**LAB-1 Report**

Table of Contents

[1. Calculator 2](#_Toc496001513)

[1.1. Introduction 2](#_Toc496001514)

[1.2. System Design 2](#_Toc496001515)

[1.3. Results 2](#_Toc496001516)

[1.4. Performance 6](#_Toc496001517)

[2. Dropbox 12](#_Toc496001518)

[2.1. Introduction 12](#_Toc496001519)

[2.2. System Design 13](#_Toc496001520)

[2.3. Results 14](#_Toc496001521)

[2.4. Performance 19](#_Toc496001522)

[3. Answers 32](#_Toc496001523)

# Calculator

## Introduction

Goal:

Designing Calculator Web API to demonstrate client server operation using react.js and node.js where front-end is developed using react.js and back-end with node.js.

Purpose:

Purpose of this system is to demonstrate stateless web service with react.js and node.js by building Calculator API with following operations like:

1. Addition
2. Subtraction
3. Multiplication
4. Division

With the operation exception handling is included (e.g. divide by 0) so that system does not crash. Also, invalid input is handled at client side (e.g. character as input).

## System Design

Calculator API is created using following:

* React.js
* Node.js
* Express.js
* Bootstrap

Front-end is developed using React.js and Bootstrap for better look and feel.

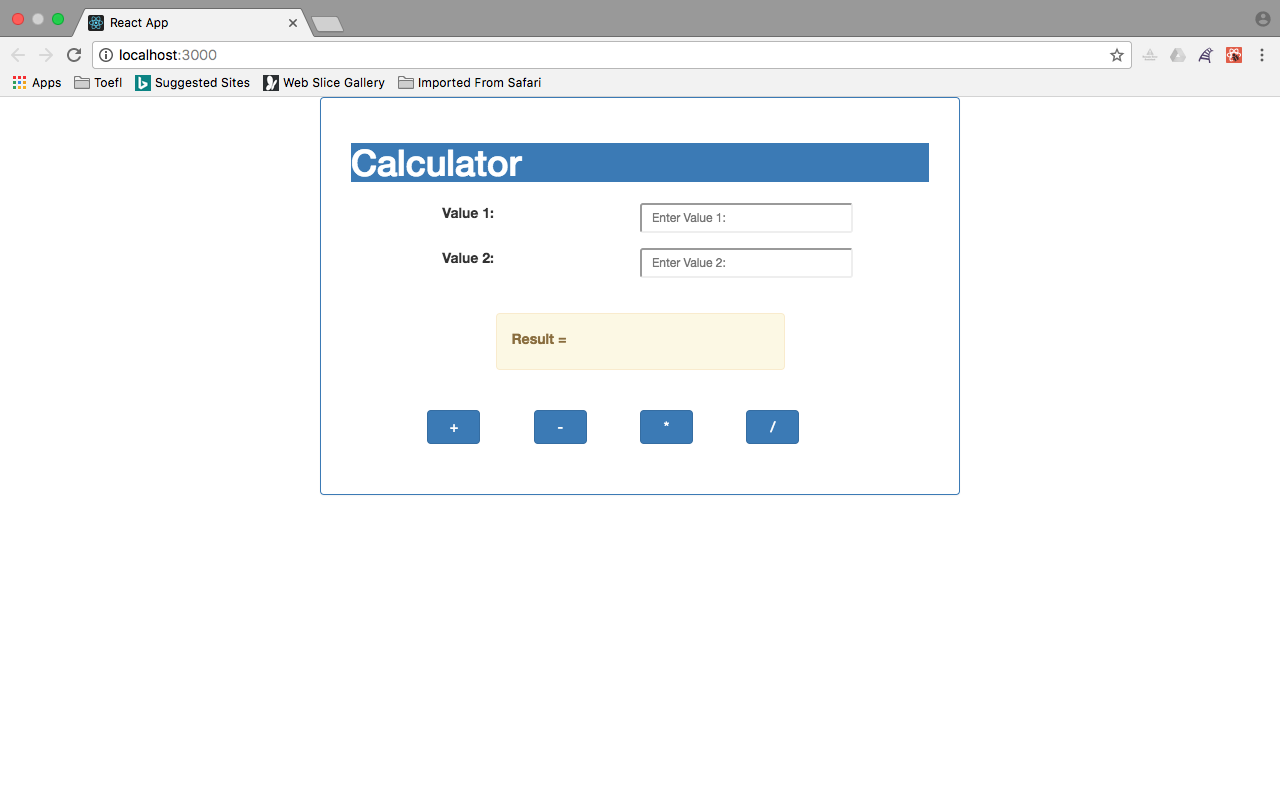
Two textboxes are given on the front-end. User is supposed to enter numeric values here. In case user gives input non-numeric or empty then error message will be sent.

For addition, subtraction, multiplication and division respective signed buttons are given on UI. On click on these buttons, respective API call is sent to server side and server handles the request.

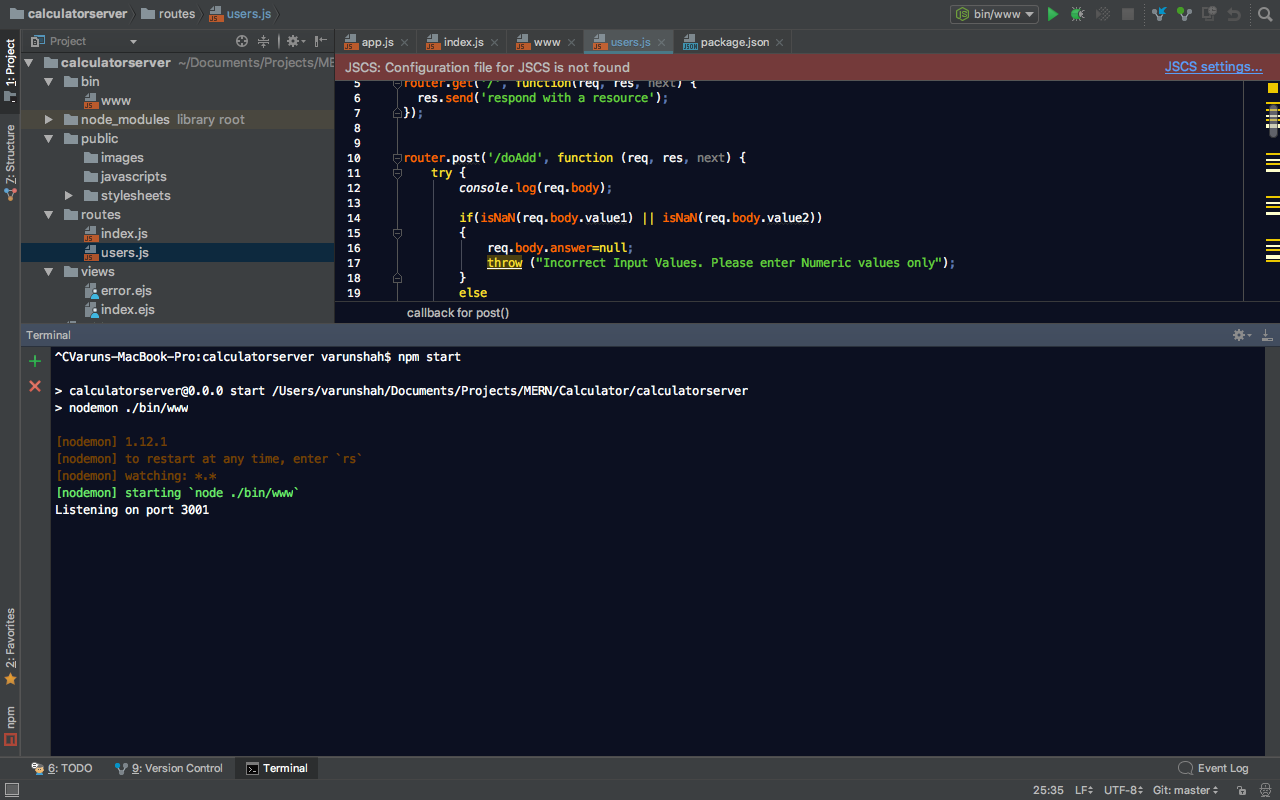
Node.js handles request sent by client on back-end. Received body from the client is verified on server side whether given values are appropriate to perform computation, only then computation is performed and answer will be sent to client in JSON form. Client verifies the response status, if status is successful then it fetches answer from the JSON object and updates it into state. Due to which Result gets updated on UI.

## Results

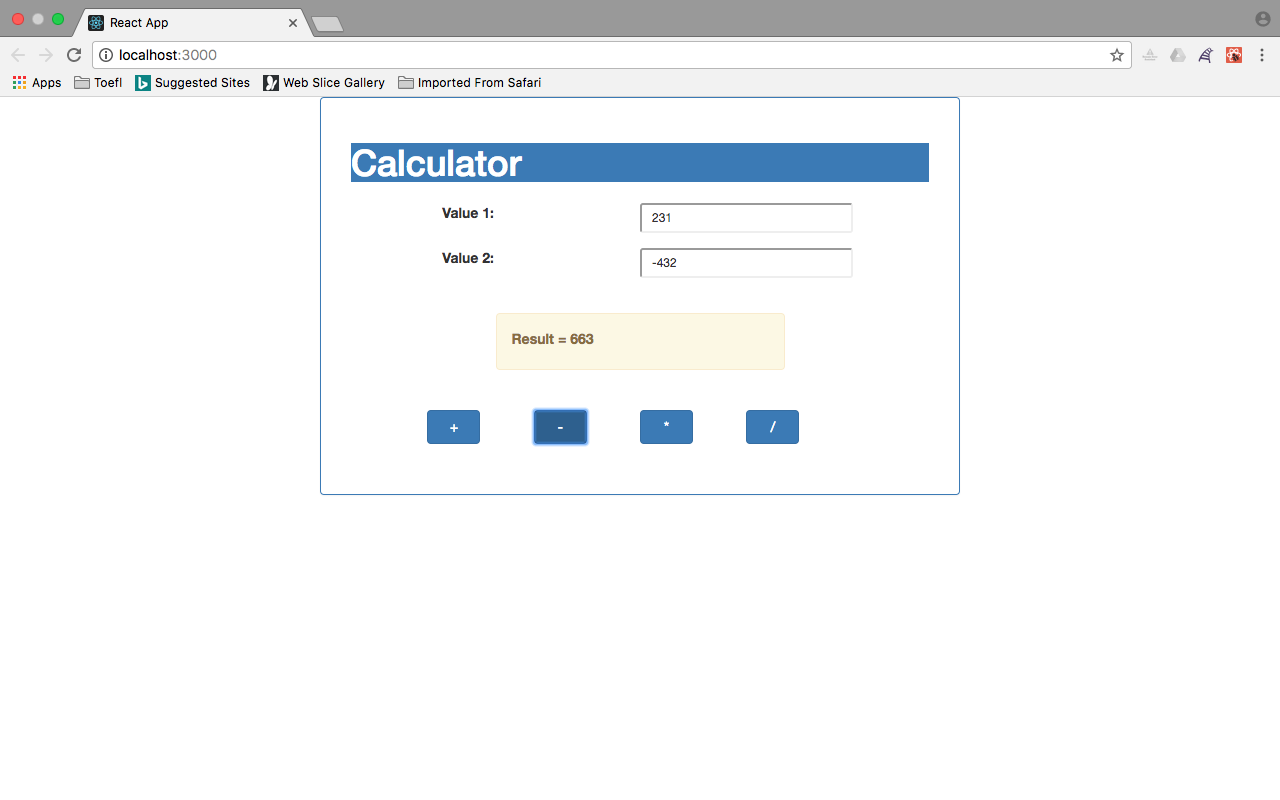
* On Client Load:



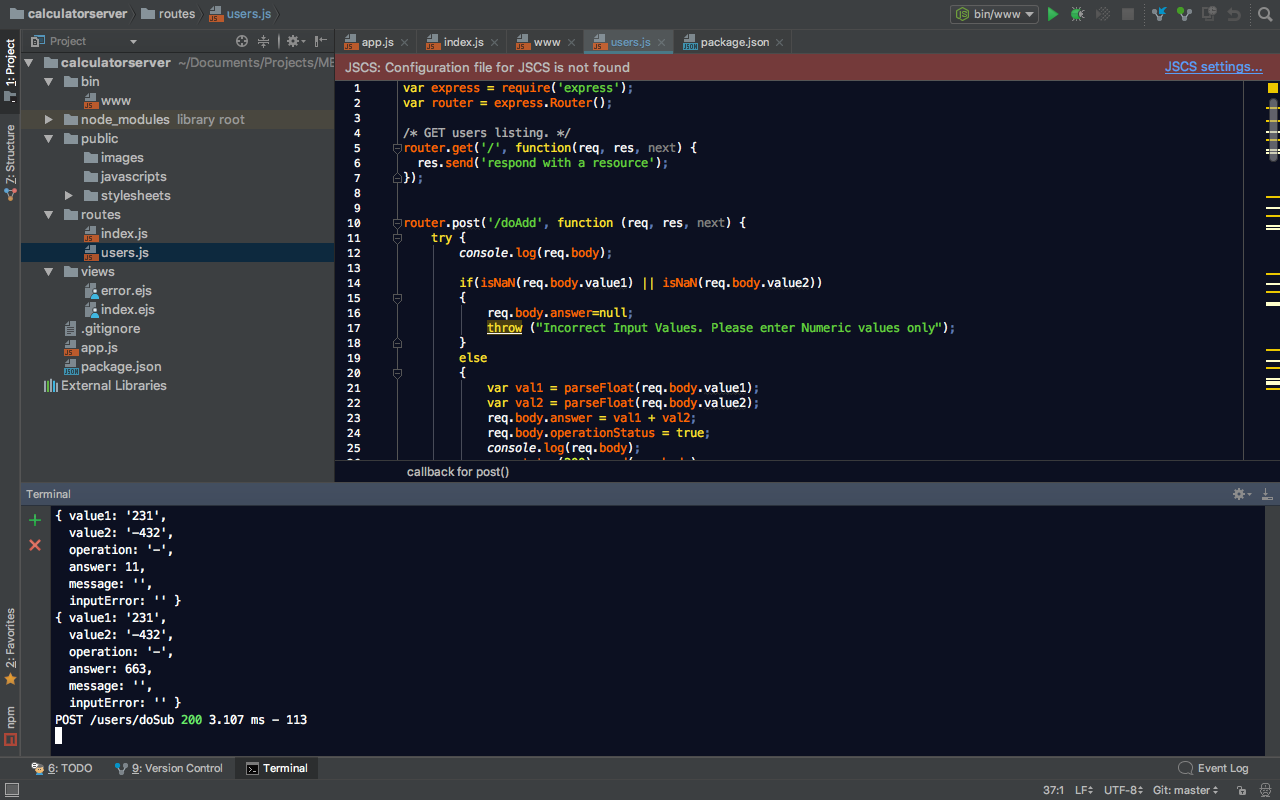
* Server side on Client load



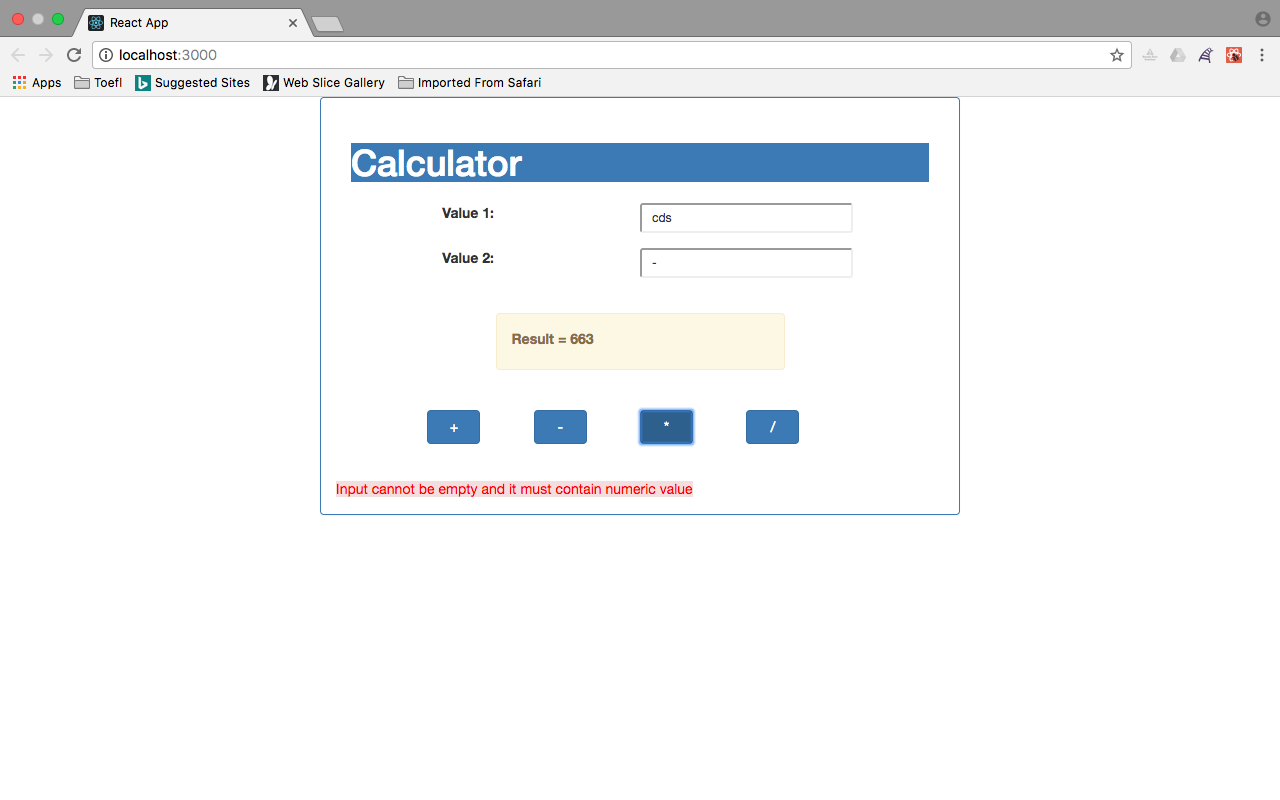
* Setting input and performing operation. Client

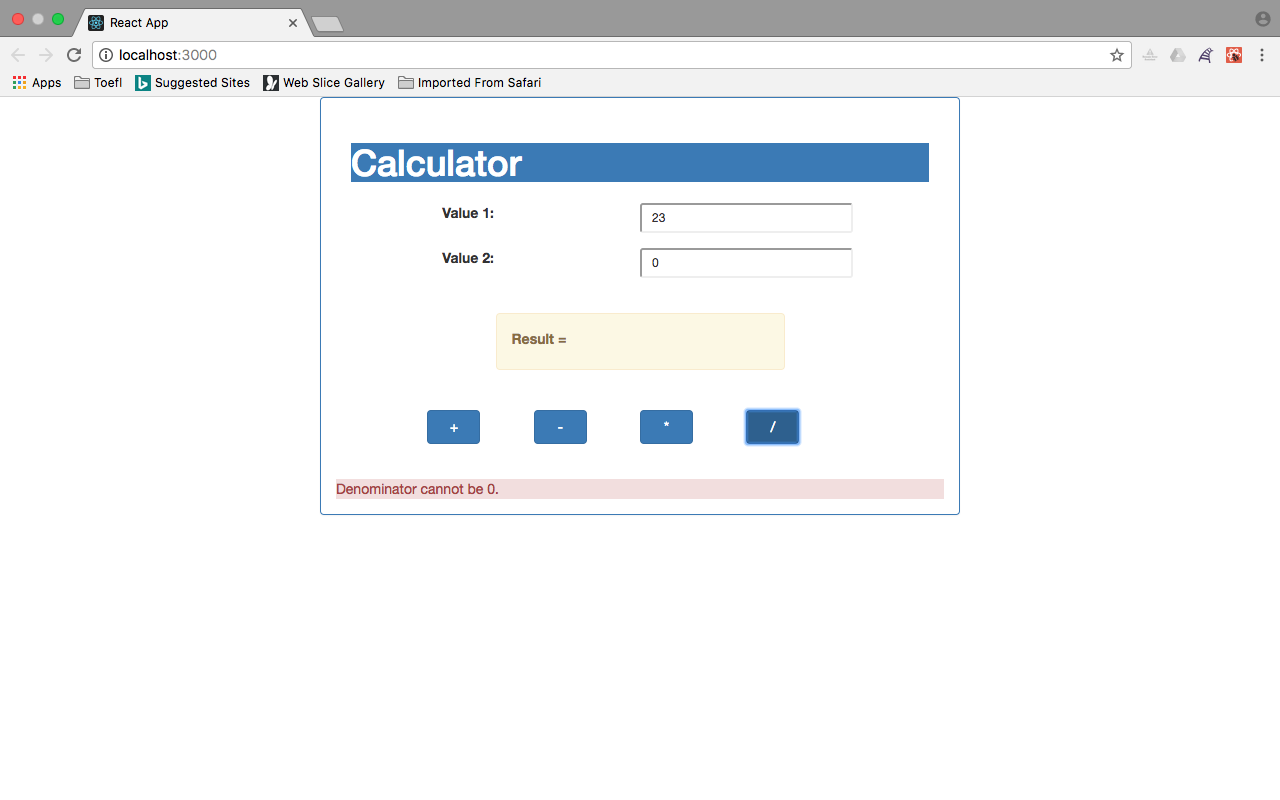


* Setting input and performing operation. Server

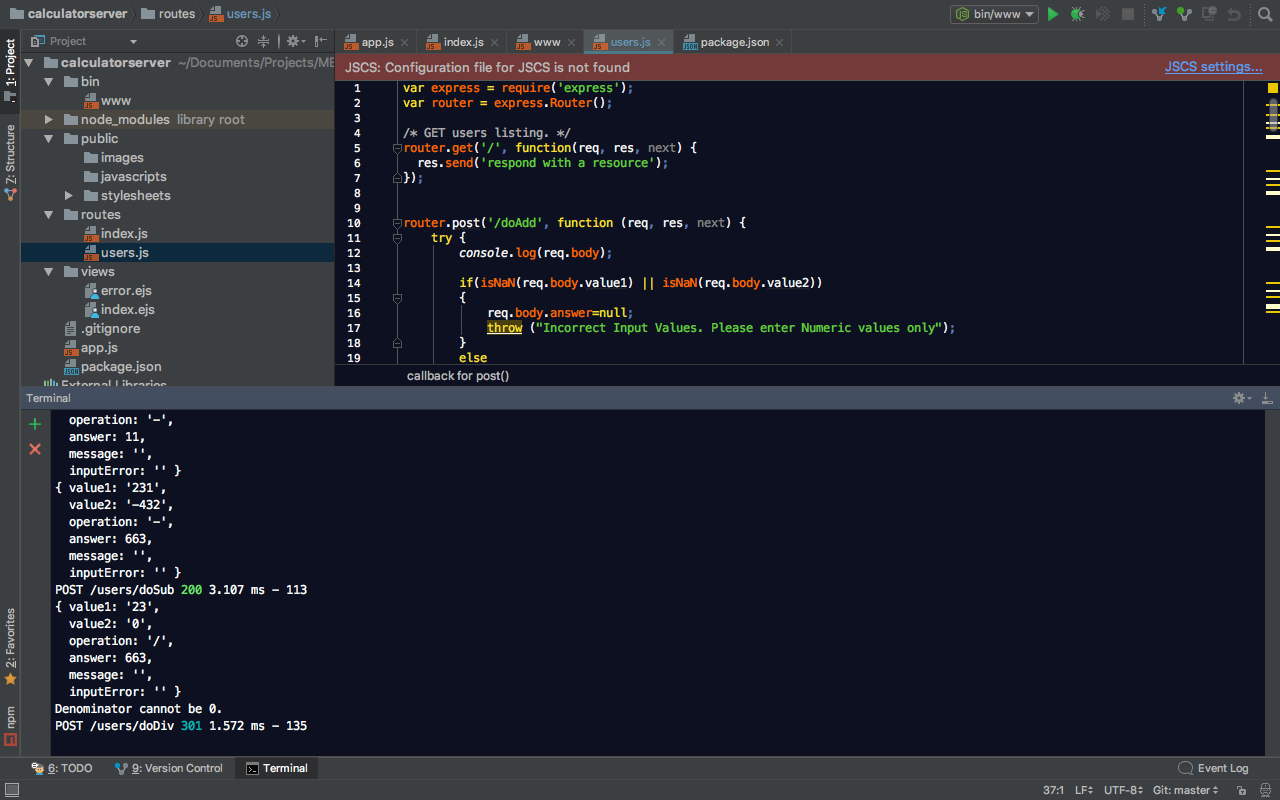


* Invalid Input. Client





* Divide by 0. Server Exception Handling

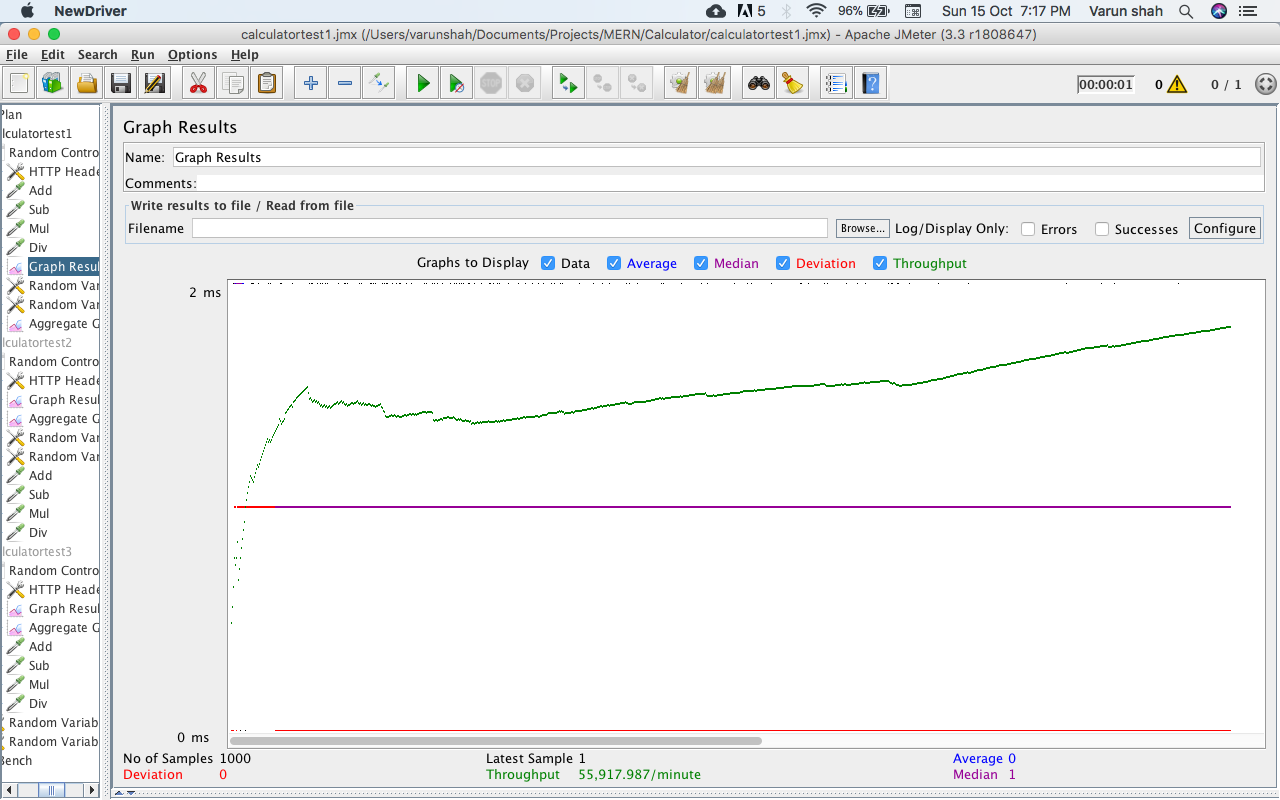


## Performance

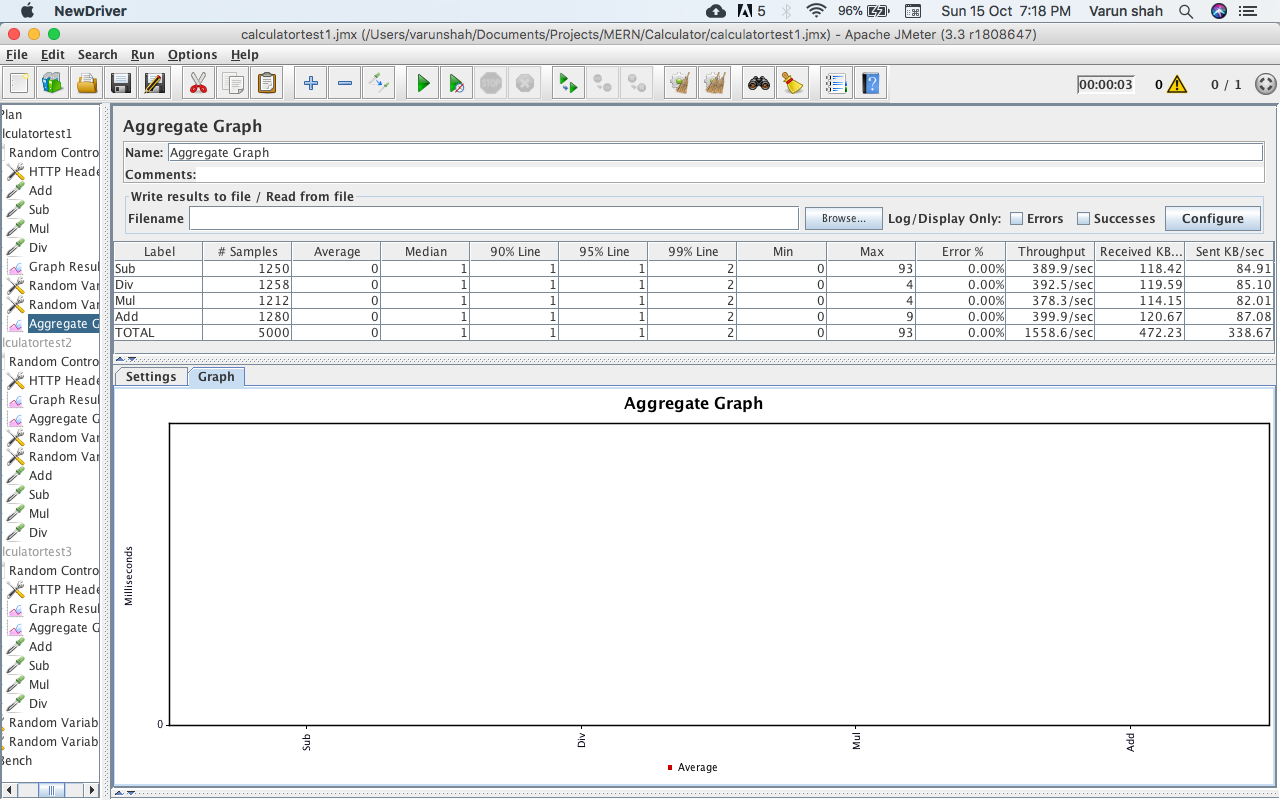
* Invoke 1,000 calculator calls on randomly selected tasks and print average time to perform each operation.
* Aggregate Graph



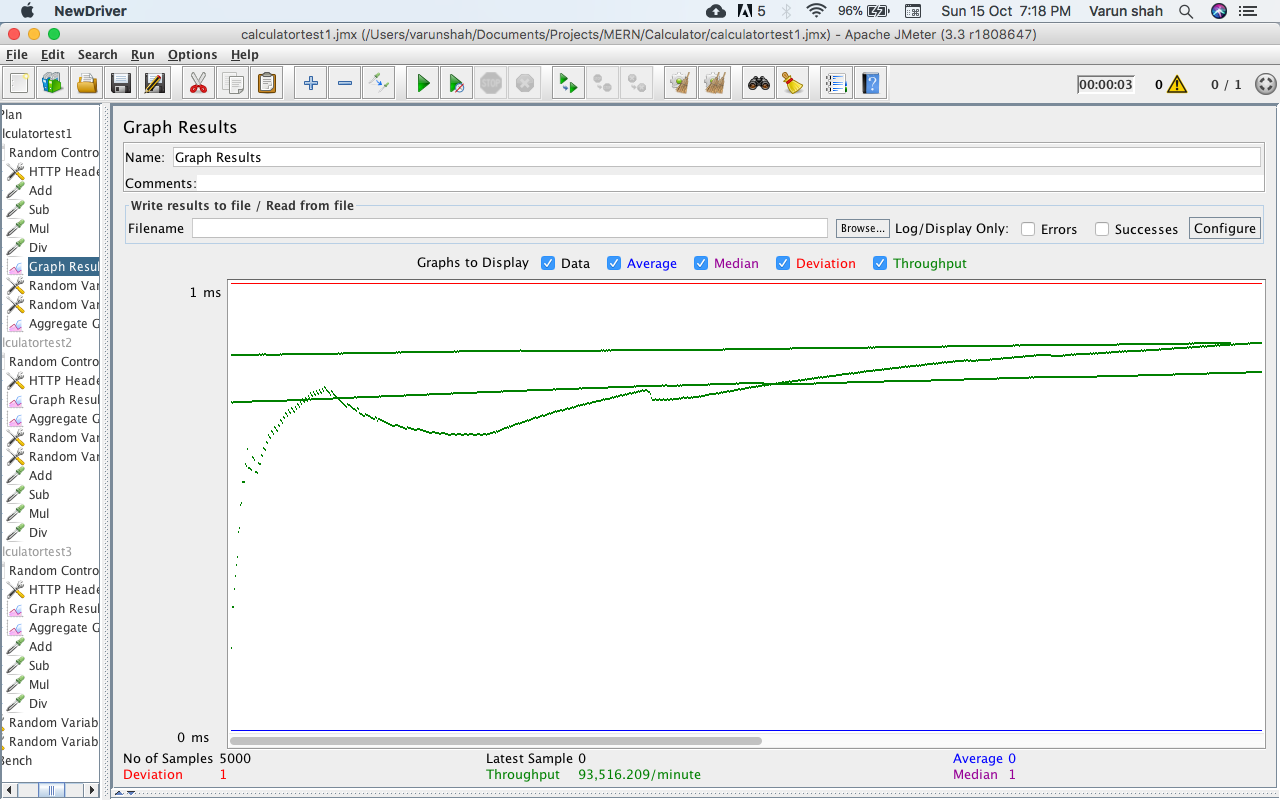
* Graph Result



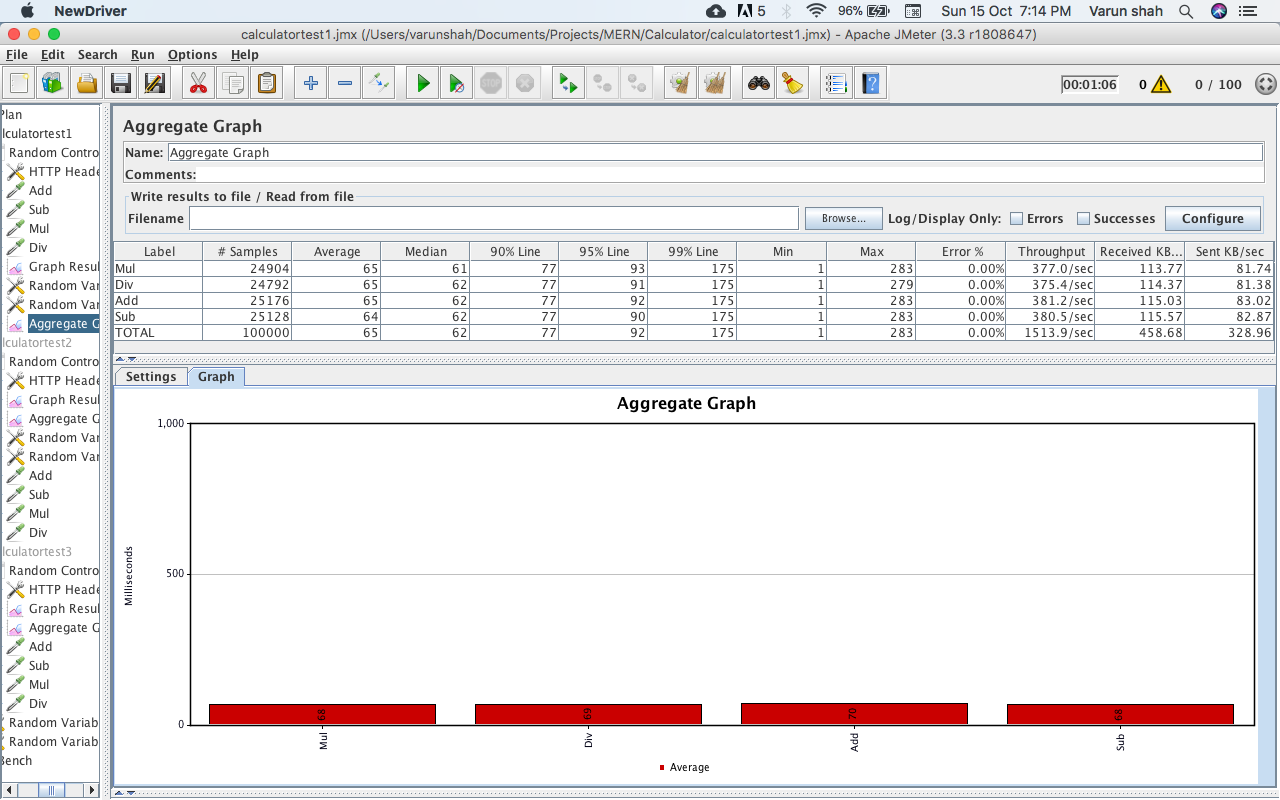
* Invoke 5,000 calculator calls on randomly selected tasks and print average time to perform each operation.
* Aggregate Graph



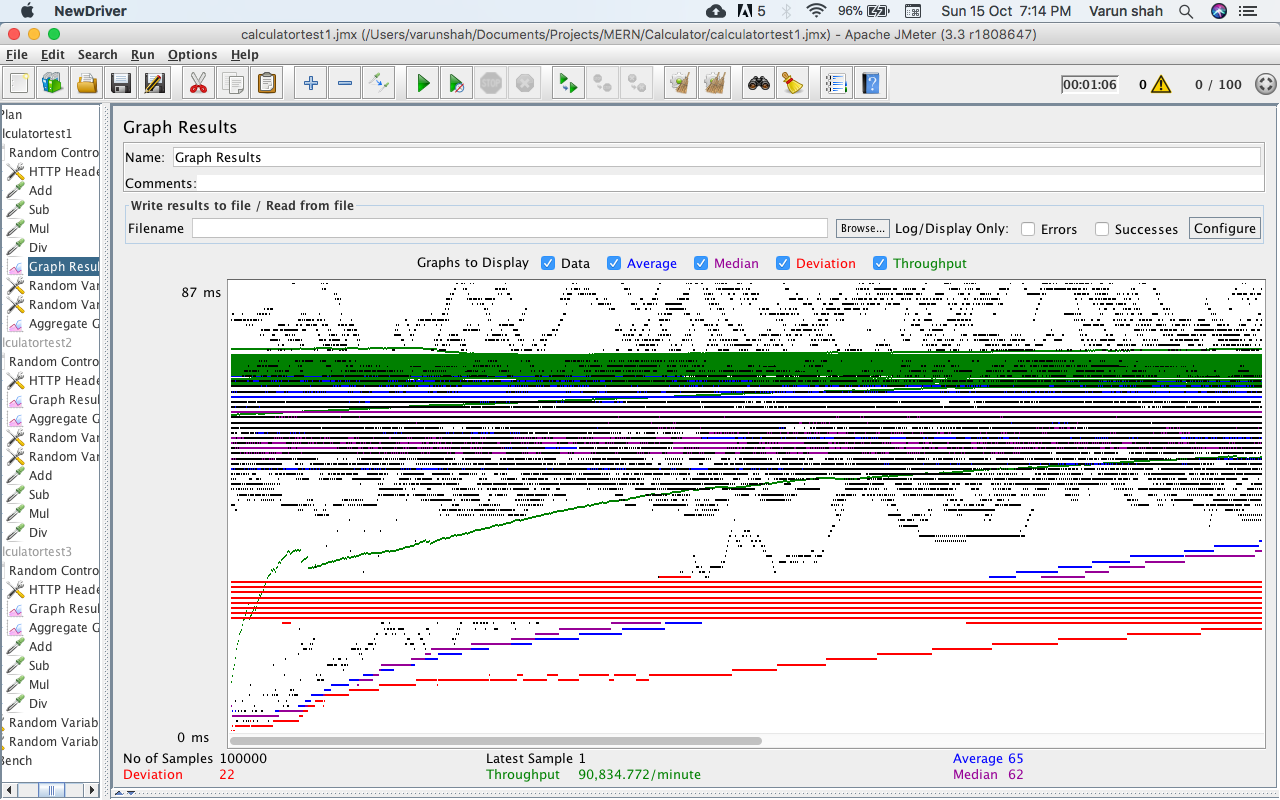
* Graph Result



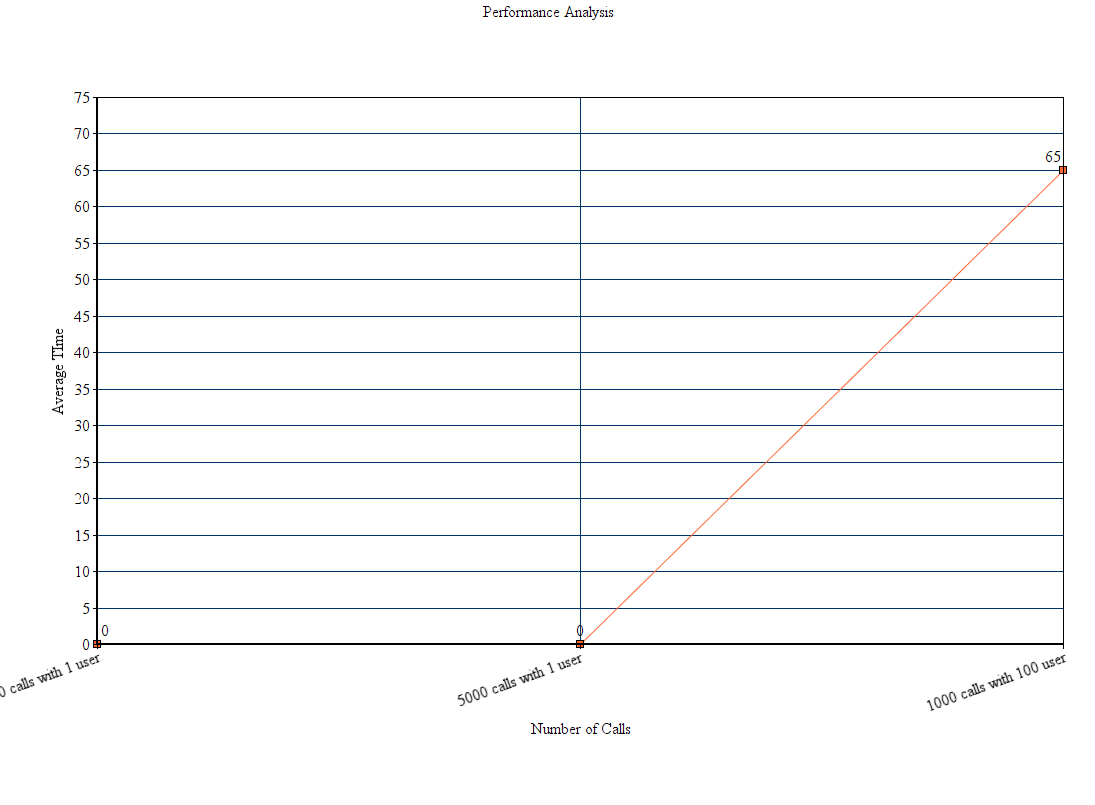
* Invoke 100 concurrent users with 1000 calls each to calculator on randomly selected tasks and print average time to perform each operation.
* Aggregate Graph



* Graph Result



* Analysis



Creating Total number of calls to Average Time graph for all three above mentioned scenarios above line chart generates. As it can be seen, average time taken for 1000 from 1 user is near to 0 milliseconds. Same way average time taken for 5000 server calls from 1 user takes near to 0 milliseconds. When 1000 server calls were sent the average time taken by the server to address all the request is 65 milliseconds.

It can be concluded that when using one server to handle multiple users’ requests, it takes a time to address them. So, having multiple servers with distributed system and it can be handled proficiently.

# Dropbox

Dropbox Web Application replica of Dropbox to demonstrate Rest Web Services

## Introduction

Goal:

Designing Dropbox Web API to demonstrate REST web services using react.js as front-end and node.js and express.js as back-end.

Purpose:

Purpose of this system is to demonstrate stateless REST web service with react.js and node.js by building Dropbox API with following operations:

* Basic Users functionalities
  + Sign up new user (first, name, last name, Email and password)
  + Sign in existing user (Encrypt Passwords))
  + Sign out.
  + Upload files
  + List a file
  + Create a directory
  + Star a folder/directory
  + Share a folder/directory by email/name/link
* Users account should provide basic details such as:
  + About: User overview, Work and education, contact info and life events
  + Interests like music, shows and sports
* Provide file list and activity report functionality

## System Design

Calculator API is created using following:

* React.js
* Node.js
* Express.js
* Bootstrap
* MySQL

Front-end is developed using React.js and Bootstrap for better look and feel. Components are divided in following way:

React Component App

MainPage.js

Successful login

SignUp

Login

User

Home

File

Activity

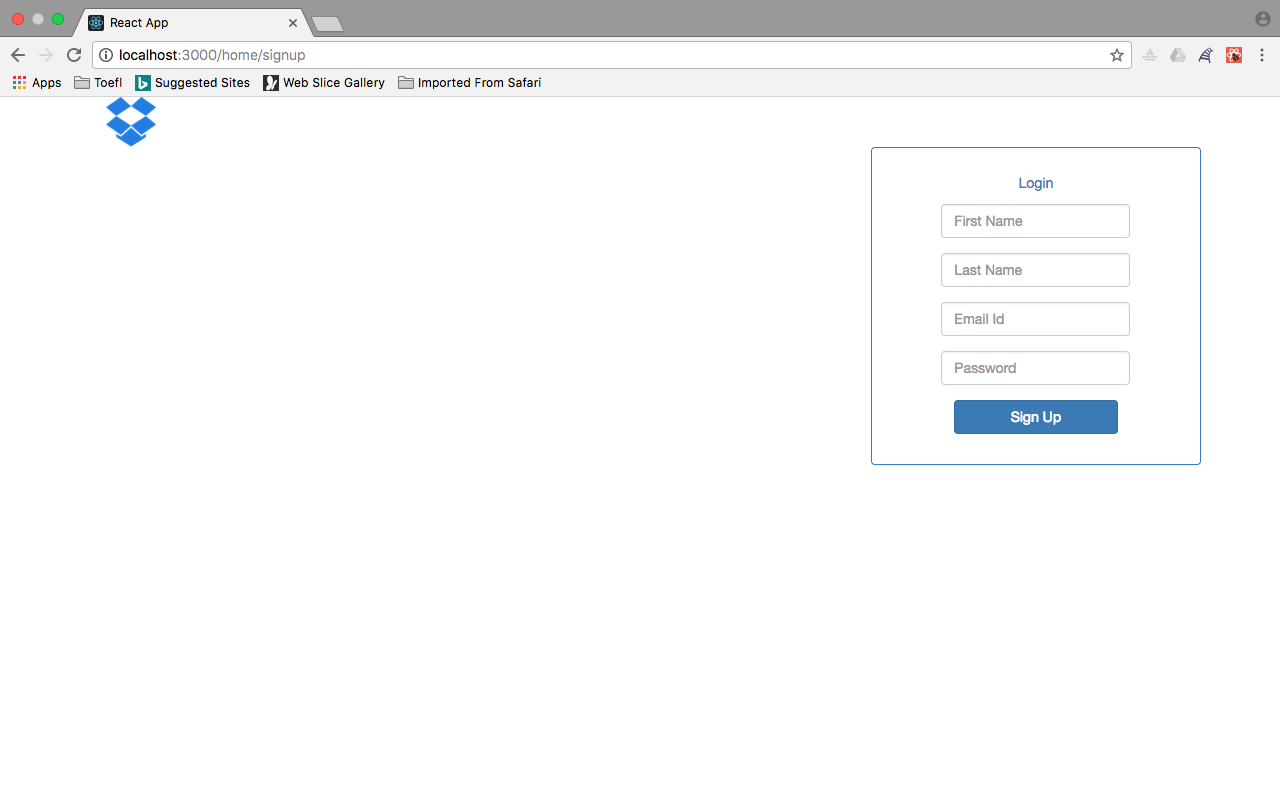
Profile

After logging in successfully user can access all the child components of User. When user logs in successfully session gets creation with user’s username and client server state will be managed by session. When session expires, user get redirected to login screen.

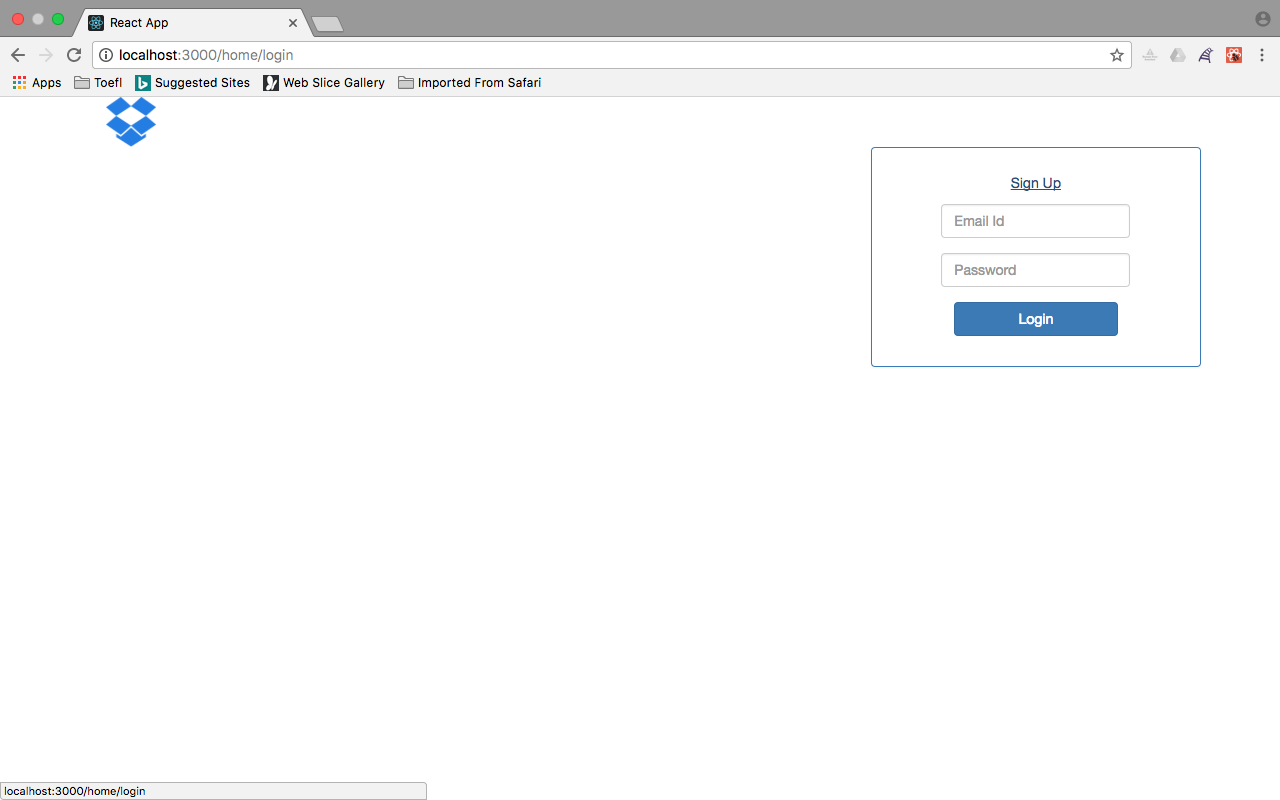
Back-end is developed using node.js and express.js with MySQL database. From server side, after successful login verification, users get access to their account. For each server request it is checked whether the respective user’s session is still active or not. If username is available in session then user will be able to access all post login features. If session expired, then server sends status that session expired so user will be logged out and redirected to login page.

## Results

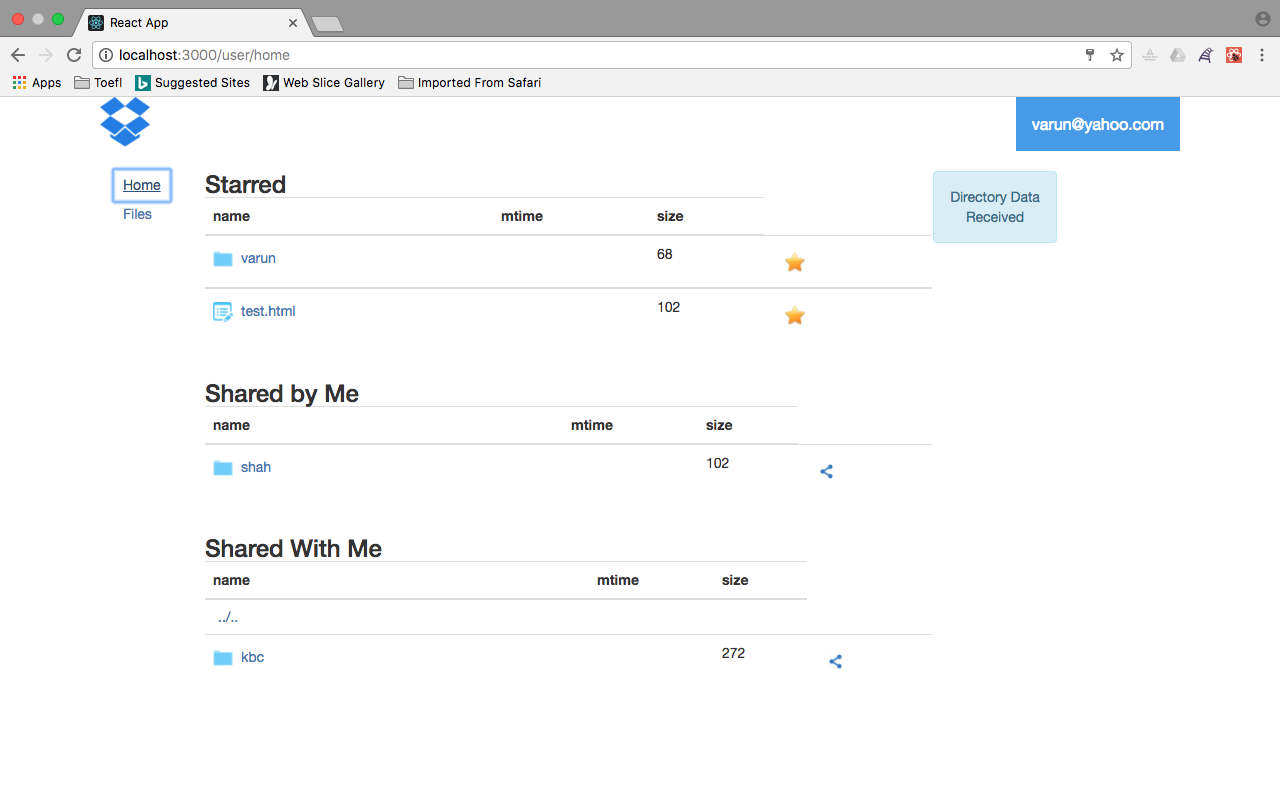
* Signup



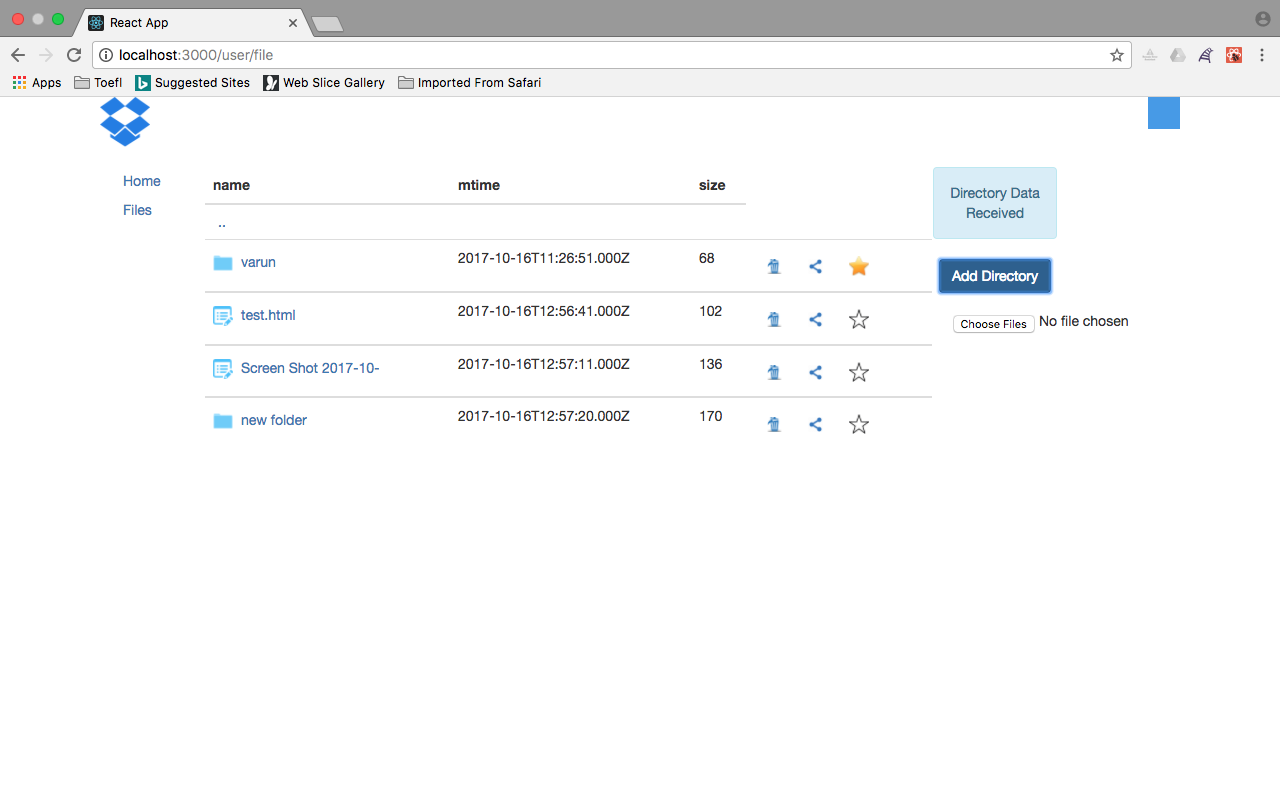
* Login



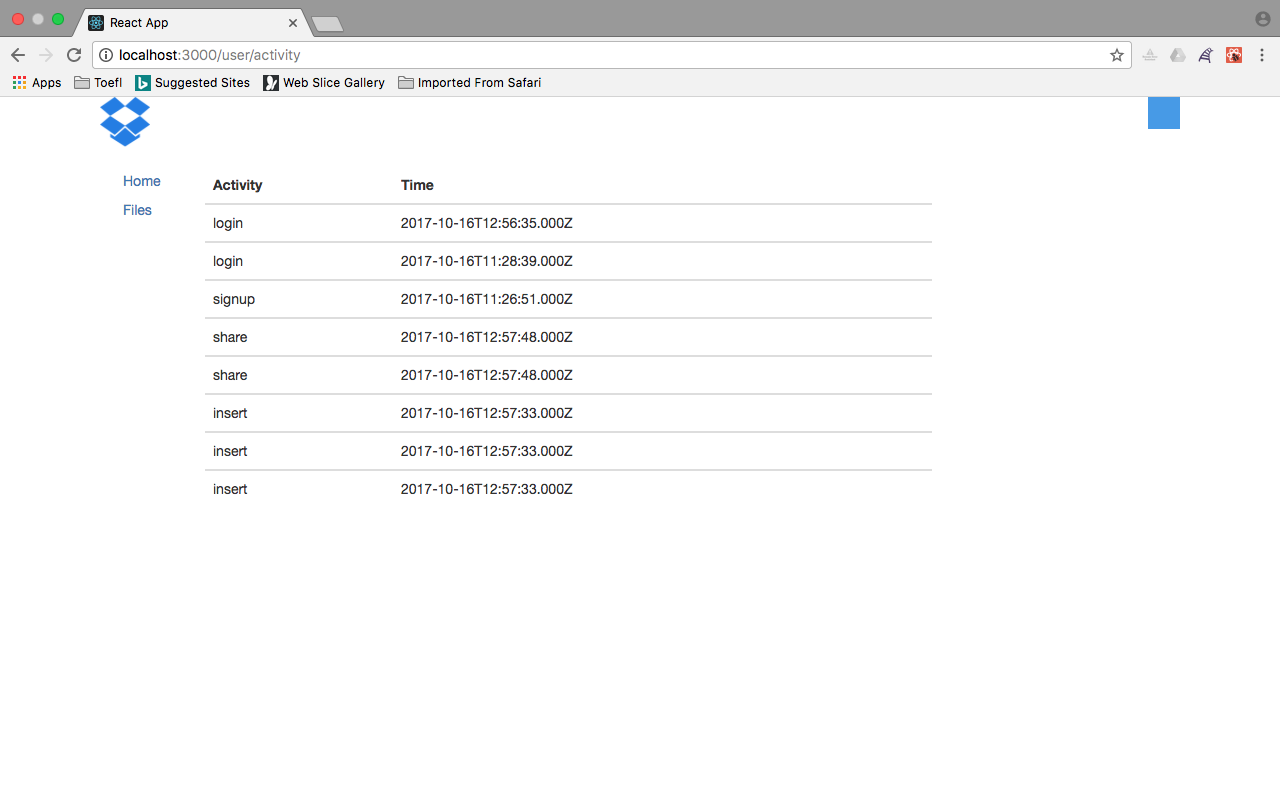
* After login User Home Screen with Starred data, data shared by user and data shared with user



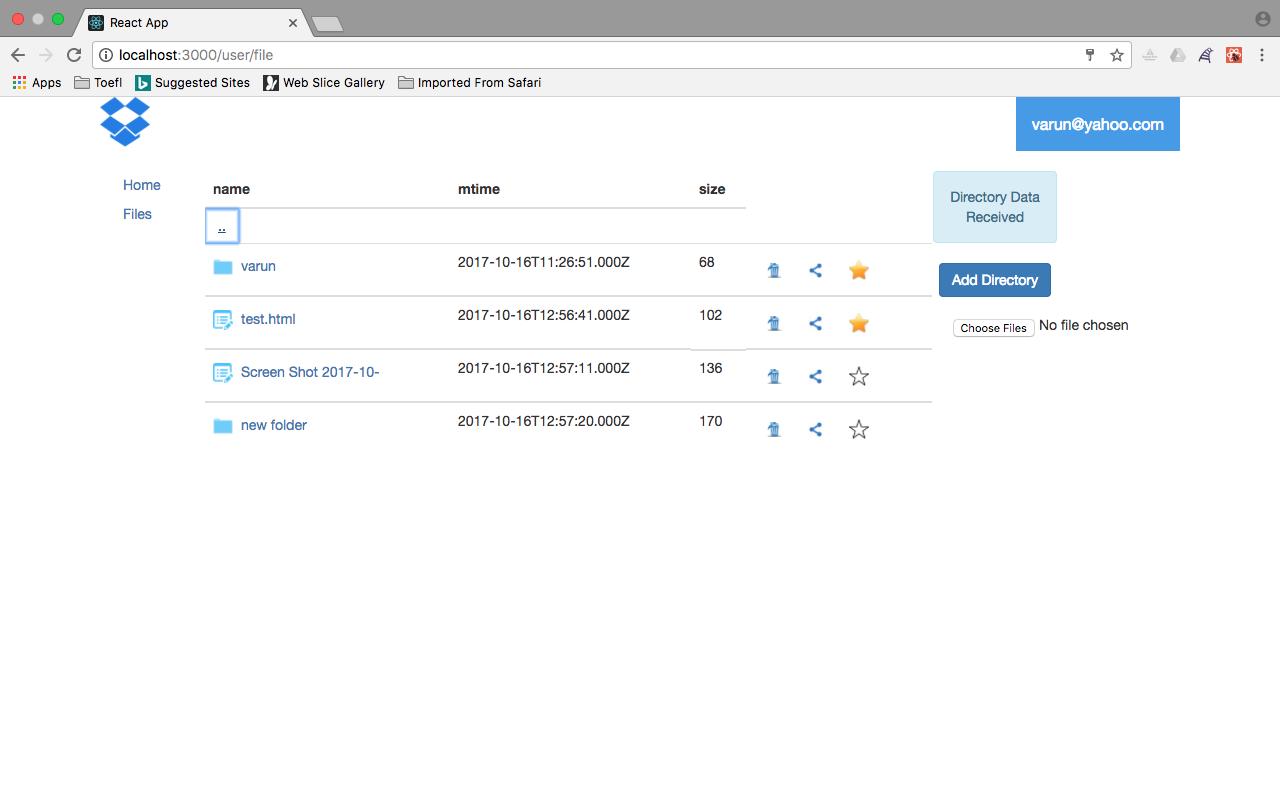
* User’s uploaded Filesystem

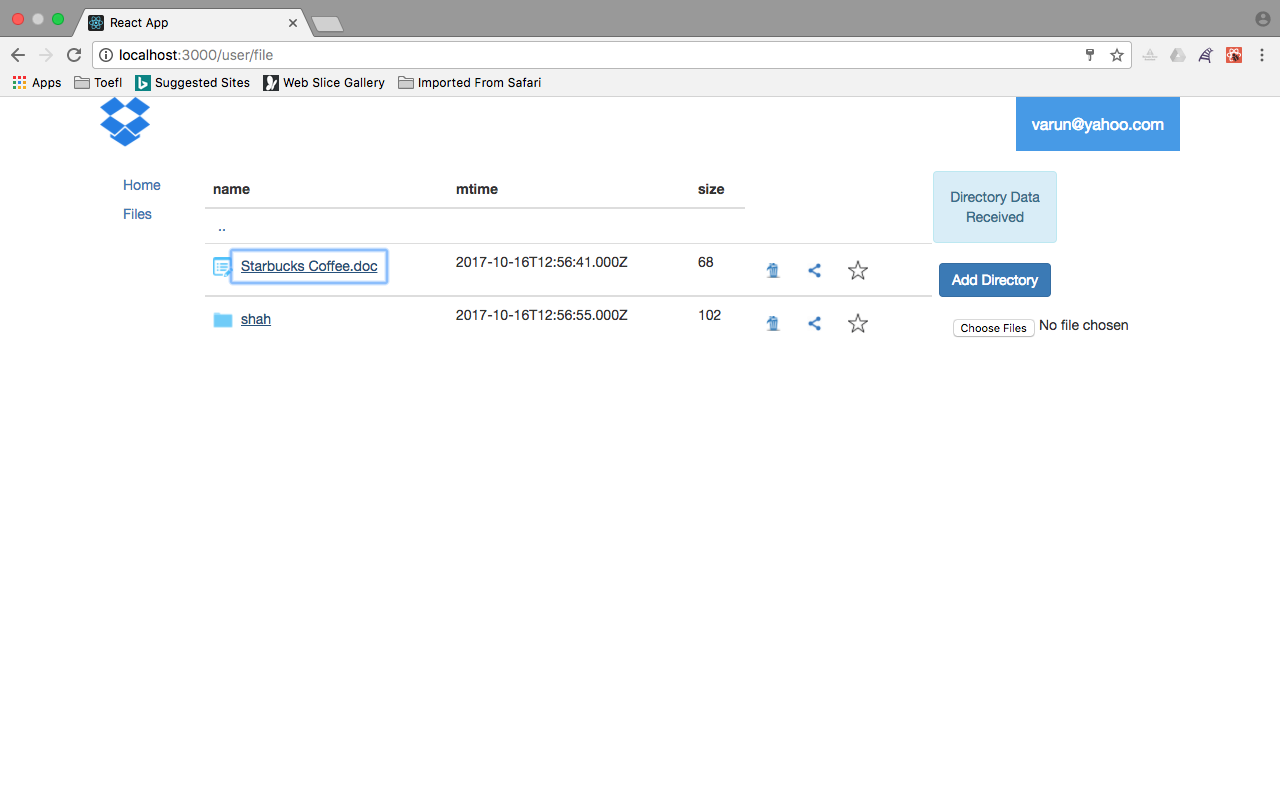


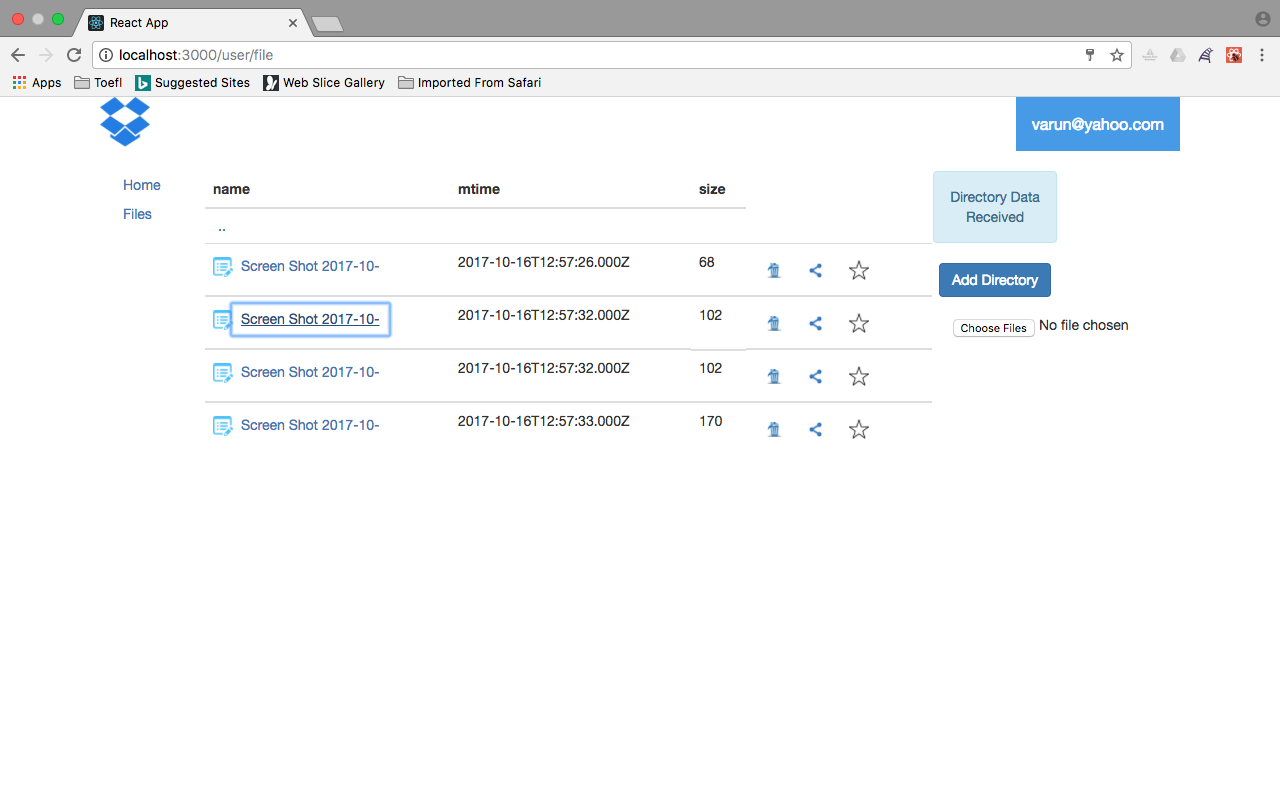
* Activity Screen



* Navigation in file system







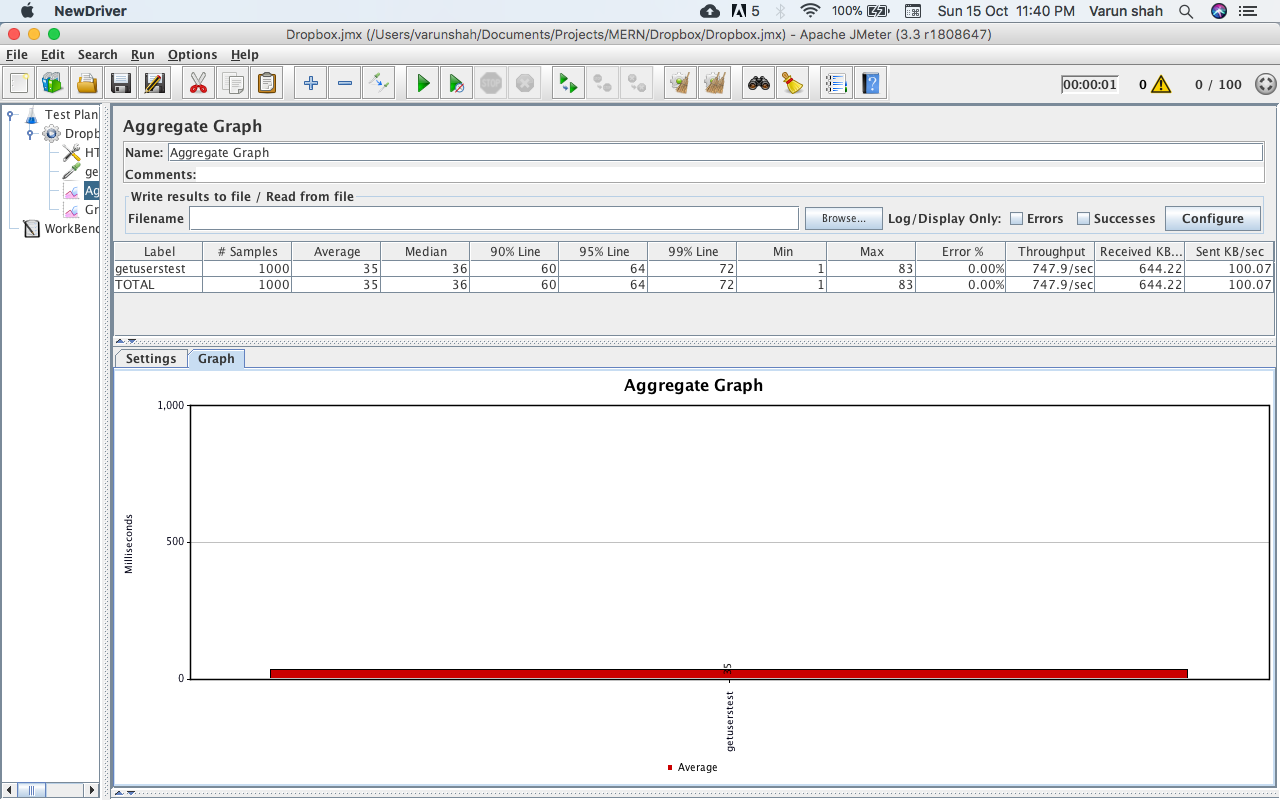
## Performance

* With connection pooling
  + 100 concurrent users

Graph Result:

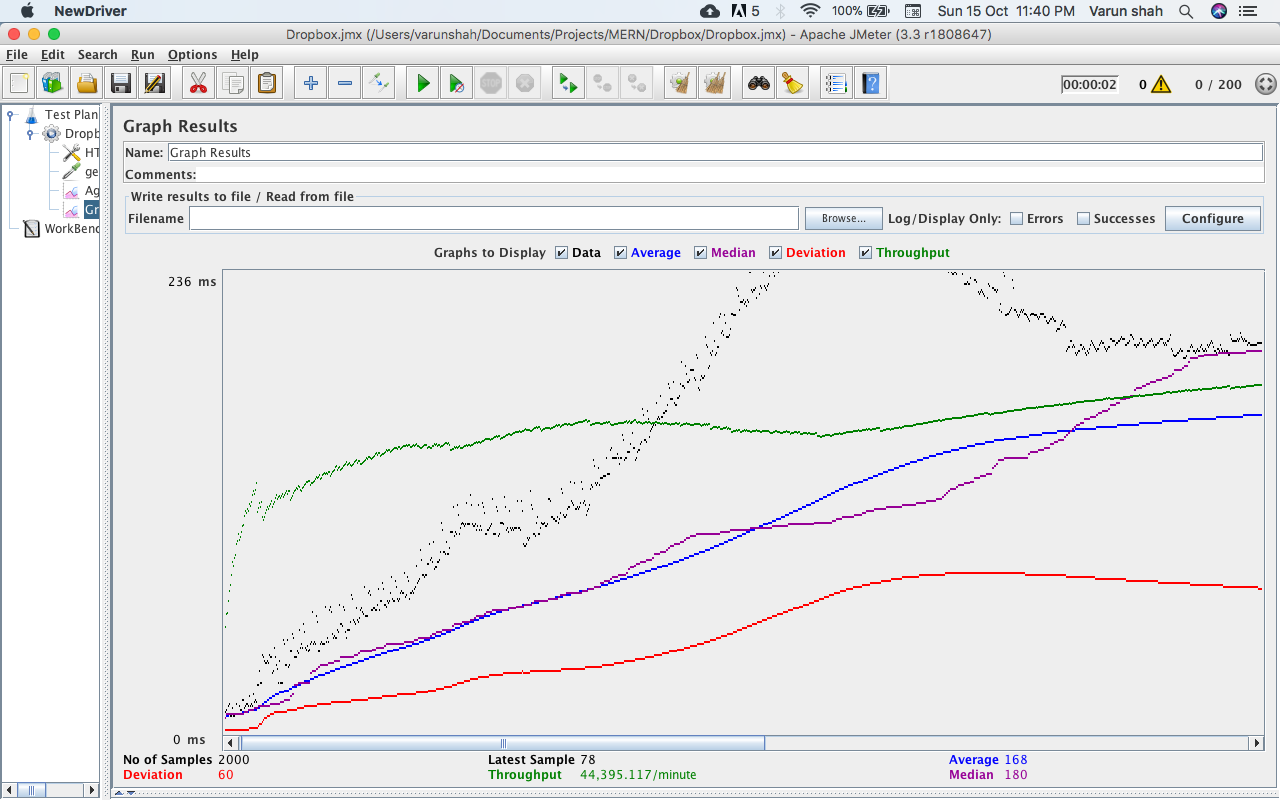


Aggregate Graph:

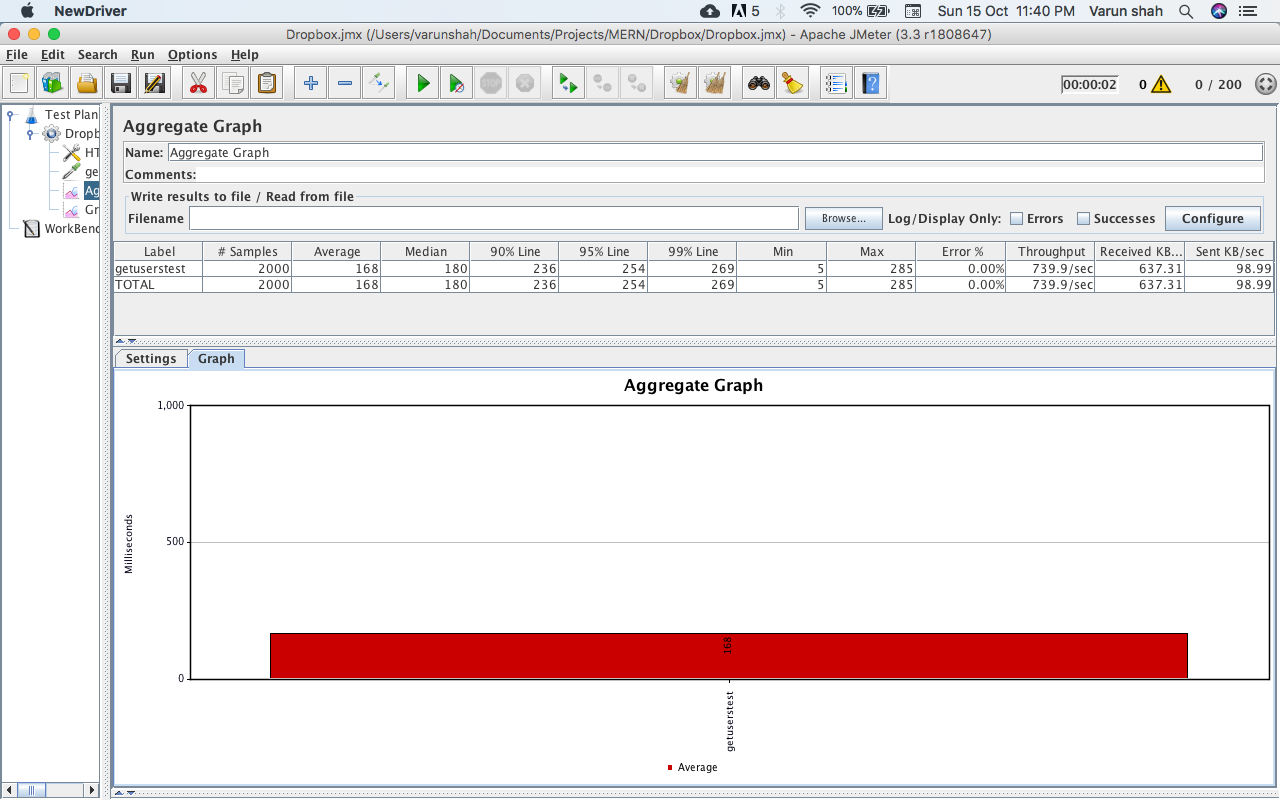


* + 200 concurrent users

Graph Result:

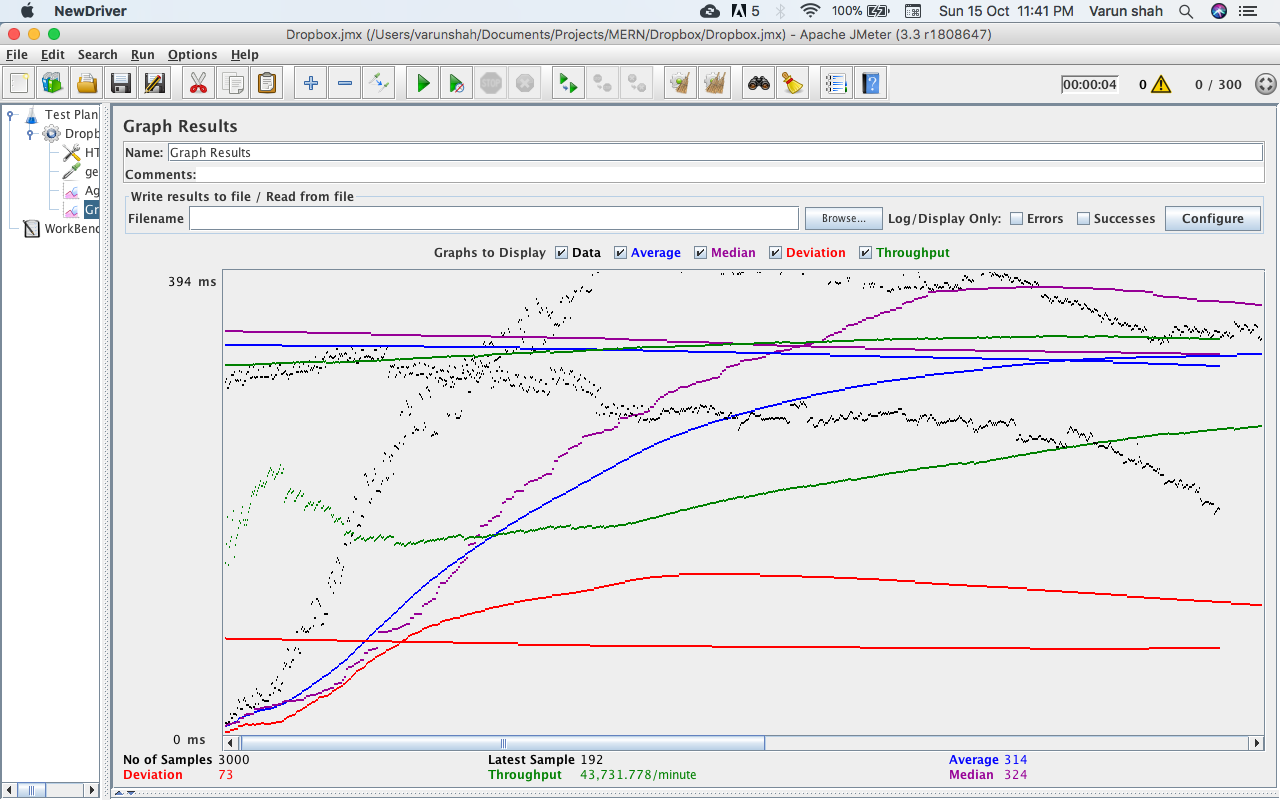


Aggregate Graph:

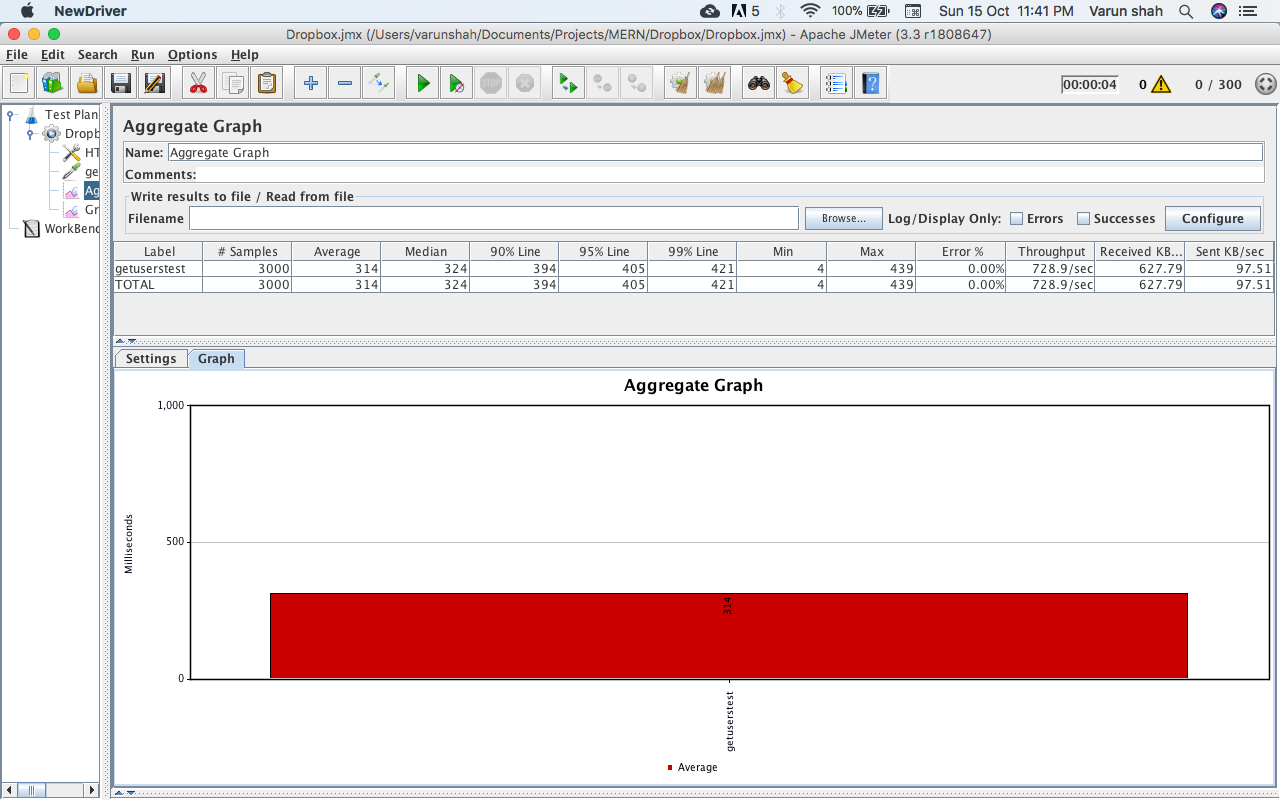


* + 300 concurrent users

Graph Result:

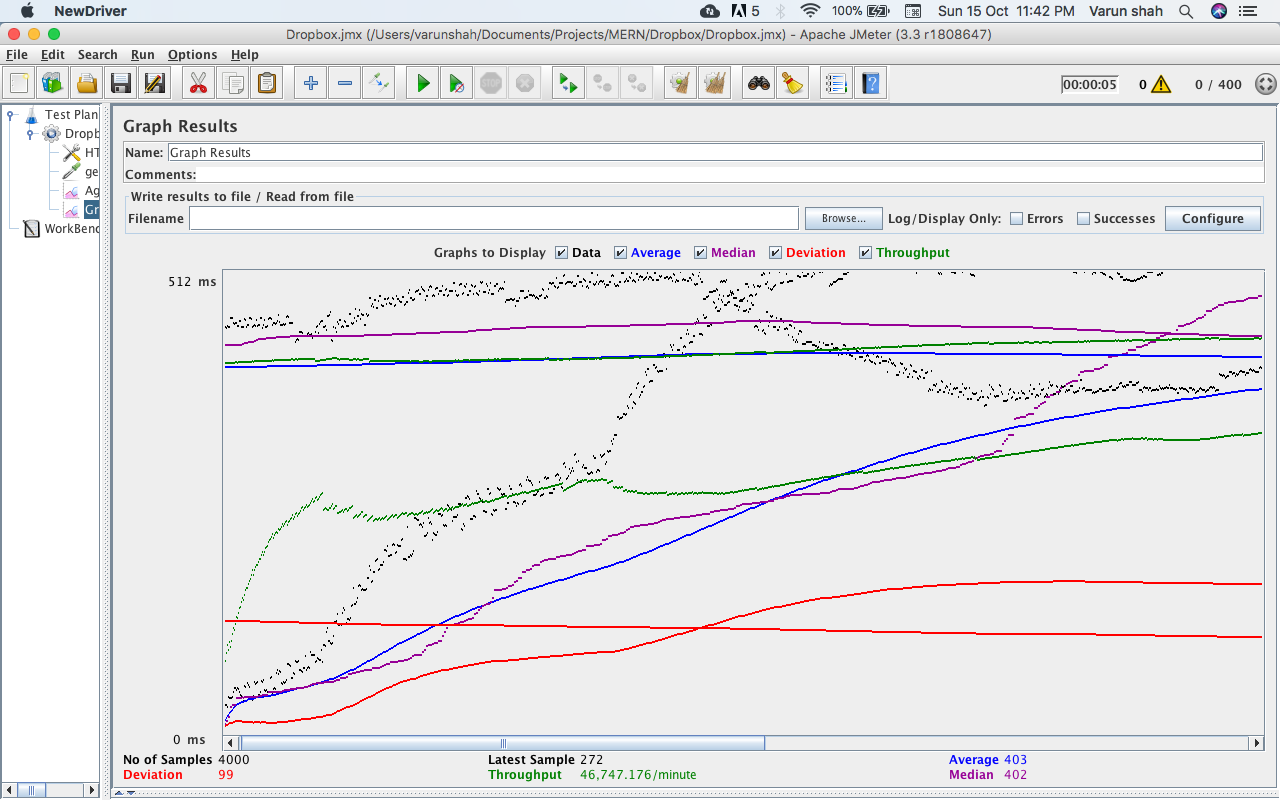


Aggregate Graph:

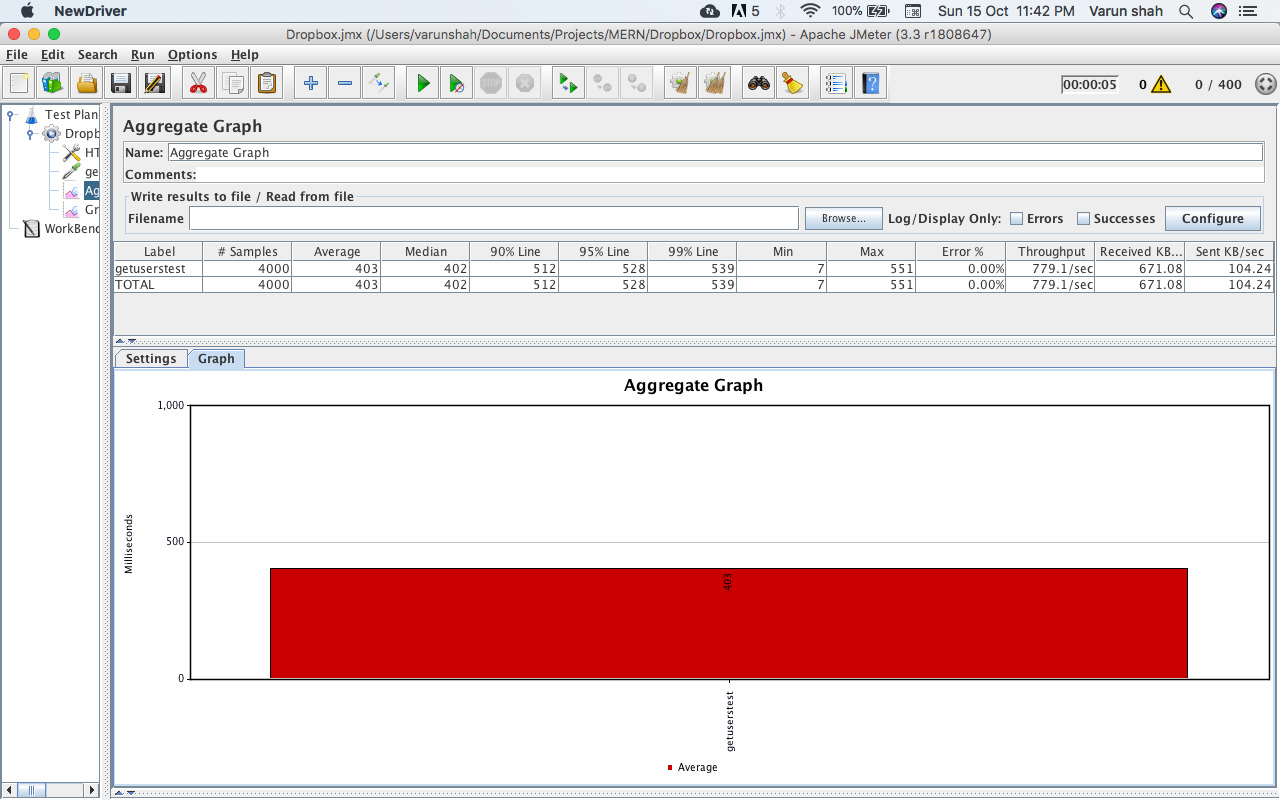


* + 400 concurrent users

Graph Result:

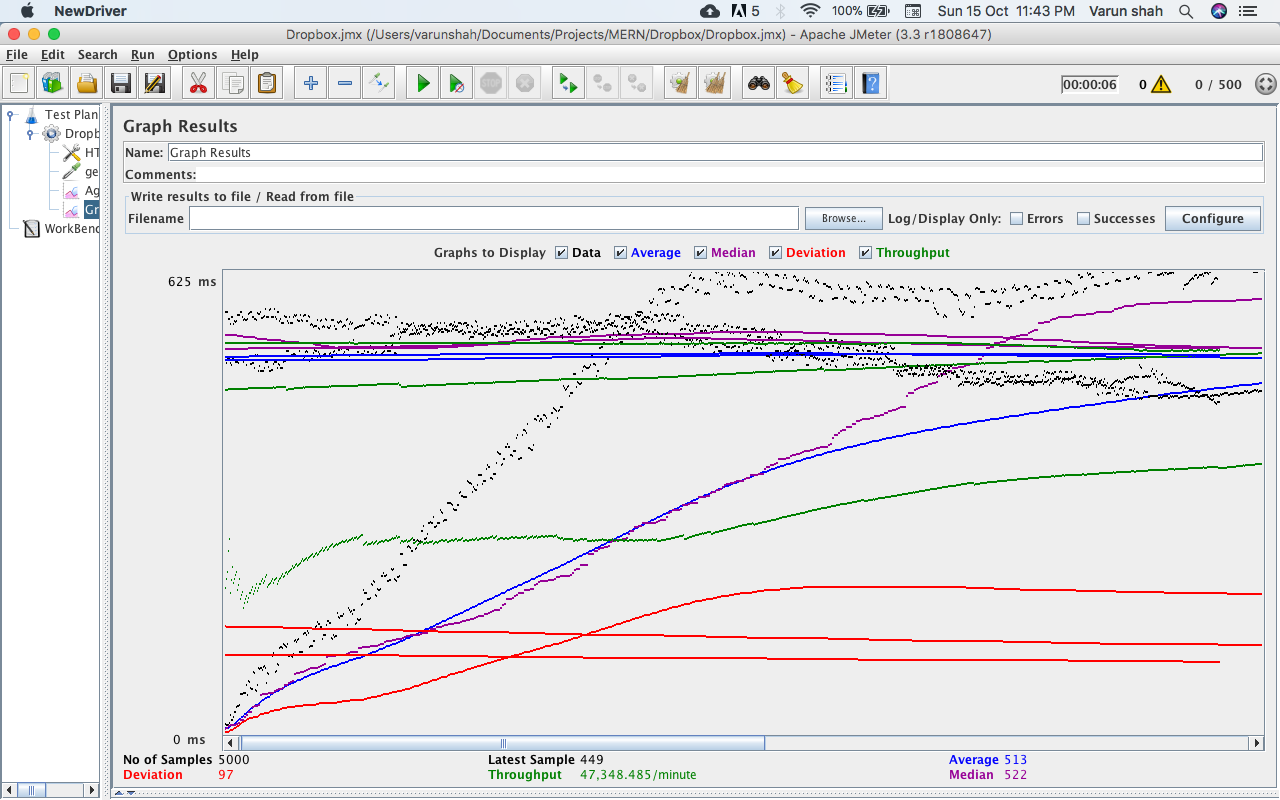


Aggregate Graph:

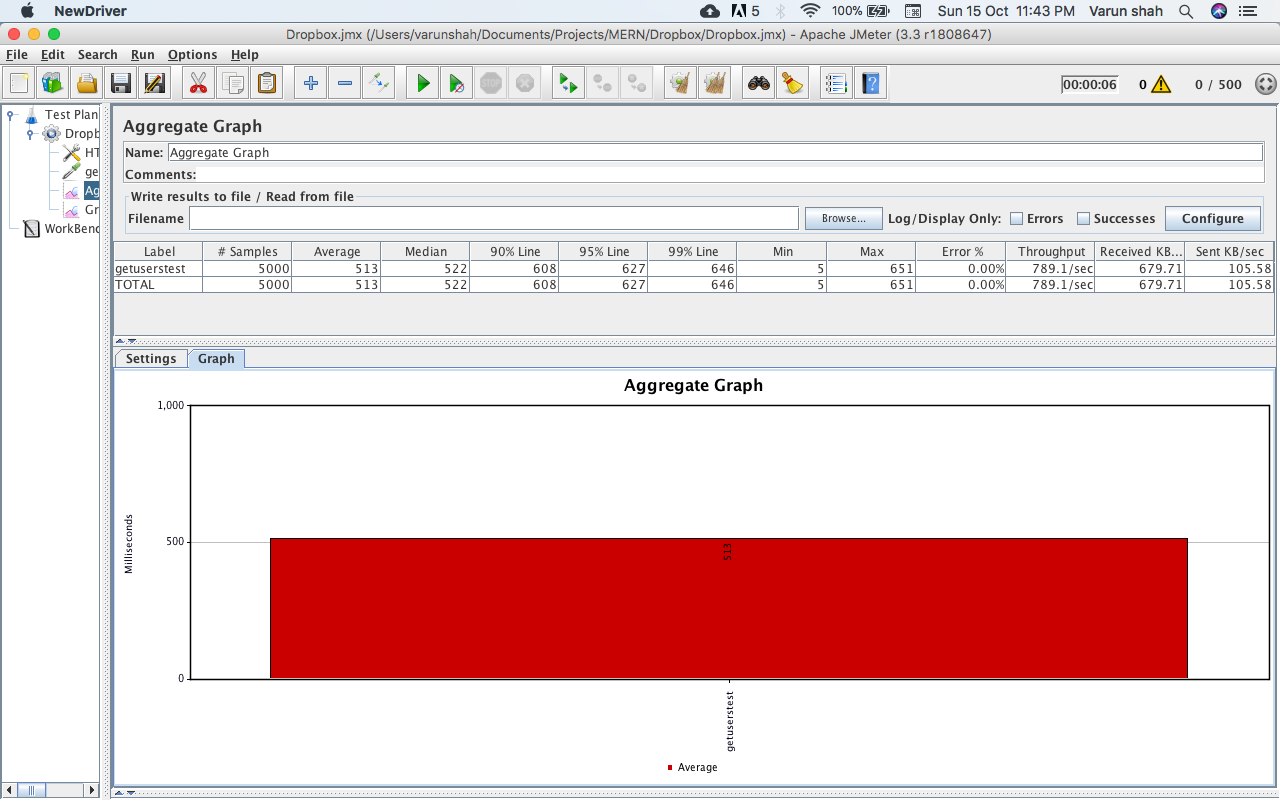


* + 500 concurrent users

Graph Result:

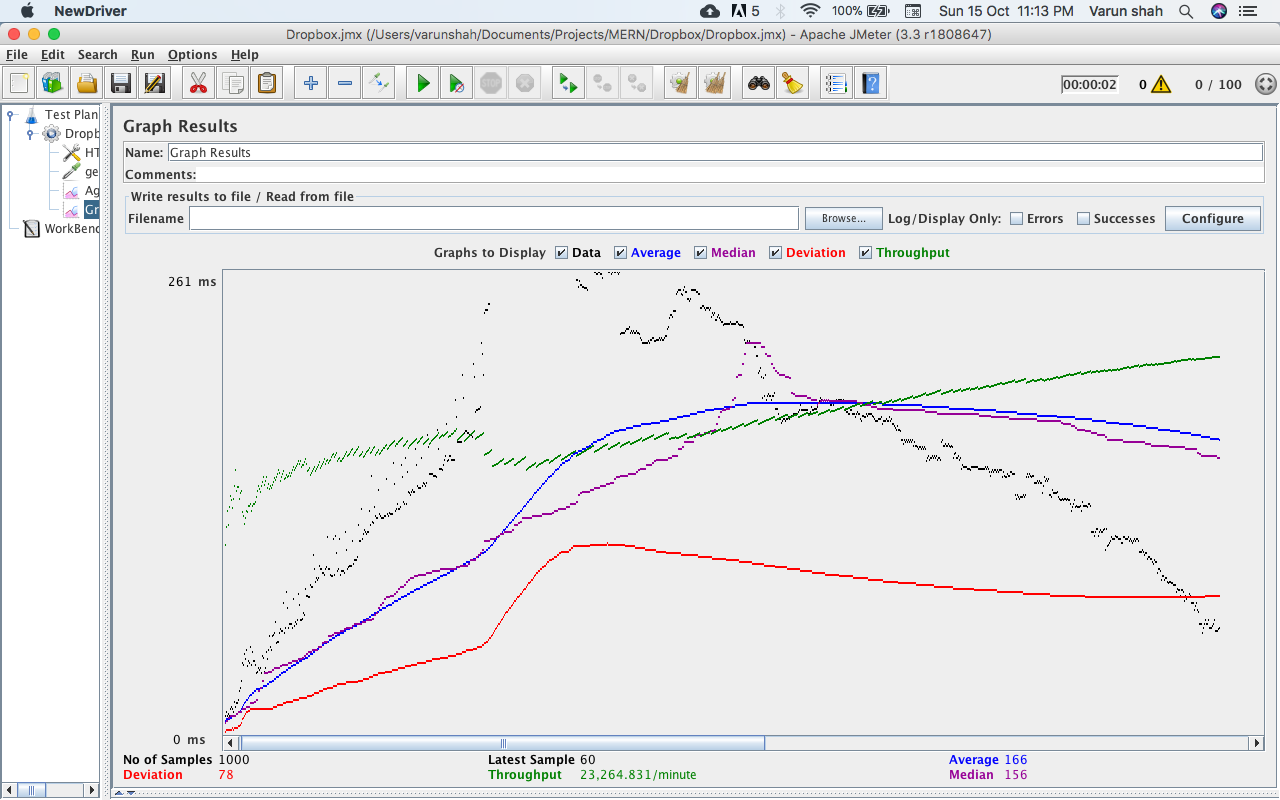


Aggregate Graph:

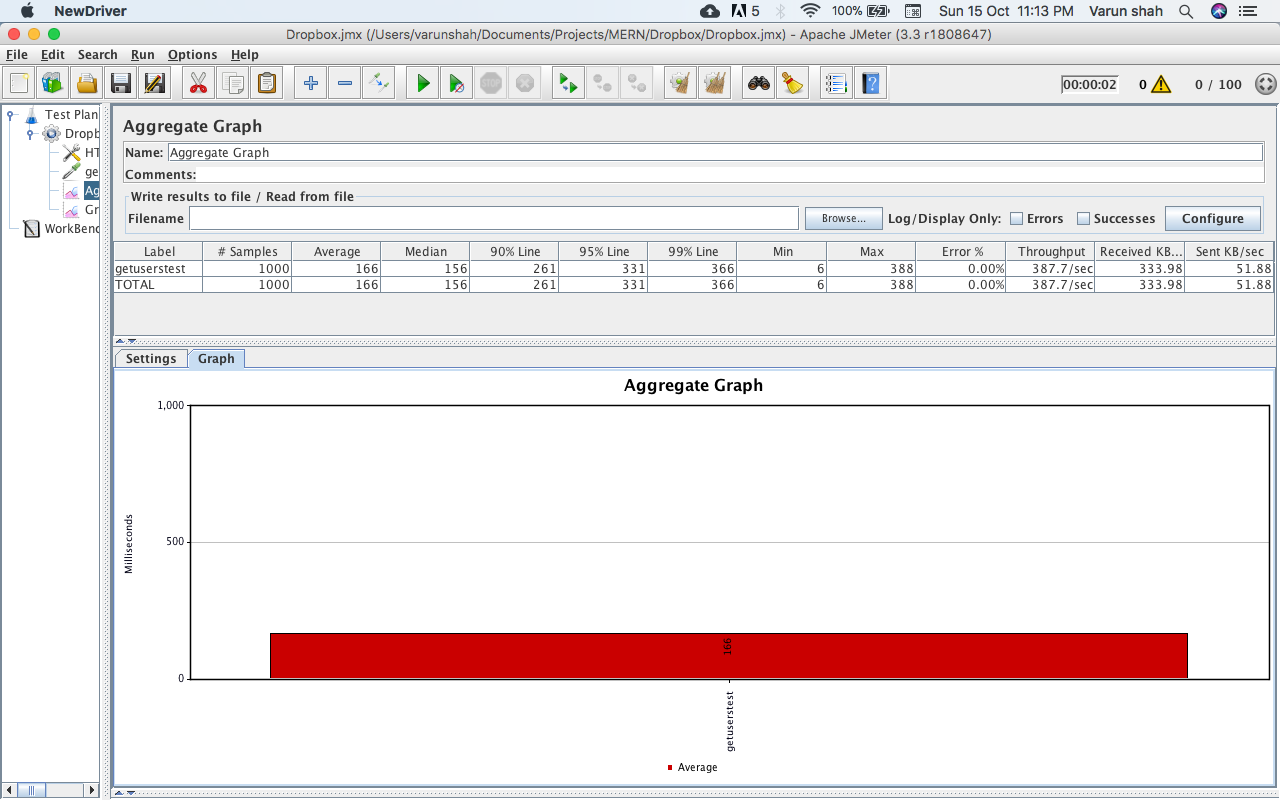


* Without connection pooling
  + 100 concurrent users

Graph Result

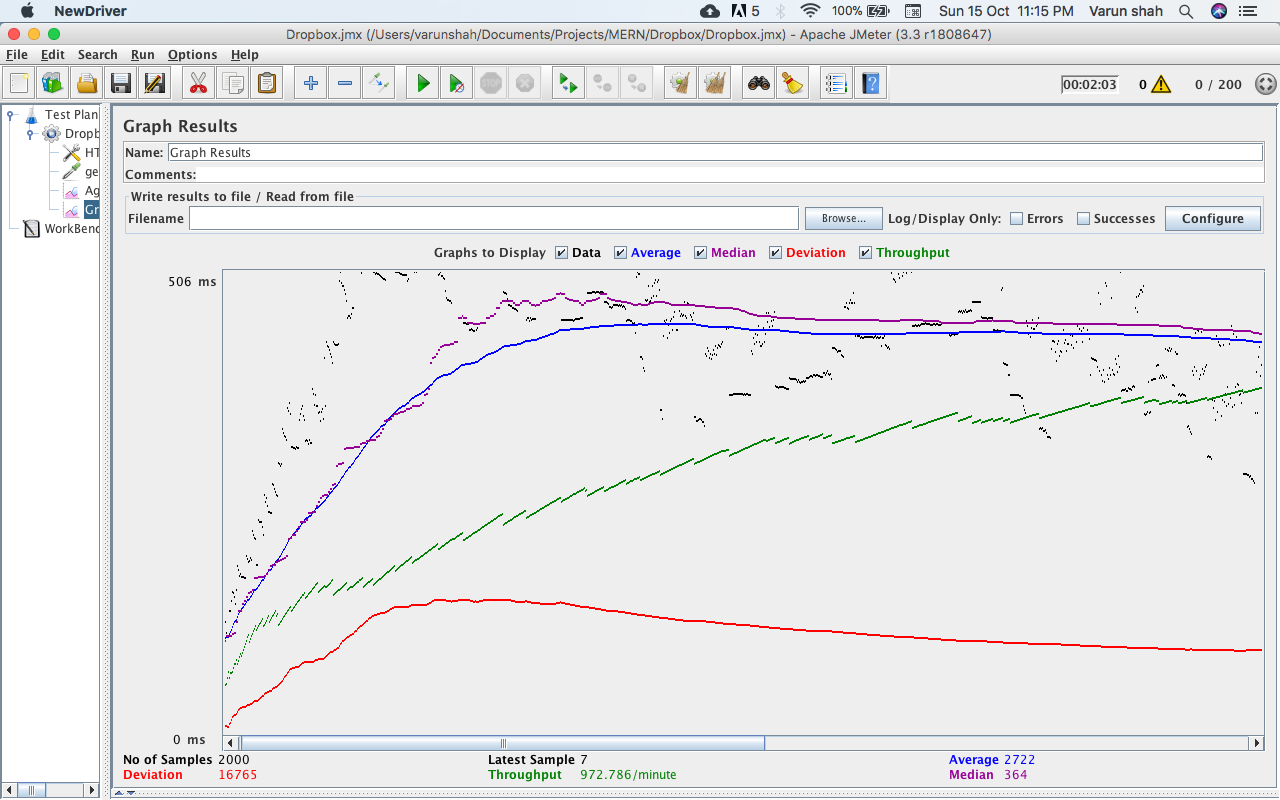


Aggregate Graph

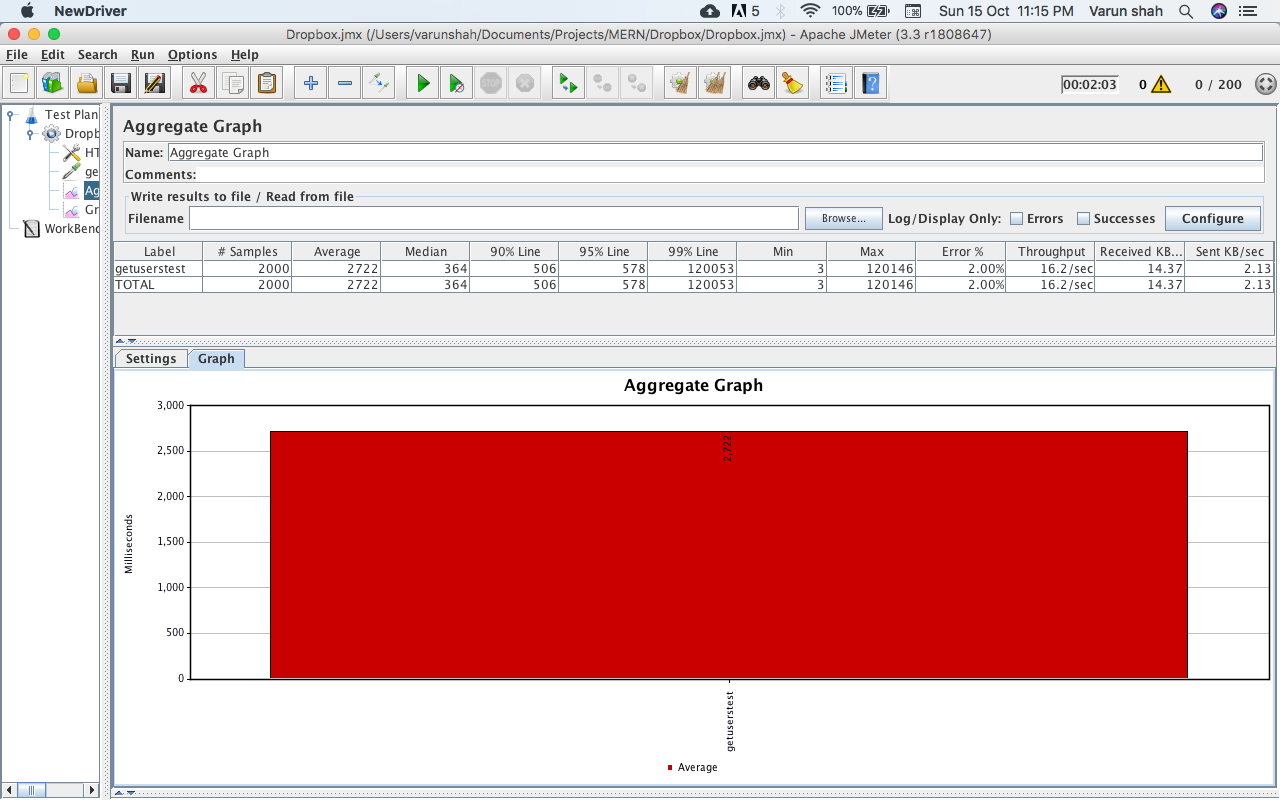


* + 200 concurrent users

Graph Result

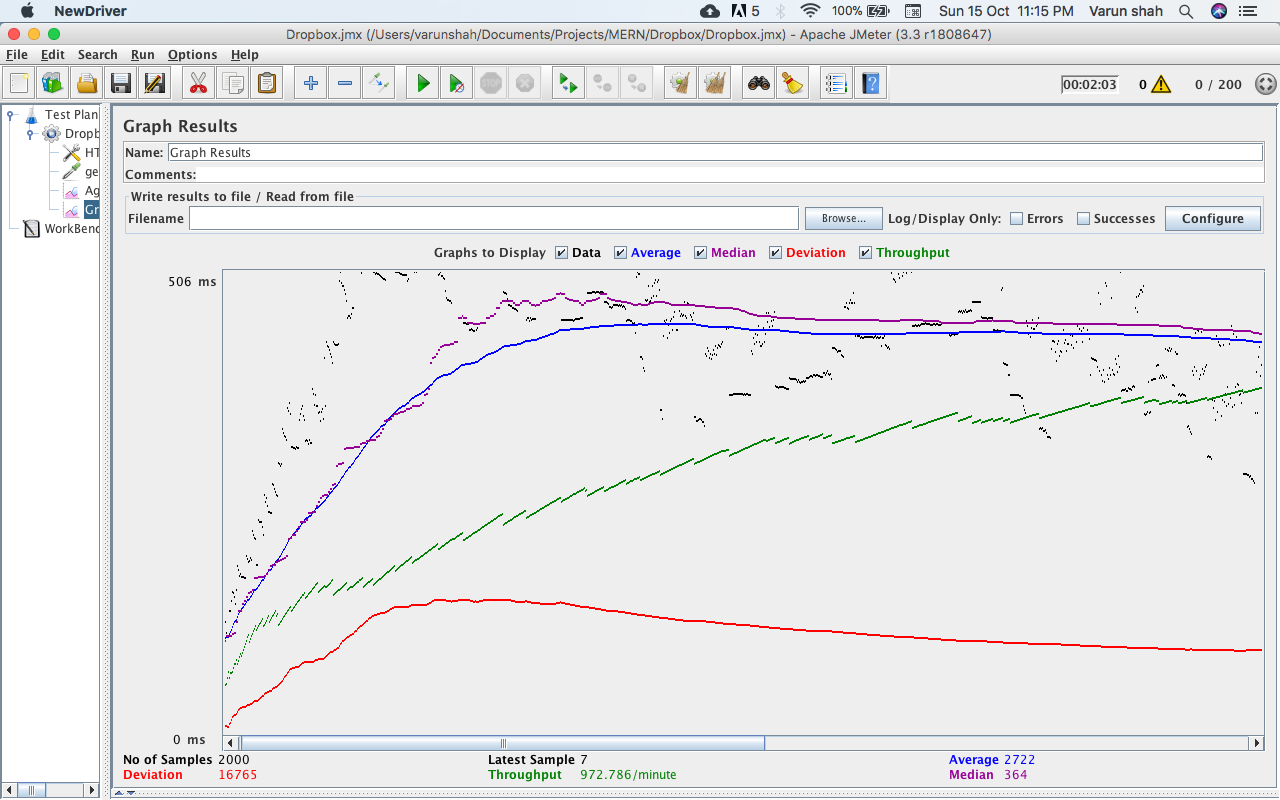


Aggregate Graph

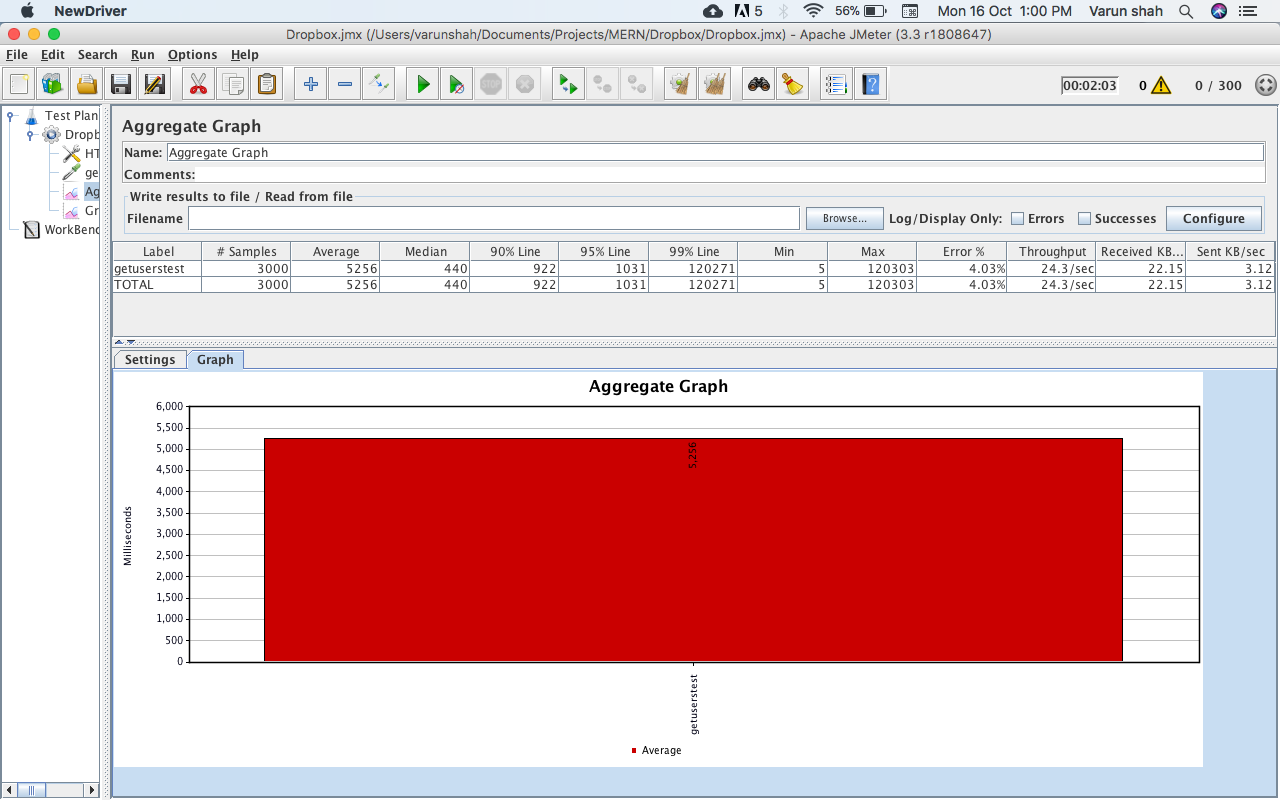


* + 300 concurrent users

Graph Result

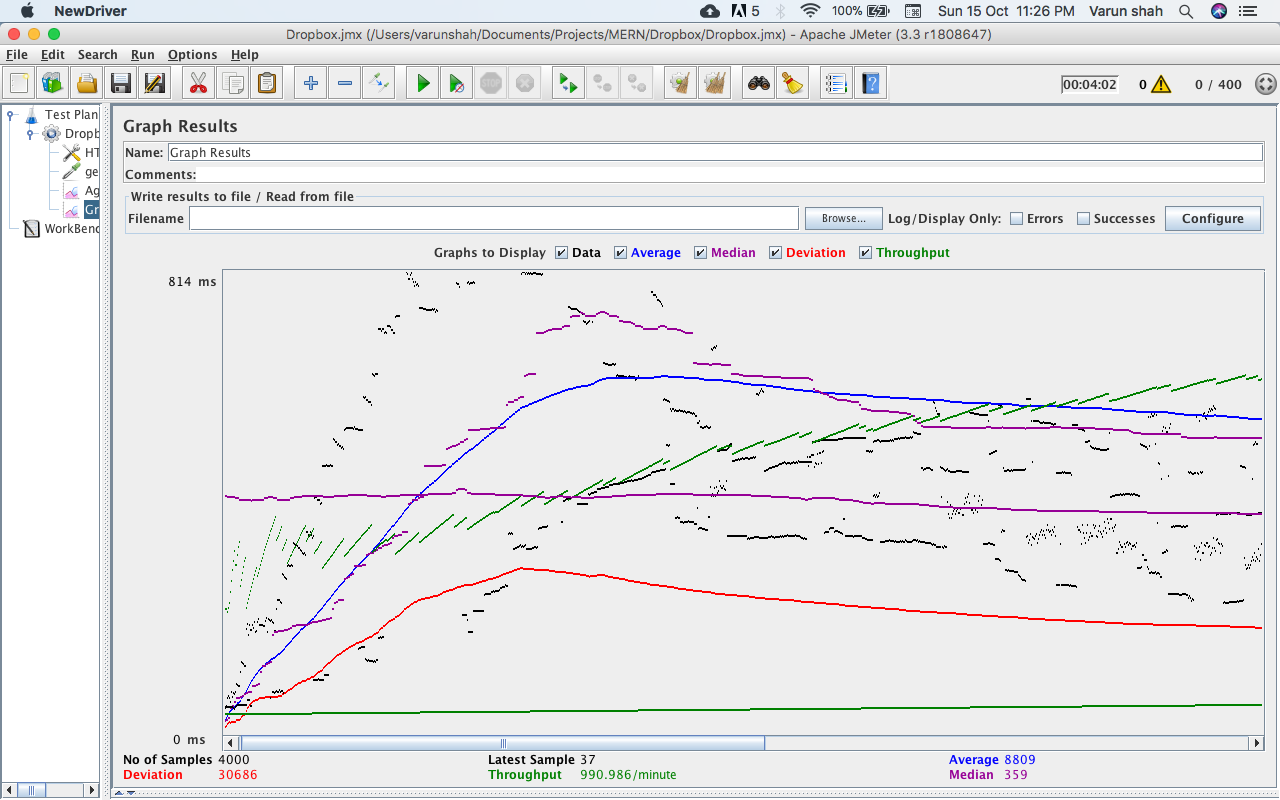


Aggregate Graph

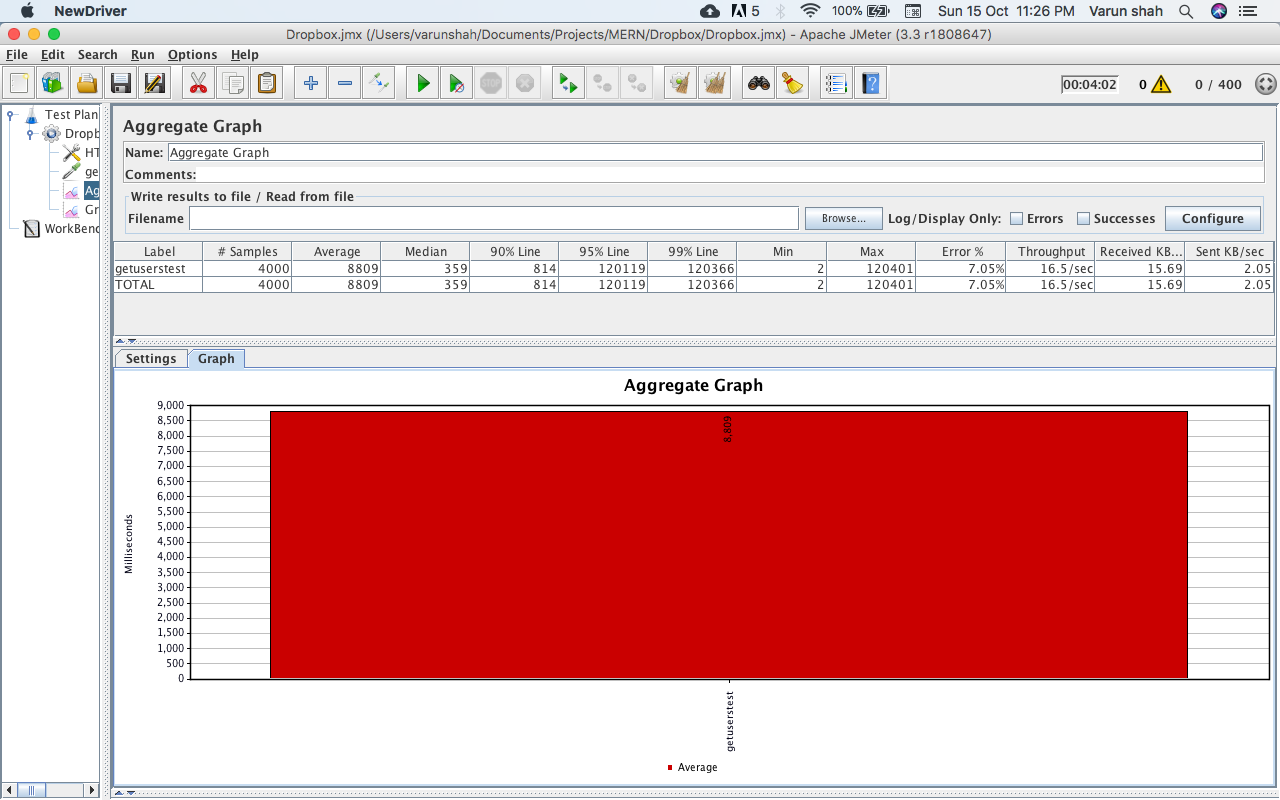


* + 400 concurrent users

Graph Result

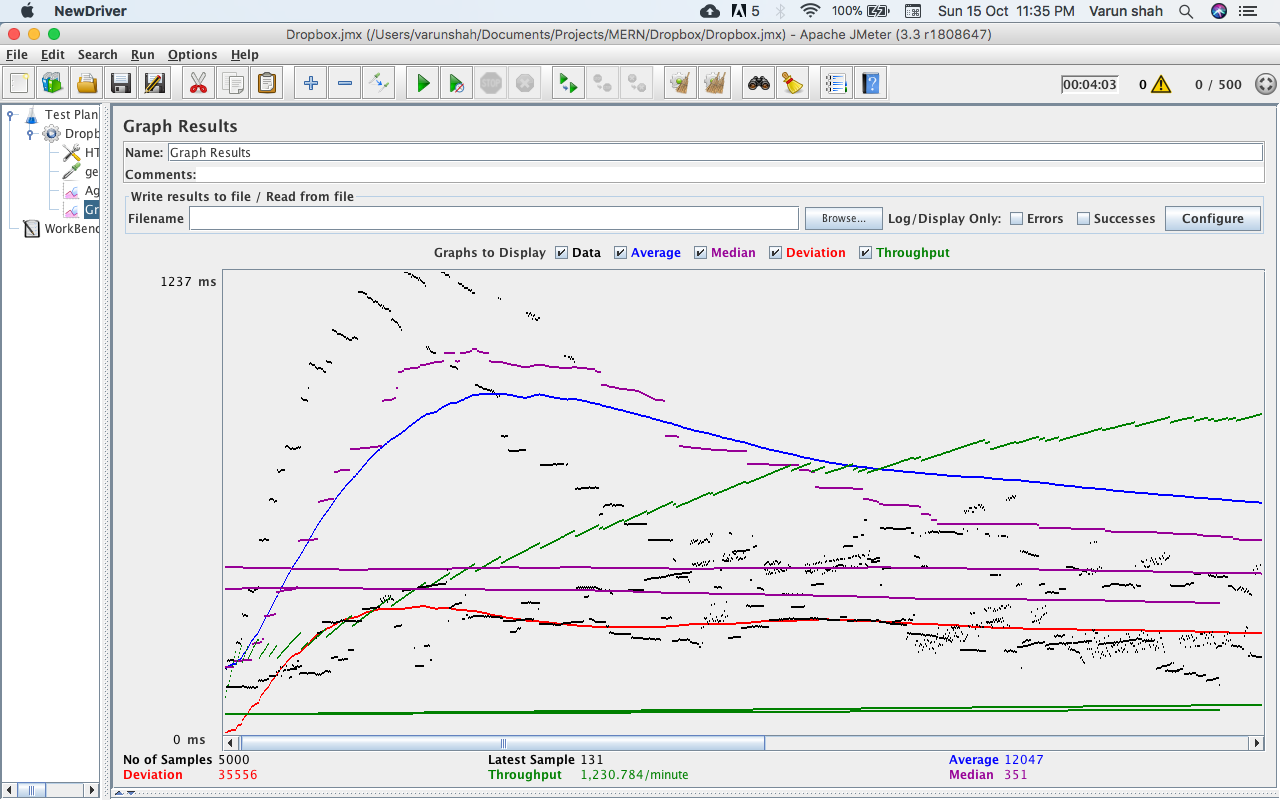


Aggregate Graph

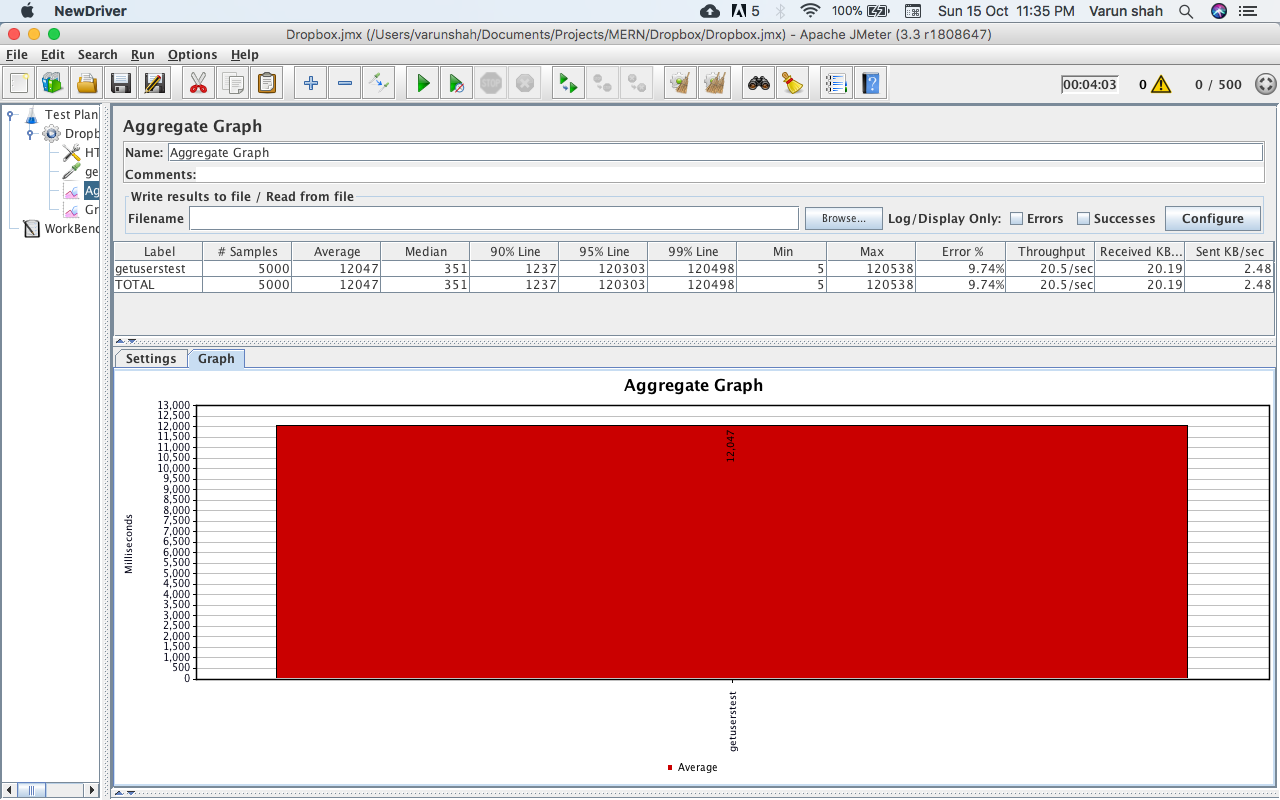


* + 500 concurrent users

Graph Result



Aggregate Graph



# Answers

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

nodejs library “bcrypt” provides **bcrypt algorithm** for encrypting and validating user’s password using salt value. It is based on **blowfish cipher**.

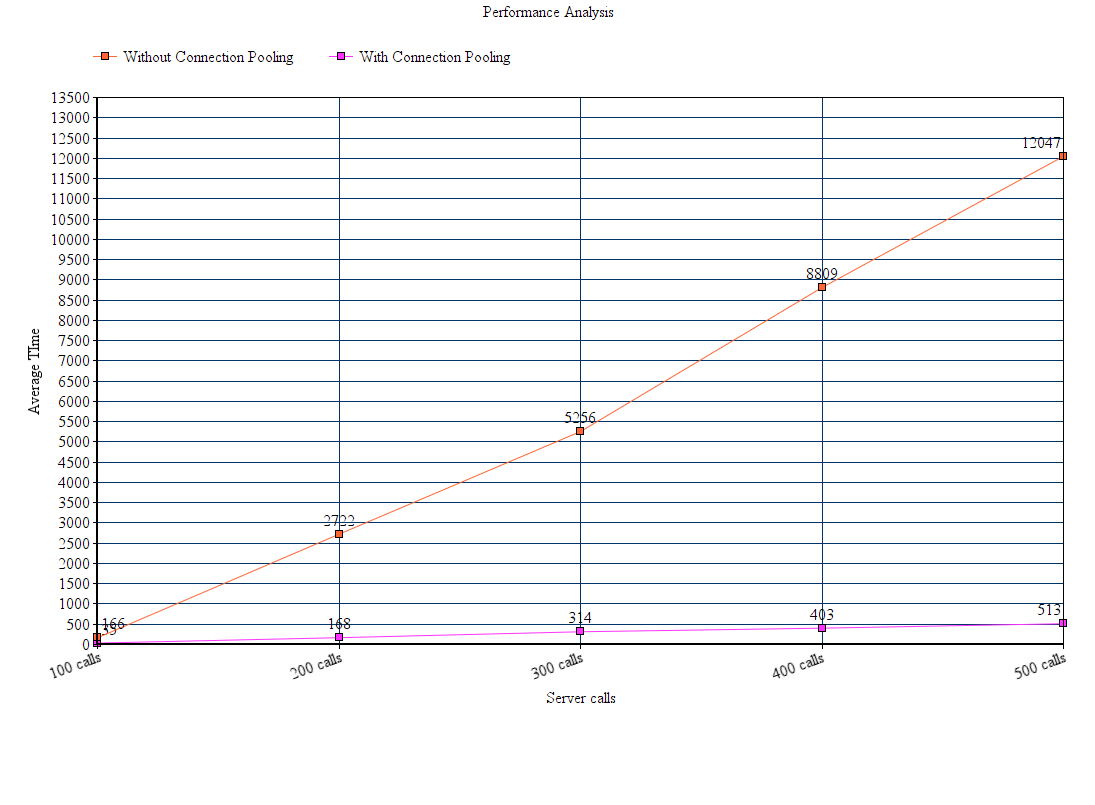
There are many other encrypting algorithms:

* AES
* SHA (1, 224, 256, 384, 512)
* RSA
* TRIPLE DES
* Twofish etc.

bcrypt algorithm was selected in this API 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. In fact, bcrypt has more resistant towards brute force attacks because of its adaptive functionality. 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 SALT that is used as an additional input.. This also prevents any rainbow table attacks.

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

1. 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.



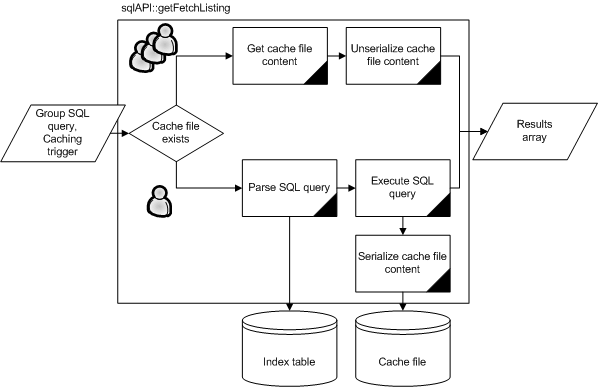
The graph above shows the comparison between the applications’ average response time when connection pooling is implemented and when it is not implemented.

When there are 500 concurrent users each having one call, the average response time is 12047ms without connection pooling is implemented, which reduces to 513ms after 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 the maximum size of pool is 10. When a user will access the server, he will get one connection from the pool. When the connection releases, it will be added back to the pool.

let pool= mysql.createPool({  
 connectionLimit: 10,  
 host : 'localhost',  
 user : 'root',  
 password : 'varun1993',  
 database : 'dropbox',  
 port : 3306,  
 debug : false  
});  
  
function insertData (callback,sqlQuery){  
  
 *console*.log("\nSQL Query:: " + sqlQuery);  
  
 // connection= getConnection();  
  
 pool.getConnection(function (err, connection){  
 connection.query(sqlQuery, function(err, result) {  
 if(err){  
 *console*.log("ERROR: " + err.message);  
 }  
 else  
 { // return err or result  
 *console*.log("DB Results:"+result.affectedRows);  
 callback(err, result);  
 }  
 });  
 *console*.log("\nConnection closed..");  
 // connection.end();  
 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.



As seen in the image above, the sql query execution plan or sql query is cached by the database system so it decreases the amount of time to execute a query. When server sends 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 does not need to evaluate different plans again for executing the query and will immediately need to evaluate different plans again for executing the request.

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

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. Along with a secret, when the cookie is passed by client in any request, the session variables are validated and the user session continues. When many requests are made by new clients, sessions are created and stored as cookies in their browser and when the subsequent requests are made along with cookie, after validating cookie data the user is authenticated. There is no dependency on database to check whether user is authenticated every time new request arrives. If the cookie gets deleted or session expires, then user will be logged out and forced to login again. Again session gets created.