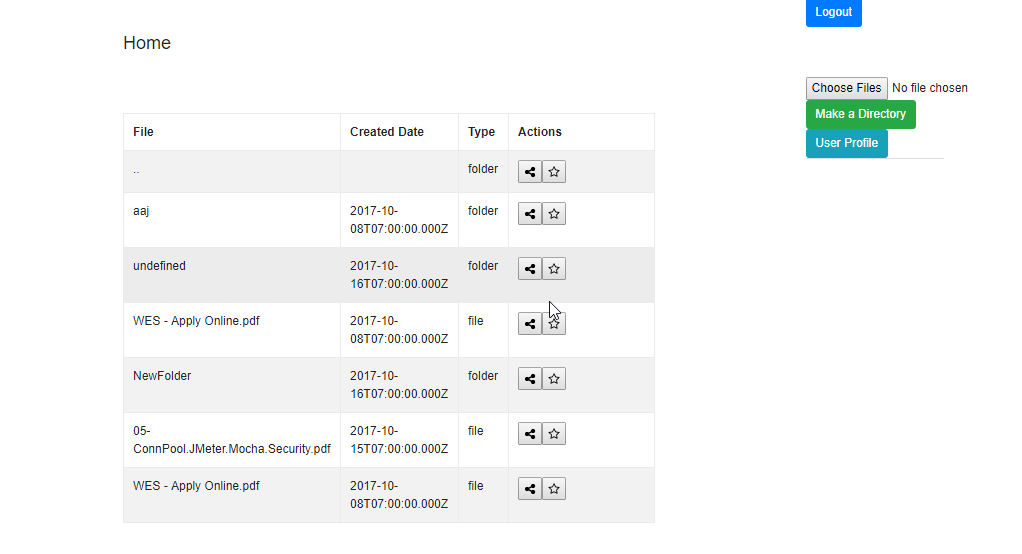
User gets to the home page after successful login. The home page shows a list of files that are uploaded and starred by user. It also has options to logout of system, create a directory or to view user profile.



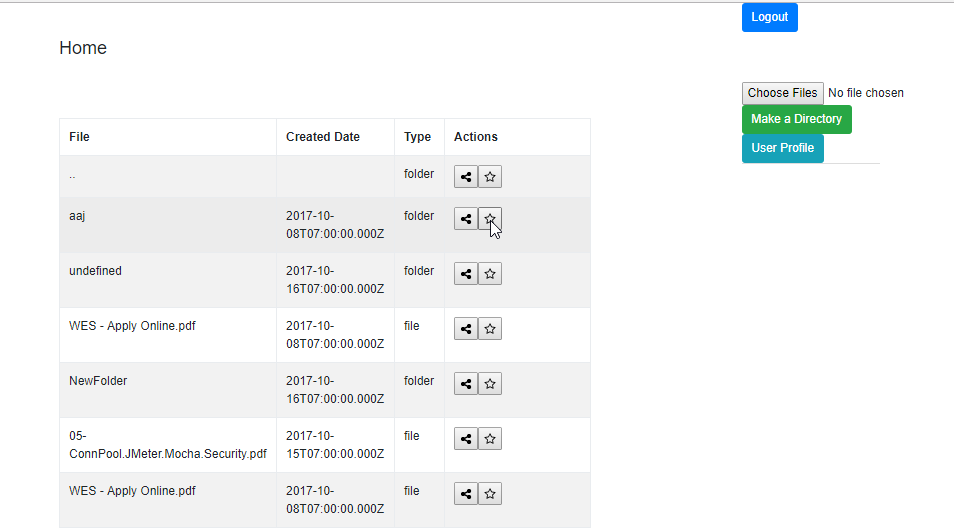
**Create a directory:**

****

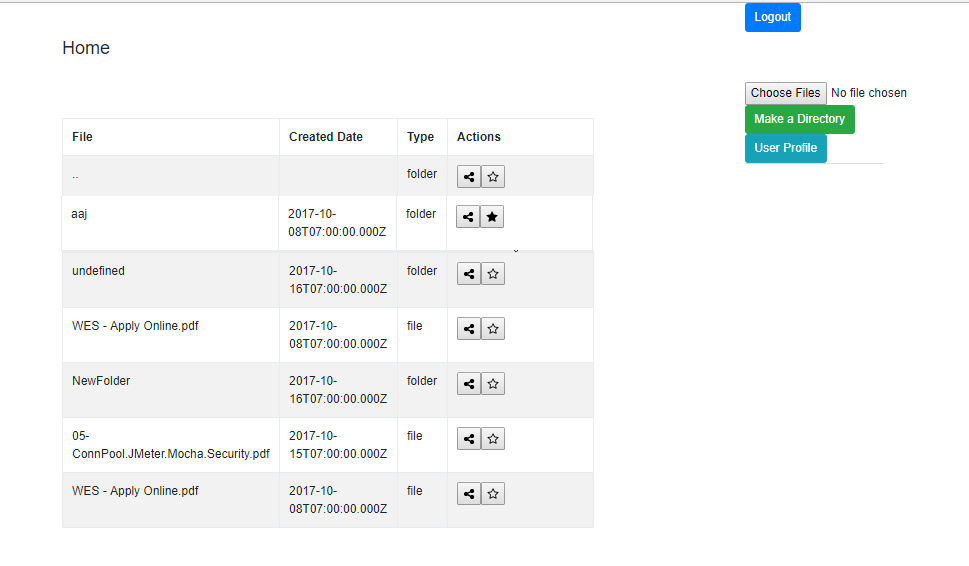
**Result:**

******

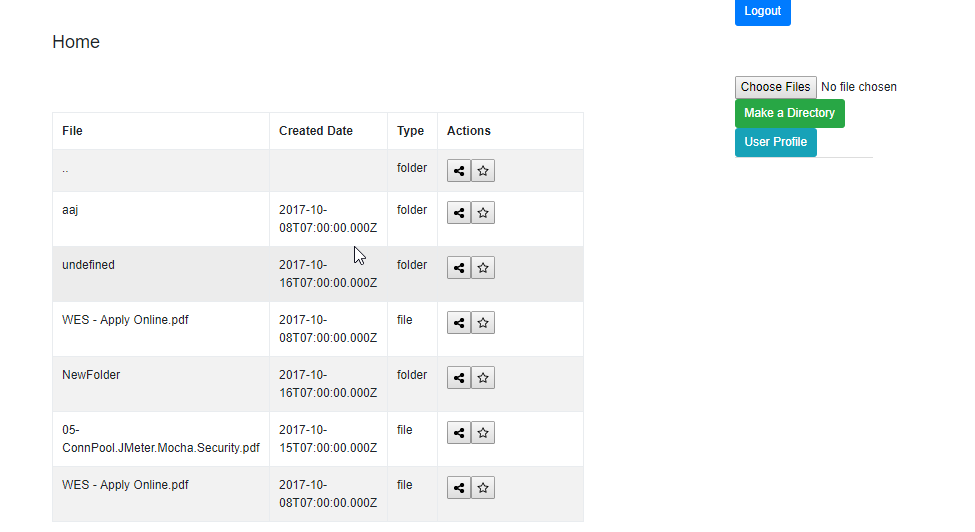
**Star a directory:**

****

**Result:**

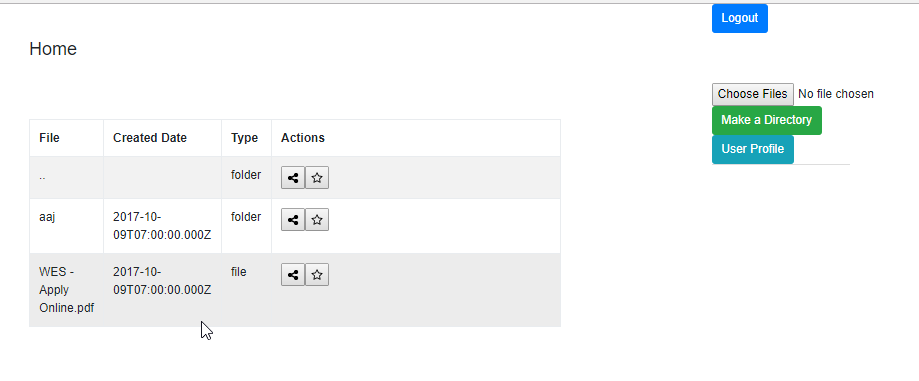
****

**Go inside Directory:**

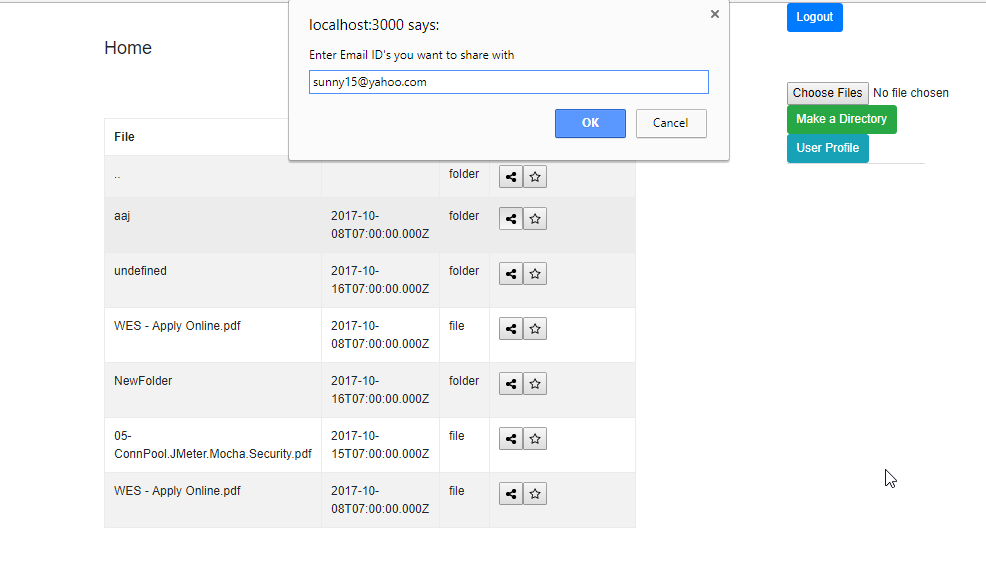
****

**Result:**

Lists files inside the directory. And allows user to upload files inside the directory.

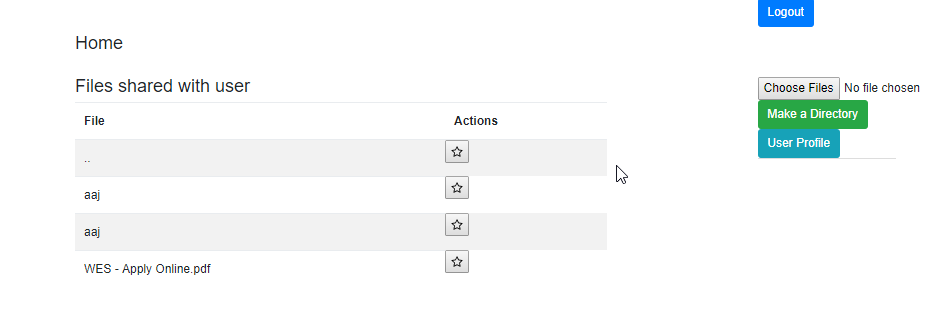
****

**Share file/directory**

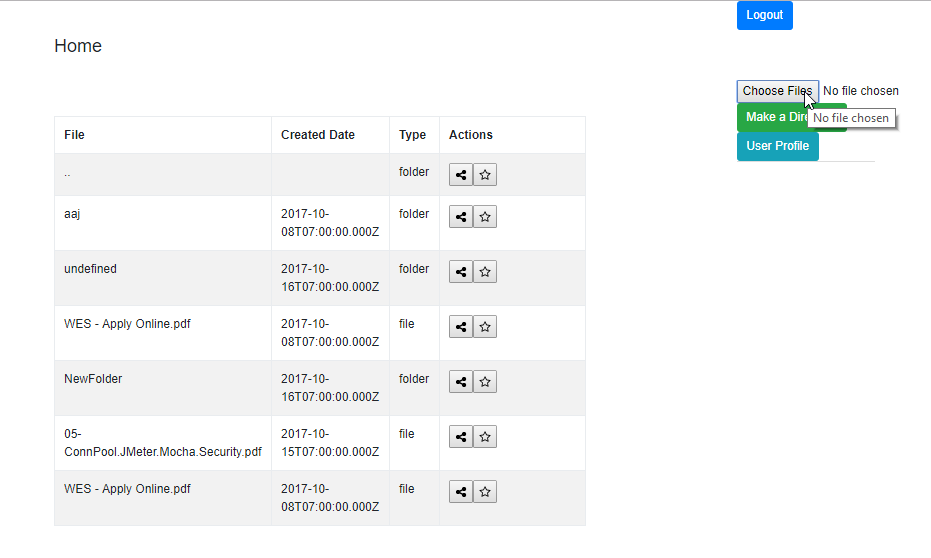
****

**Result:**

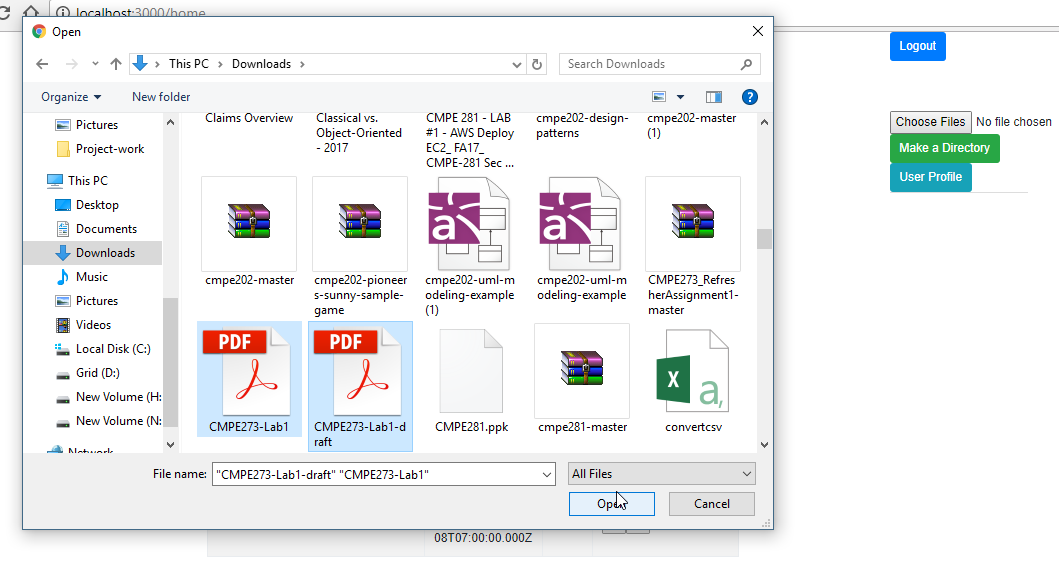
When I login with [sunny15@yahoo.com](mailto:sunny15@yahoo.com) I will be shown another table with files shared with the user.

****

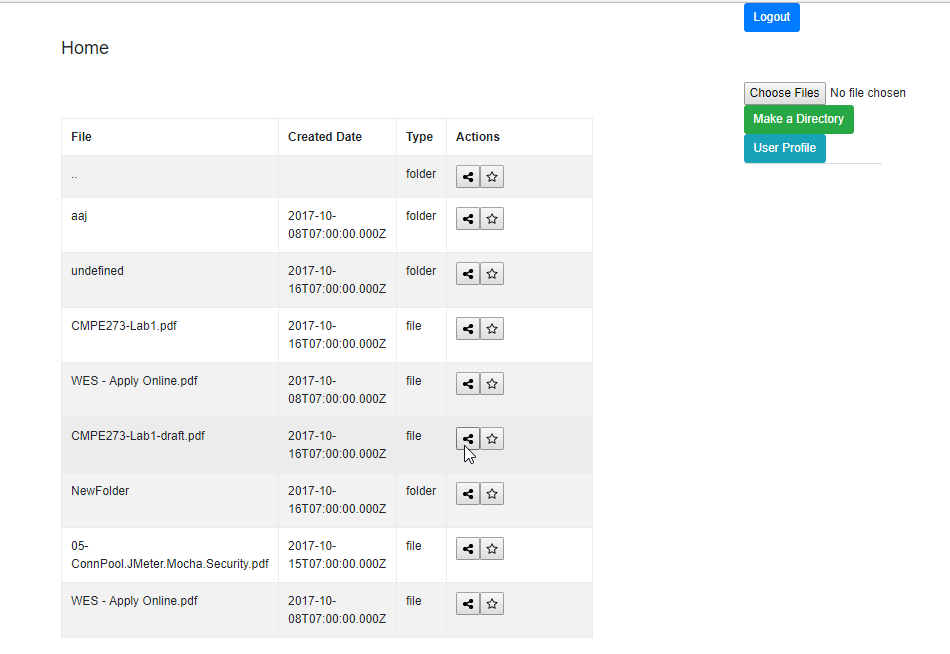
**Upload files:**

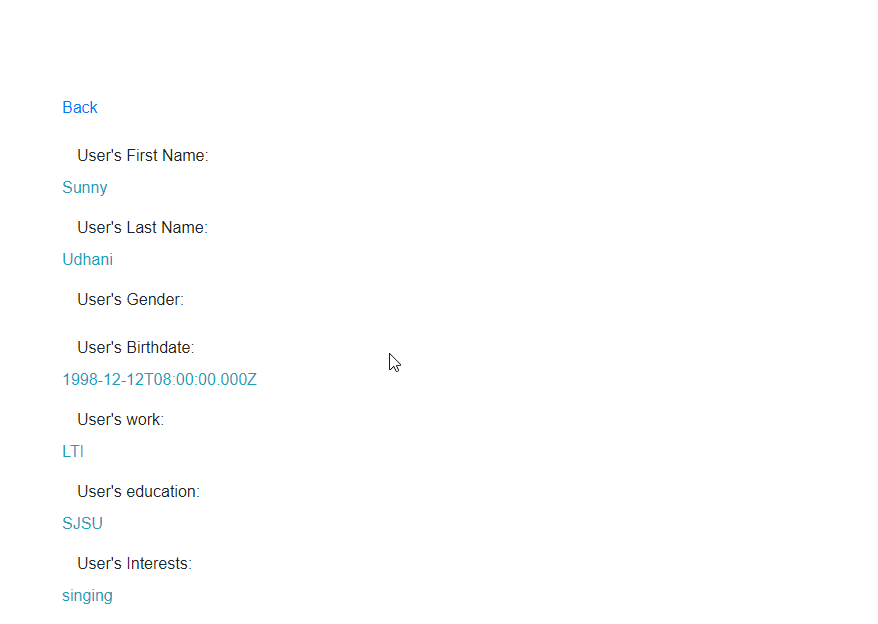
****

**Select files to upload :** two files selected for uploading.



**Result:**

****

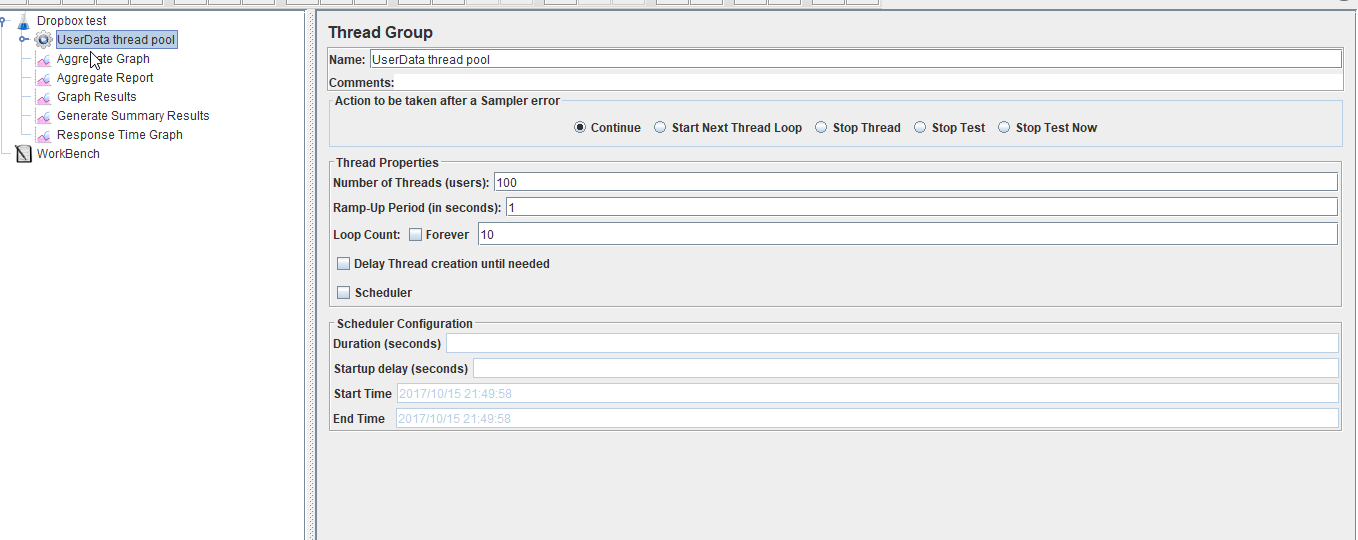
**User Profile:** Display the logged-in user’s details in a page ****

## **Performance**

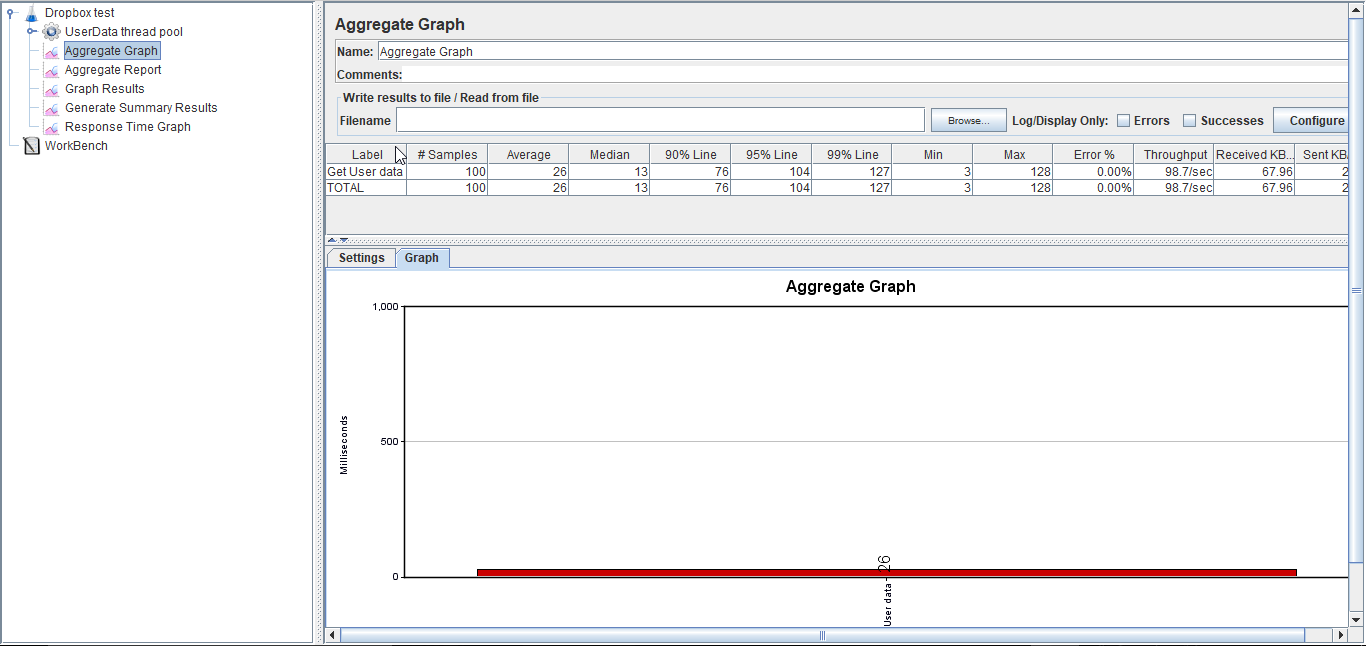
**JMETER TESTING**

1. For 100 concurrent users:

**Setup**-

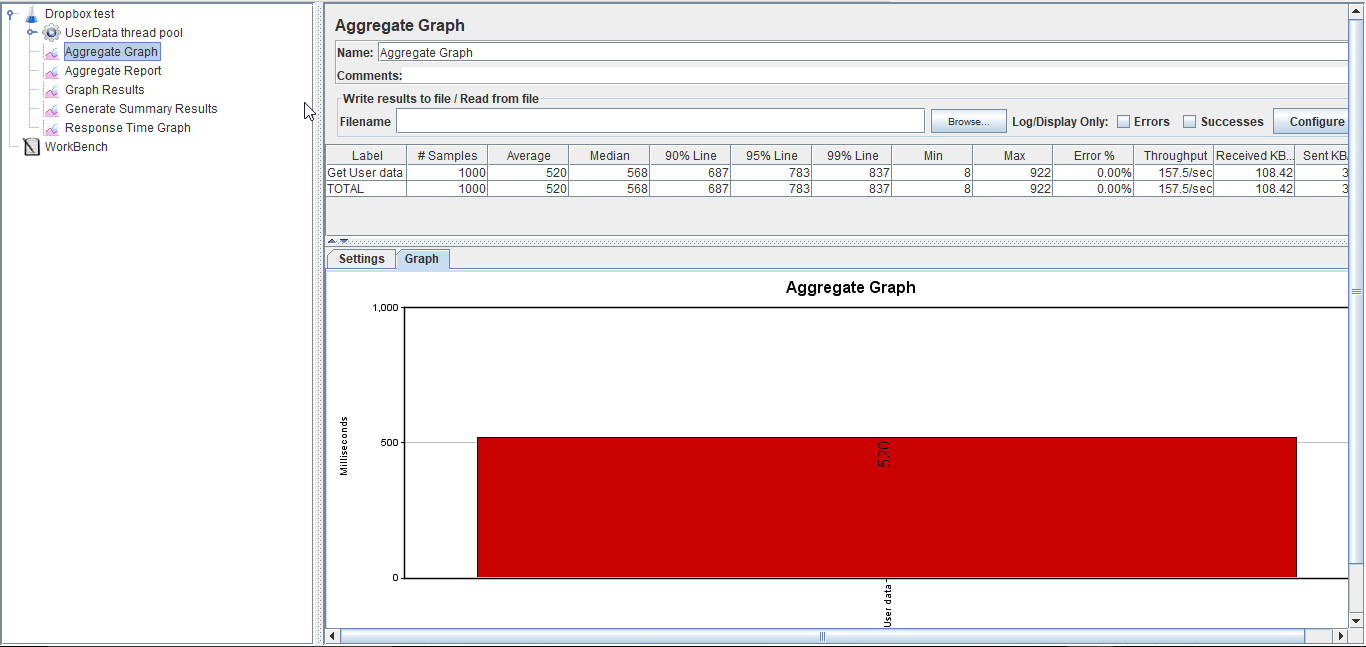


**With Connection Pooling**:



****

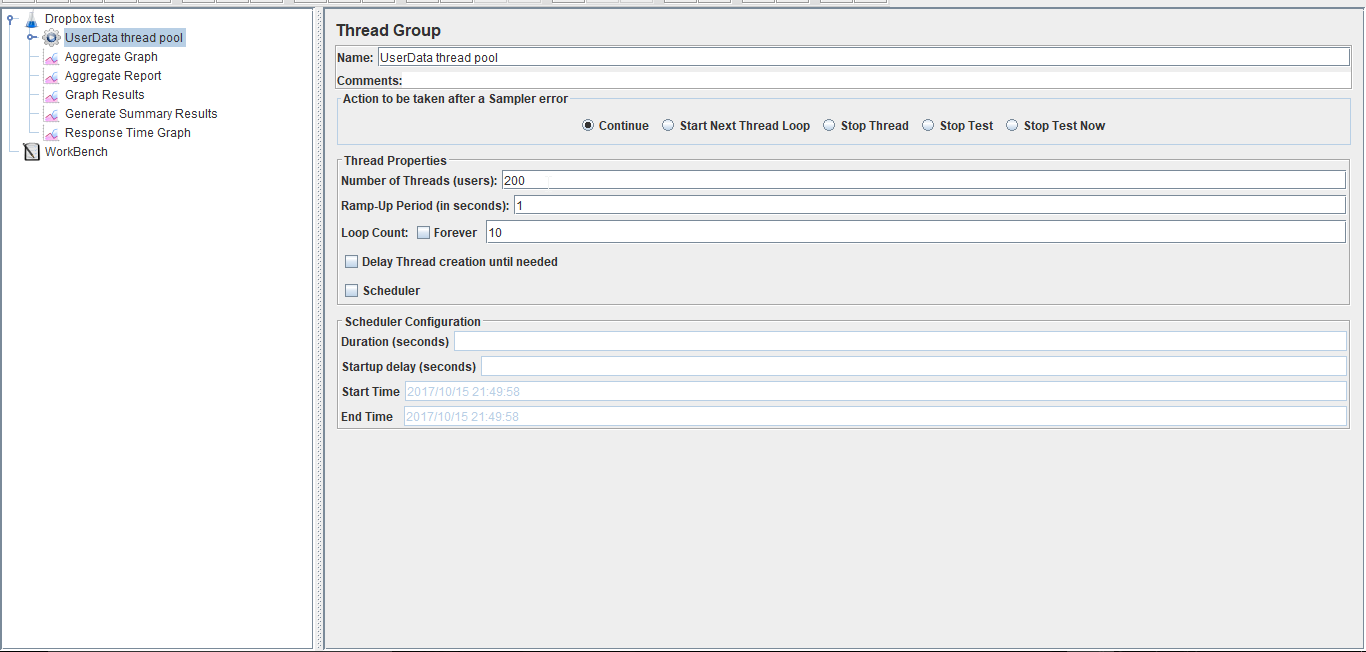
**Without Connection Pooling**:



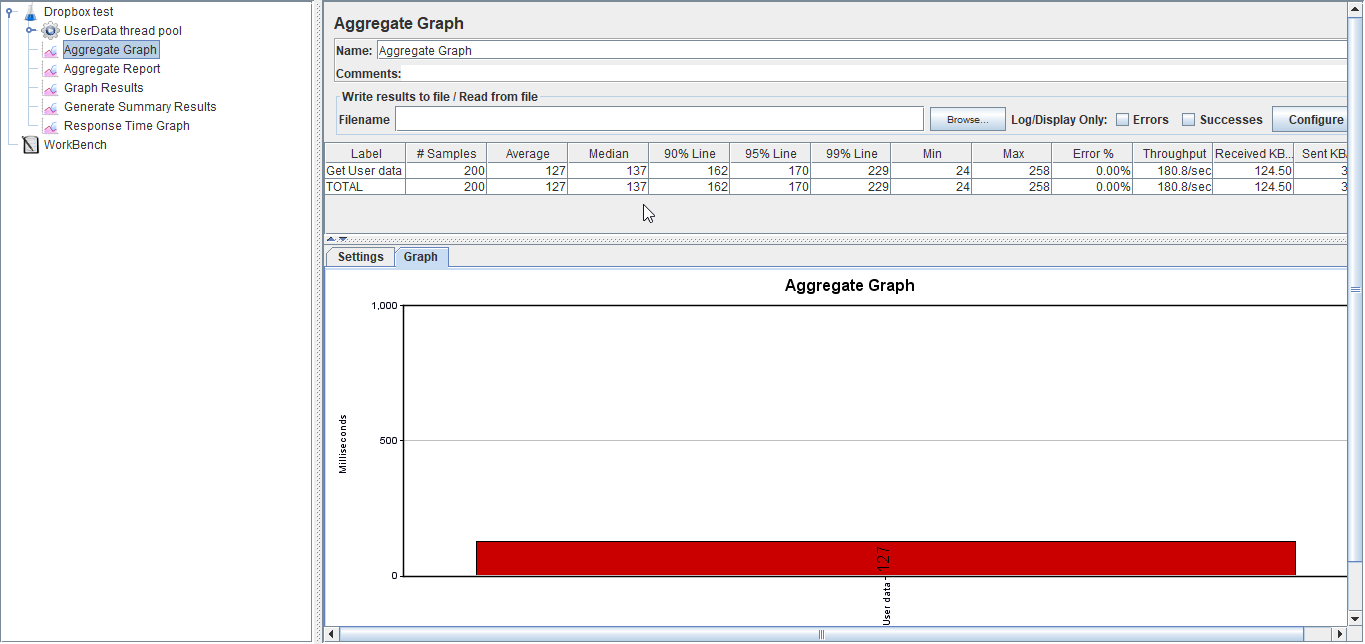
****

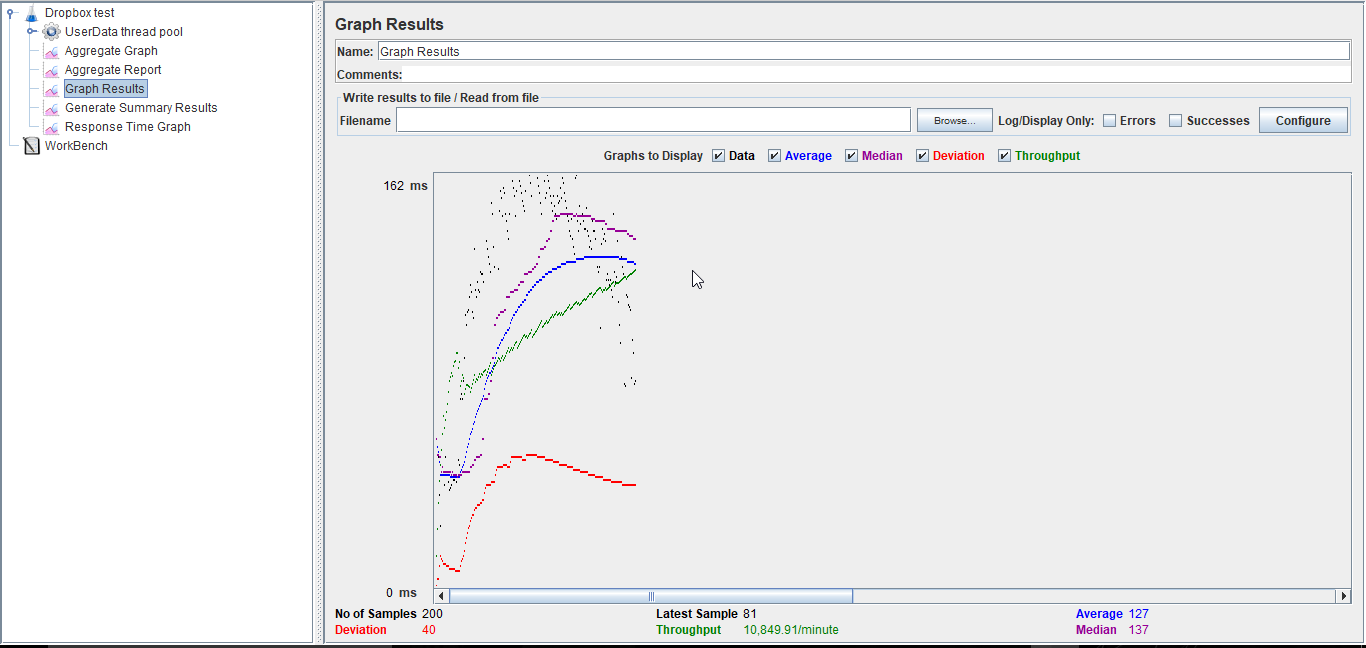
1. **For 200 concurrent users.**

Setup –

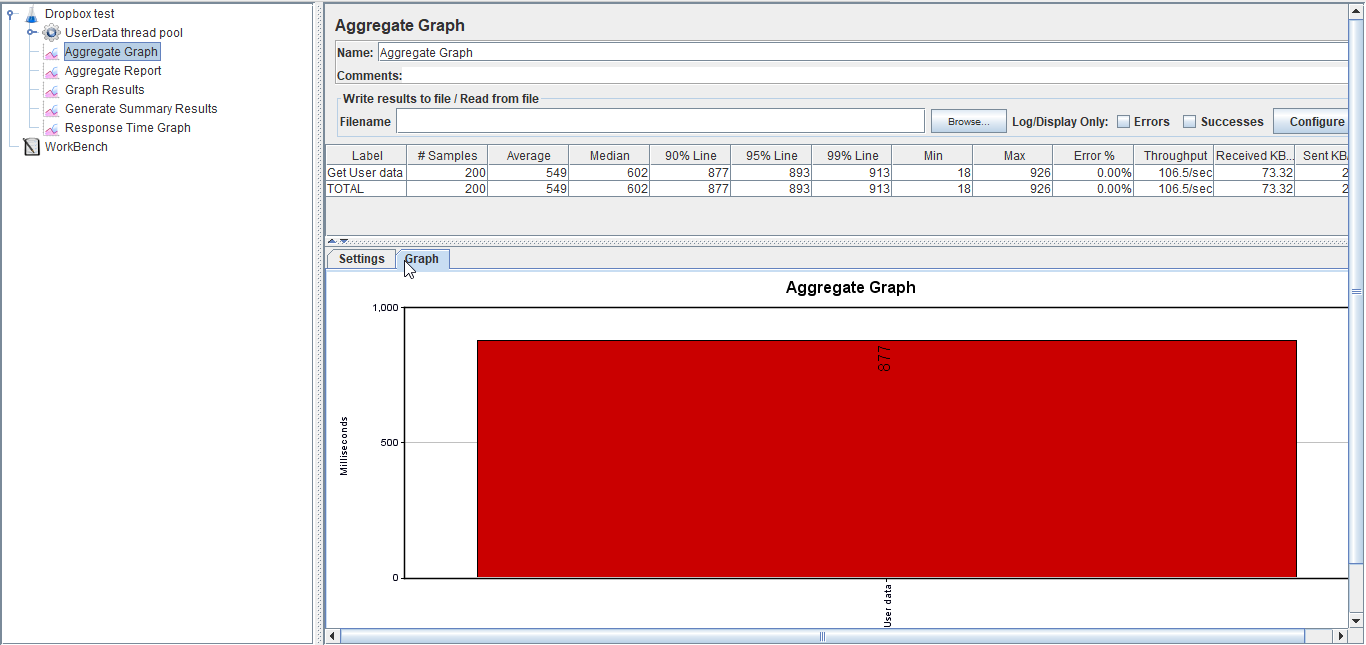


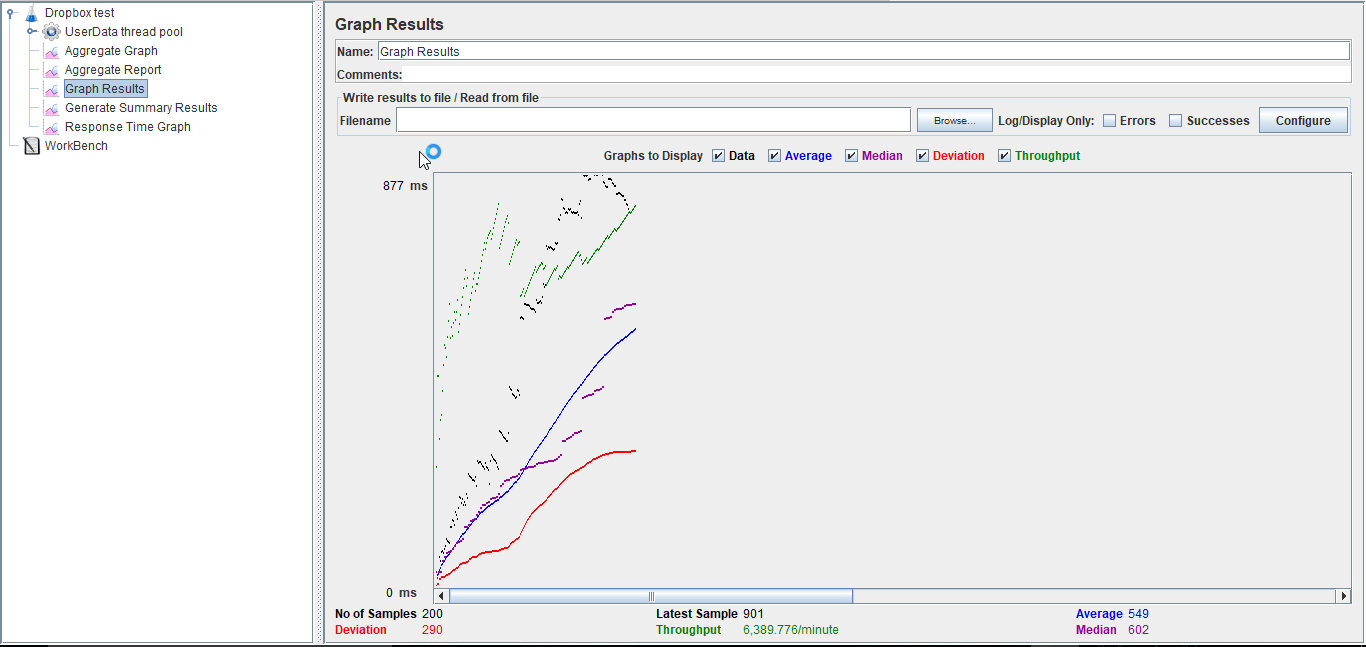
**With Connection Pooling**:





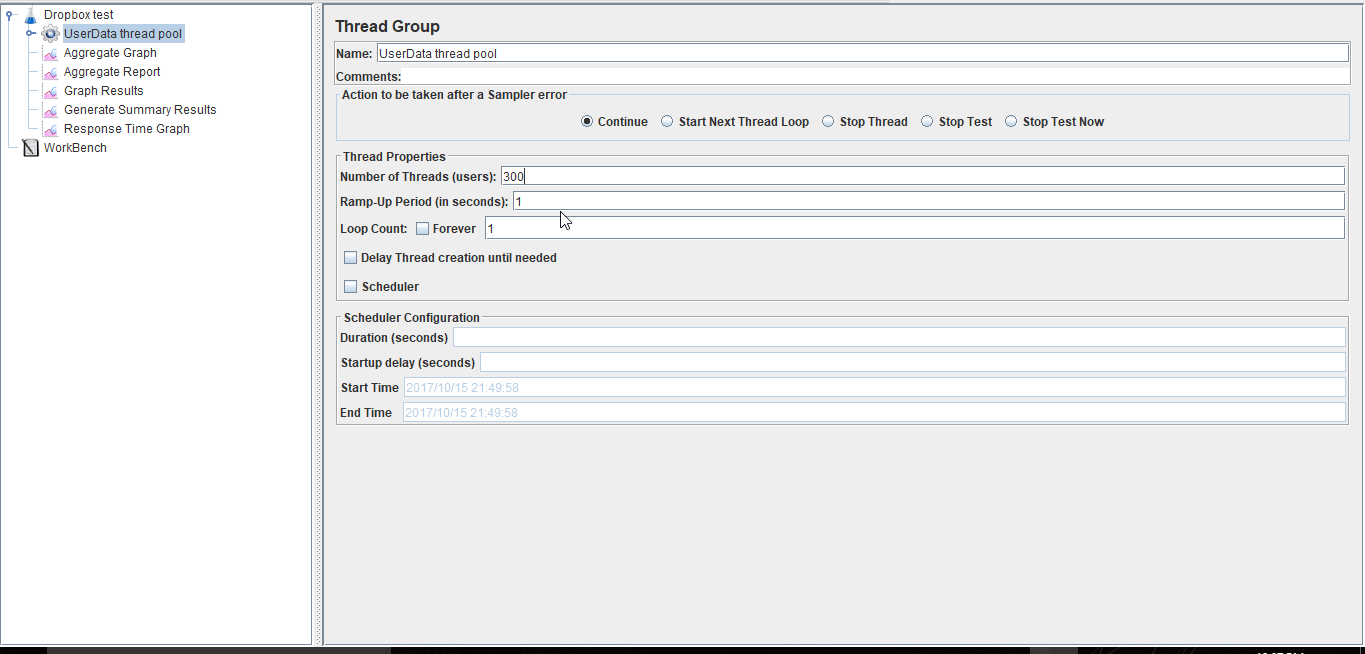
**Without connection pooling**:



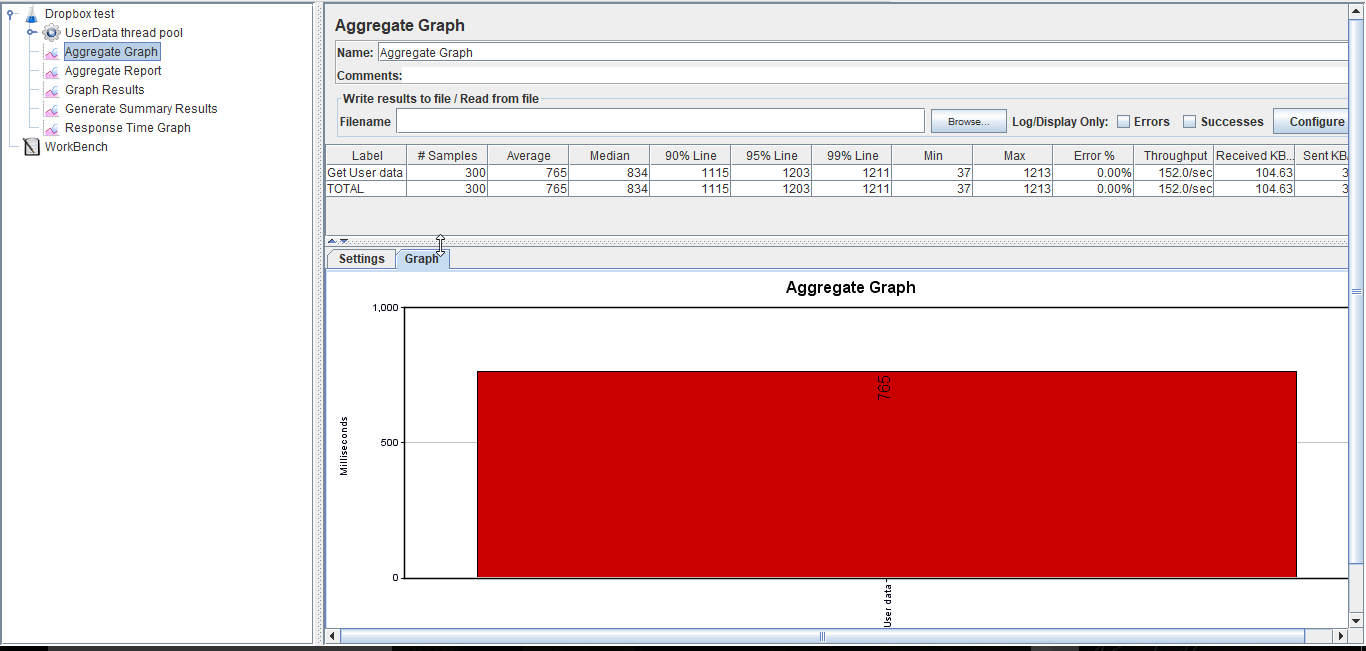


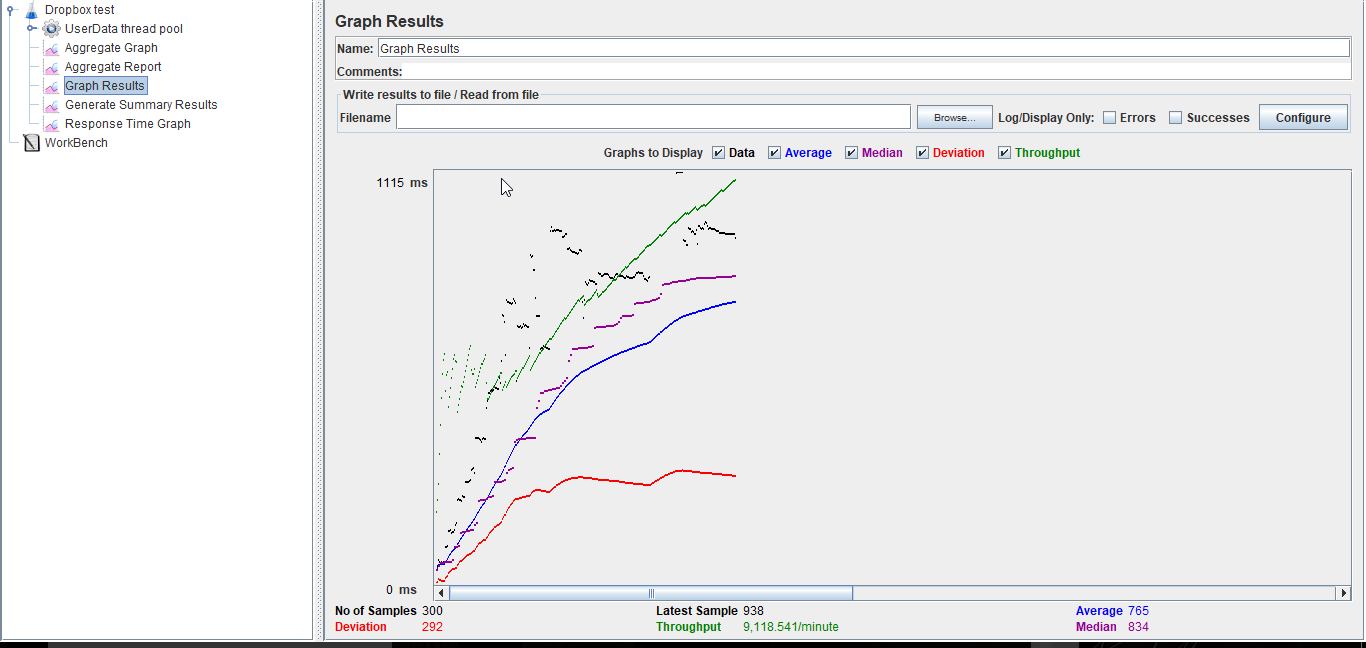
1. **For 300 concurrent users**:

**Setup** –

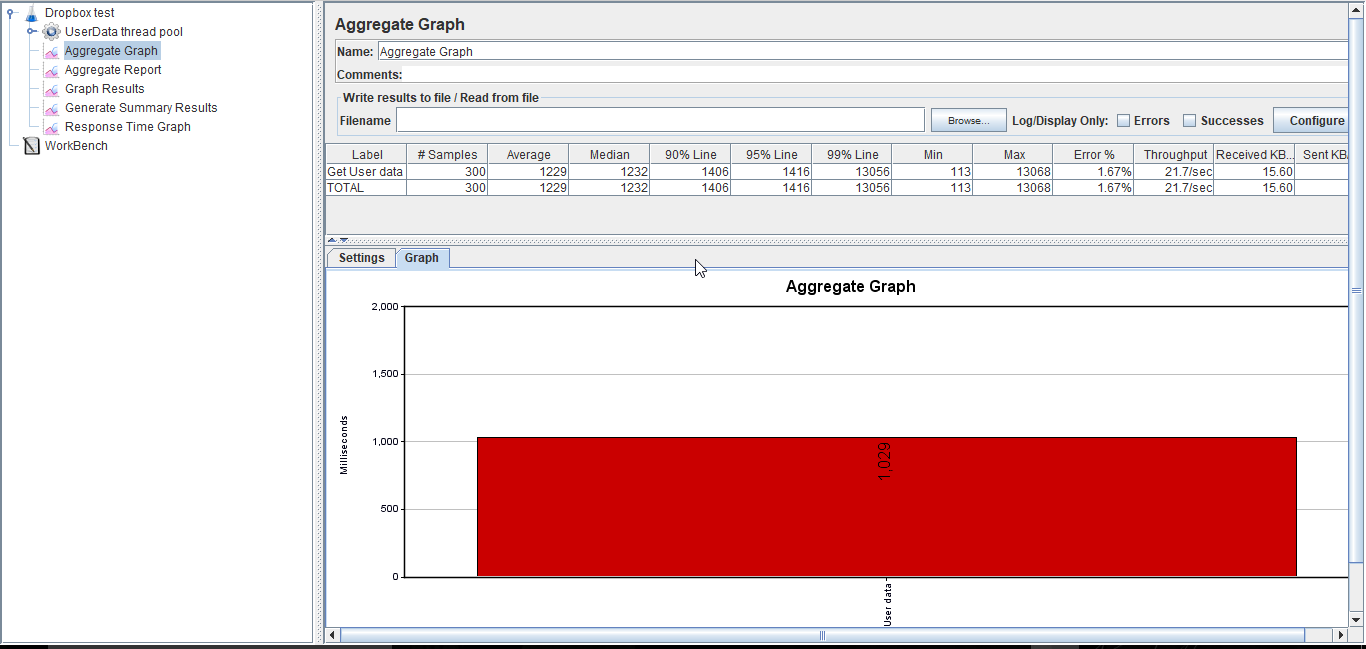


**With connection pooling**:





**Without Connection Pooling**:

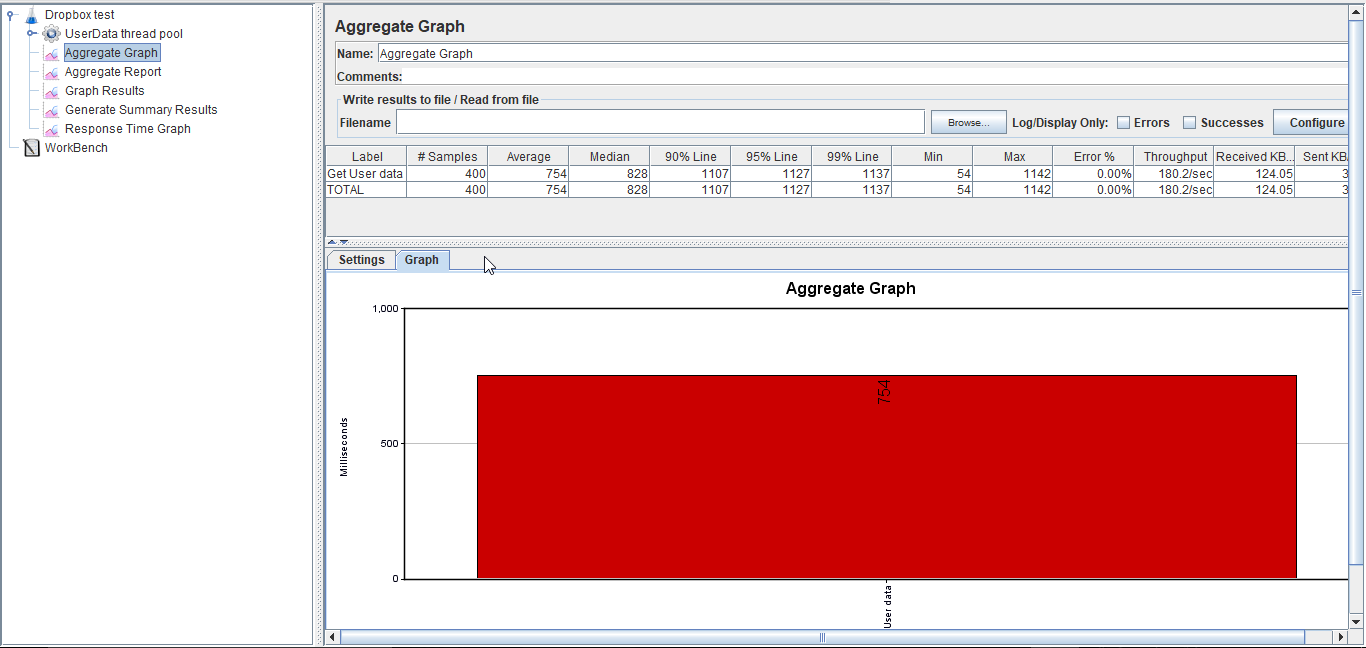




1. **For 400 concurrent users**:

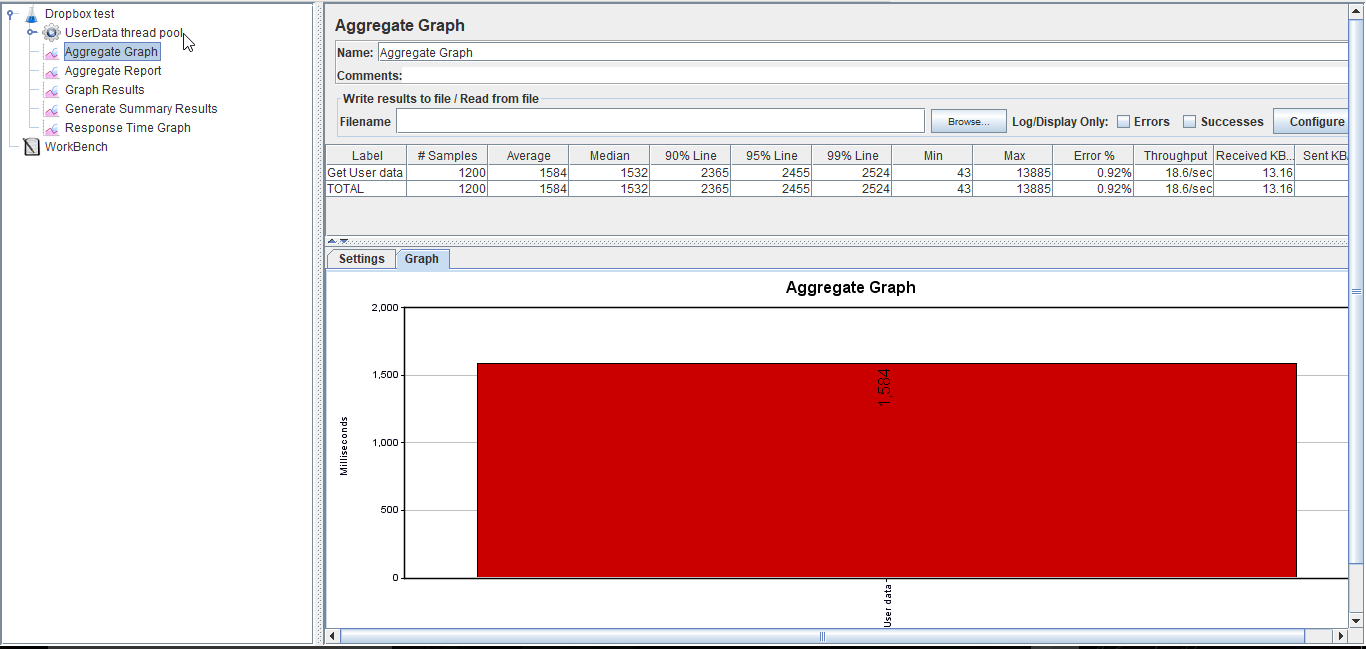
**Setup** –

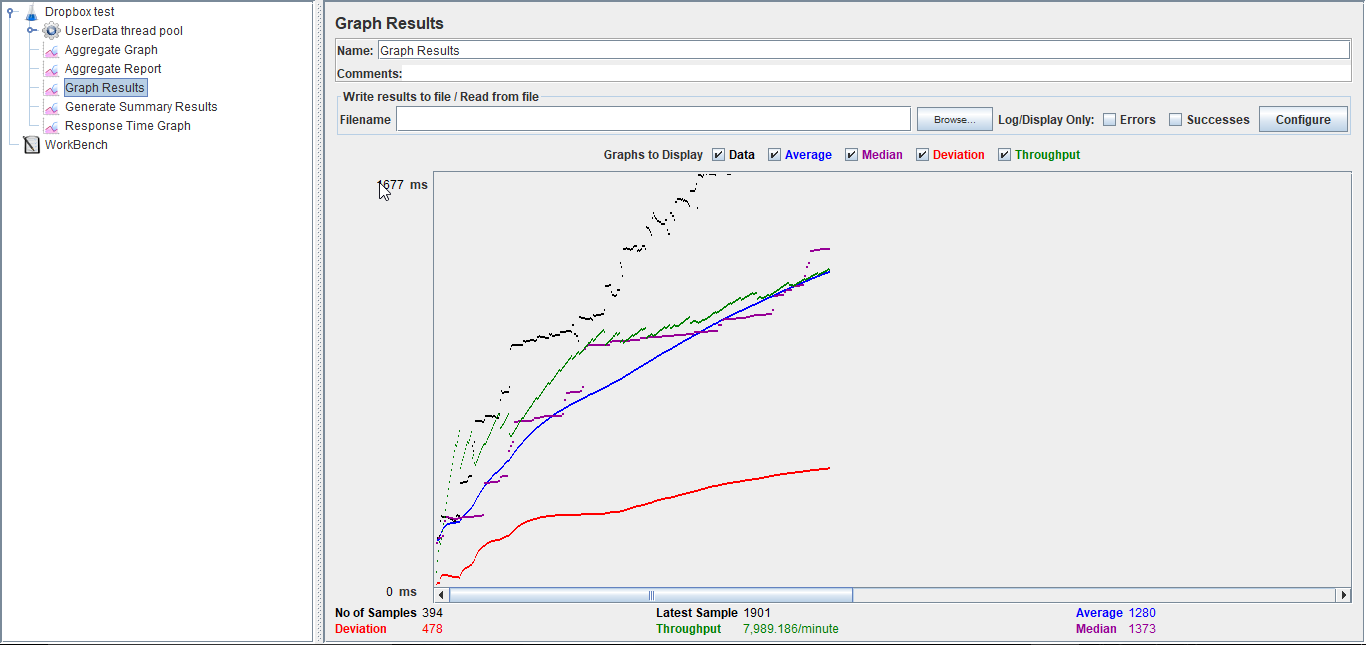
**With connection pooling**:





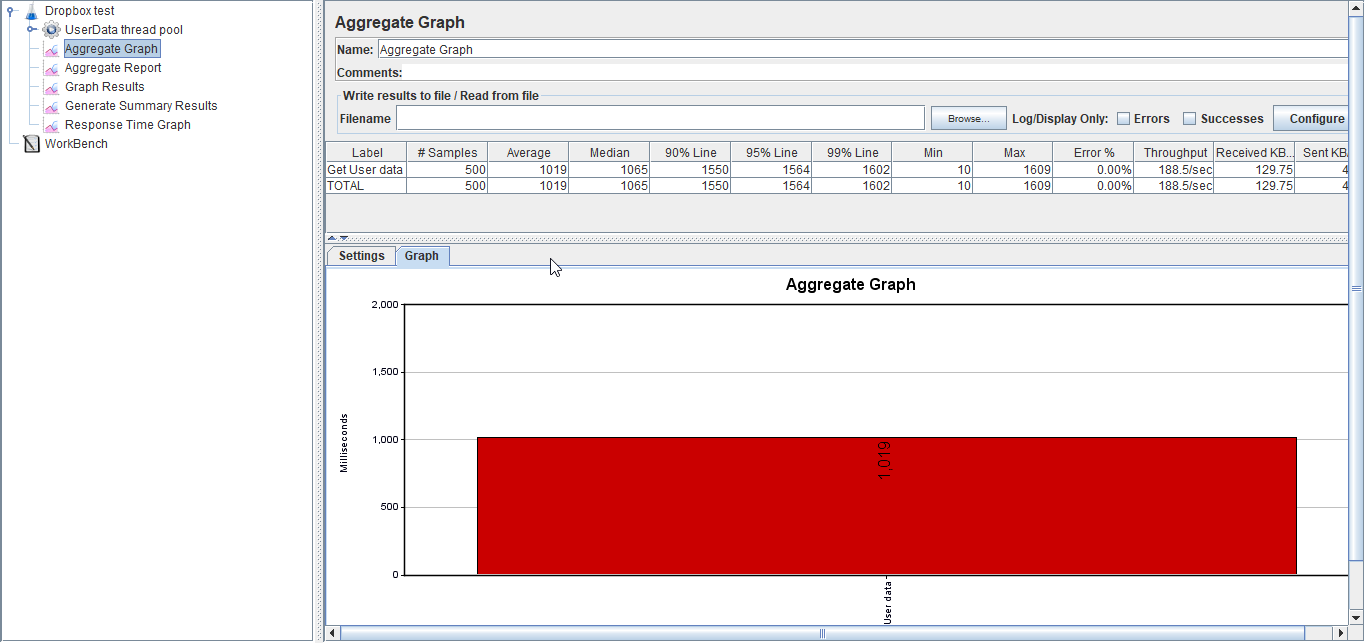
**Without Connection Pooling**:





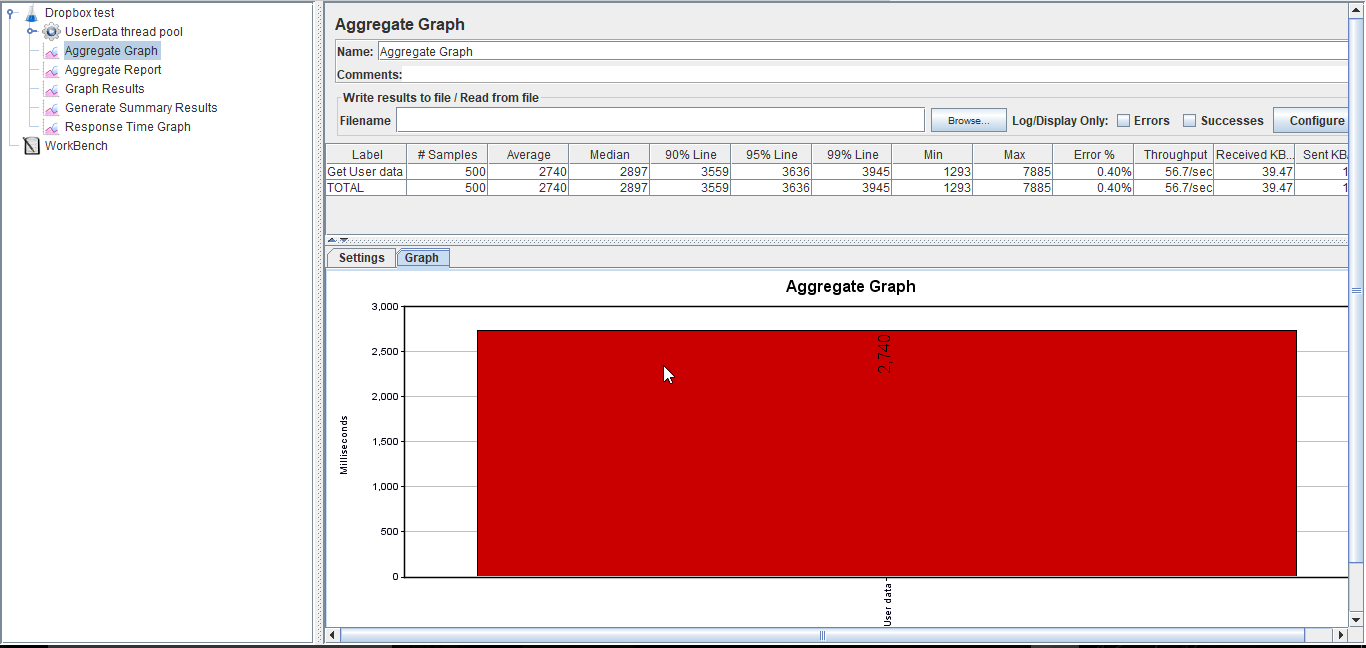
1. For 500 concurrent users:

**With Connection pooling**:



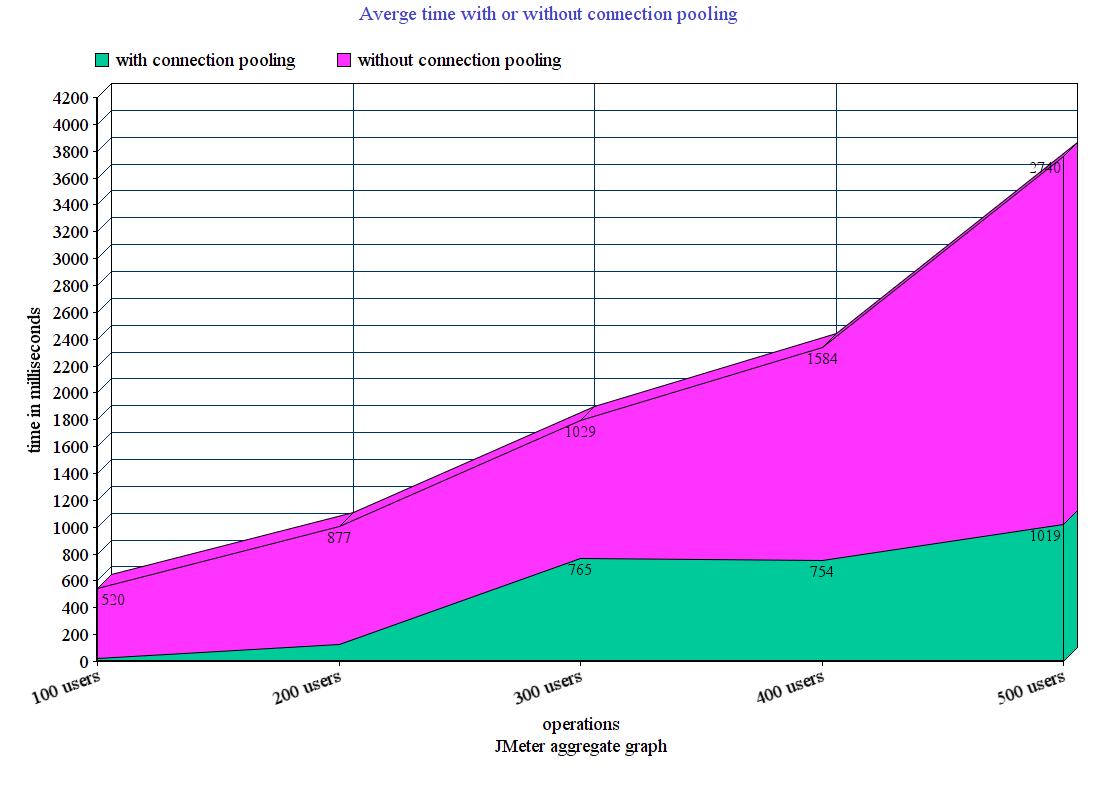


**Without Connection Pooling**:

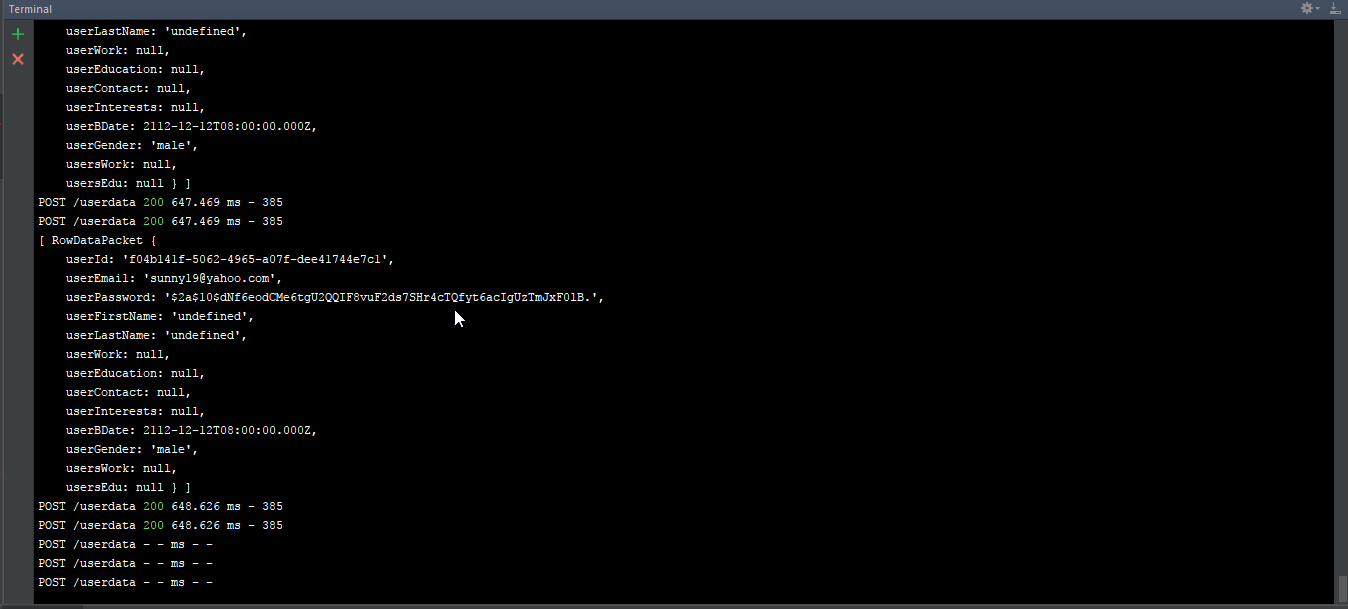




## **Graph showing average time for 100,200,300,400 and 500 concurrent users with and without connection pool.**



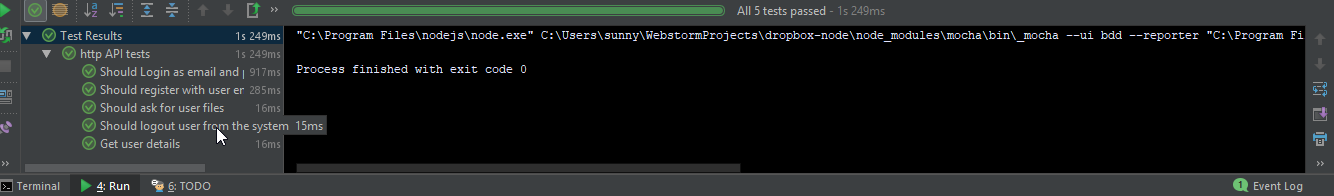
**Server:**

****

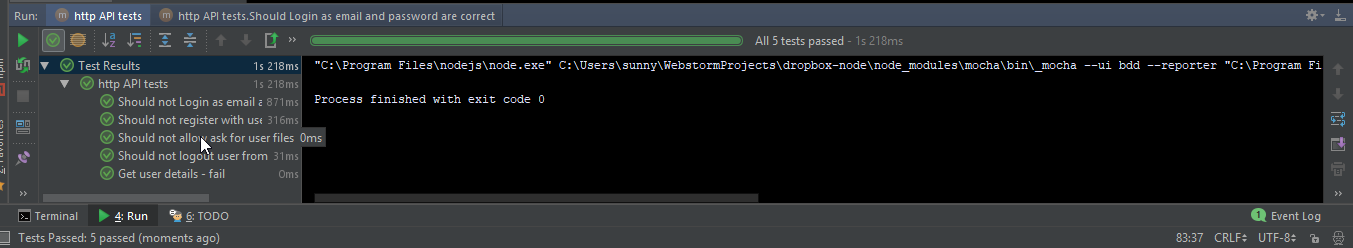
**MOCHA TESTING:**

I have tested 10 API calls using MOCHA.

**Positive Test cases:**



**Negative Test cases:**

****

# **PART: 3**

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

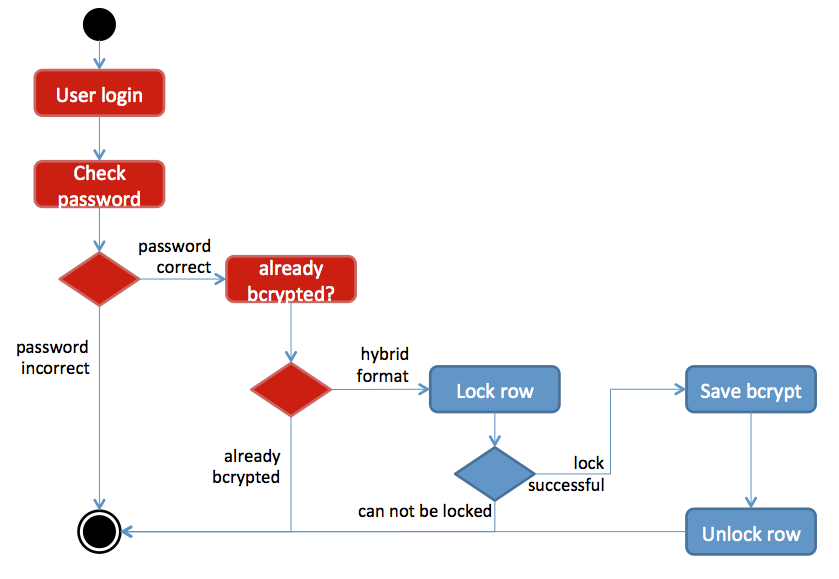
I have used nodejs library “bcrypt” which provides **bcrypt algorithm** for encrypting and validating user’s password. It is based on **blowfish cipher**. There are many other encrypting algorithms like: AES, SHA (1, 224, 256, 384, 512), RSA, TRIPLE DES, Twofish etc.

I selected bcrypt algorithm because of its expensive key setup phase. It starts off with a set of subkeys in standard state, which is used later to perform block encryption using part of key and use it to replace some of the subkeys. It repeats this process till it replaces all the subkeys Moreover, bcrypt is more resistant to brute force attacks because it is an adaptive function. Due to its adaptive nature, whenever a hacker tries to use brute-force attack to get into the system, overtime the iteration count keeps on increasing which makes it slower.

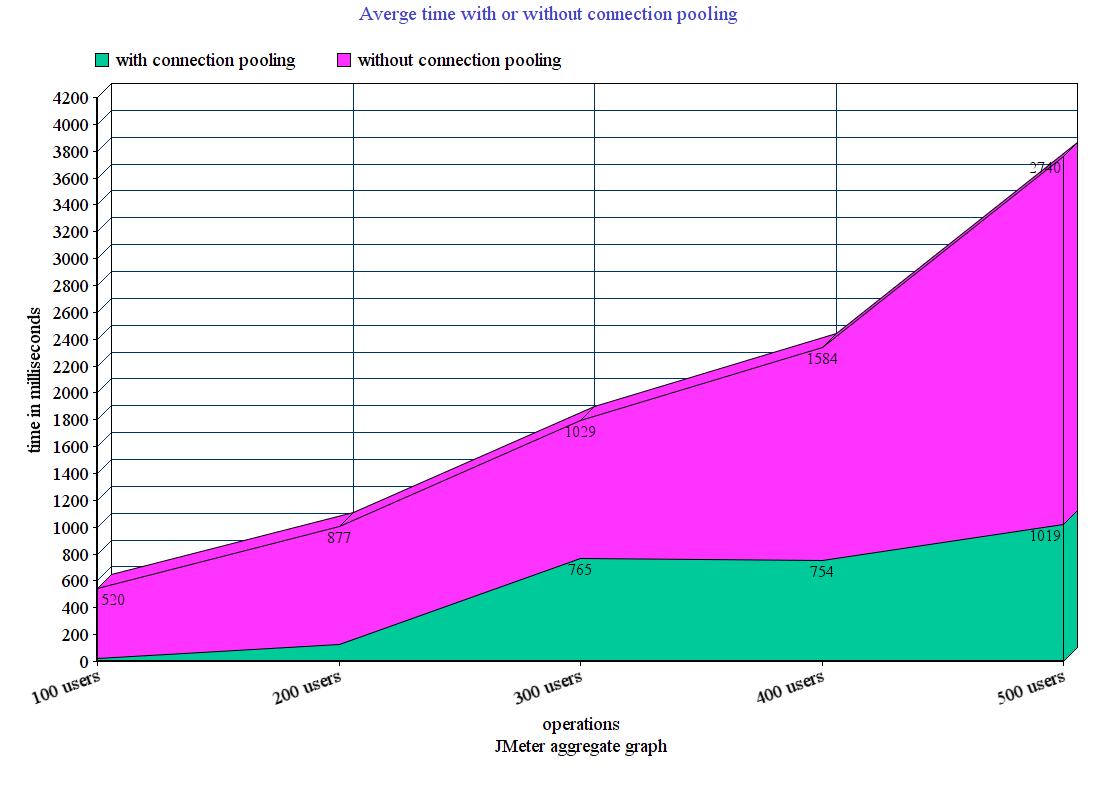
Bcrypt also adds random data that is used as an additional input. The random data added is called SALT. This also prevents any rainbow table attacks.

**Security Concern – Bcrypt algorithm only considers first 72 characters of a string and the rest are ignored.**

Bcrypt algorithm working:



1. *Compare the Results of the graphs with and without connection pooling. Explain the results in detail. Describe the algorithm of connection pooling used in your application.*



The graph above shows the comparison between the applications’ average response time when connection pooling is implemented and when it is not implemented.

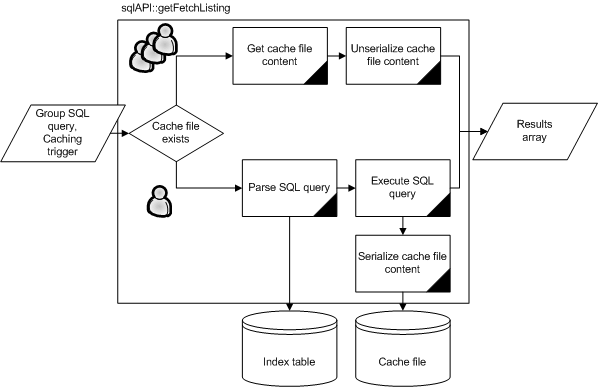
When there are 500 concurrent users each having one call, the average response time is 2740 millieseconds when no connection pooling is implemented, which reduces to 1019 millieseconds on implementing connection pooling. The reason for this significant drop in response time by 50% is that the connections are created when the application is started and are allocated from the pool to a request as and when a request comes, as a result the time of creating a new connection is reduced. The server picks up a connection from pool of connections and serves the request instead of wasting time in creating one.

In my application, initially 100 database connections will be created and added to the pool when the server starts where the maximum size of pool is 1000. When a user will access the server, he will get one connection from the pool (i.e. from array). When the connection ends, connection will be added back to the pool (i.e. it will be added back to the array).

var pool = mysql.createPool({  
 **connectionLimit** : 100,  
 **queueLimit** : 10,  
 **host** : **'localhost'**,  
 **user** : **'sunny'**,  
 **password** : **'sunny'**,  
 **database** : **'dropbox'**,  
 **port** : 3306  
});  
  
function *fetchData*(*callback*,*sqlQuery*){  
  
 ***console***.log(**"**\n**SQL Query : "**+*sqlQuery*);  
  
 pool.getConnection(function (*err* , *connection*) {  
 if(*err*){  
 ***console***.log(**"ERROR: "** + *err*.**message**);  
 *connection*.**release**();  
 }else {  
 *connection*.**query**(*sqlQuery*, function (*err*, *rows*, *fields*) {  
 if (*err*) {  
 ***console***.log(**"ERROR: "** + *err*.**message**);  
 }  
 else { // return err or result  
 *callback*(*err*, *rows*);  
 }  
 });  
 *connection*.**release**();  
 }  
 });  
}

1. *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.*

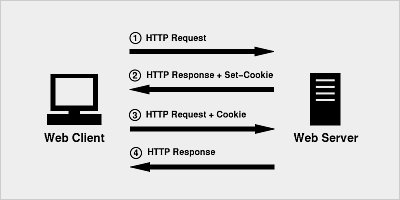
Caching in Databases:



As seen in the image above, the sql query execution plan or sql query is cached by the database system so as to decrease the amount of time it takes to execute a query. If a server sends the same query repeatedly to retrieve user details multiple times, the database will cache the execution plans for the query and when the query comes again the database doesn’t need to evaluate different plans again for executing the query and will immediately use the cache to respond to the execution request.

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

Session cookies

**

Yes, my session strategy is horizontally scalable. As I have used client sessions for session management, I save the session in cookies and the cookies are stored at client side. Alongwith a secret, when the cookie is passed by client in any request, the session variables are validated and the user session continues. When more requests are made by new clients, sessions are created and stored as cookies in their machine and when the subsequent requests are made along with cookie, after validating cookie data the user is authenticated. So no dependency on database is created and the time in validating data from database is saved. Even if the users delete the cookies from their local machine, in subsequent requests user is authenticated again and new cookies are created.