**CMPE 273 Lab 1 Assignment**

by Sreedeep Katragadda (011545222)

Github link: <https://github.com/sreedeepk/CMPE273.git>

**Introduction:**

The purpose of this lab is to create a **Calculator** to demonstrate Stateless web services and to create a **Dropbox application** to demonstrate RESTful services. Our goal is to create a **Calculator** application which should perform the basic tasks: addition, subtraction, multiplication and division. We will be using **JMeter** for testing and we will calculate the average time required for each operation. For **Dropbox application,** we have to implement several functionalities such as:

* **User functionalities:**
  + Sign up a new user.
  + Sign in existing user.
  + Sign out.
  + Upload a file.
  + List a file.
  + Create a directory.
  + Star a folder/directory.
  + Share a folder/directory by email/name/link.
* **Users account:**
  + About: User overview, Work and Education.
  + Interests.
* **Providing file list and activity report.**
* **Providing Group functionalities:**
  + Create a group.
  + Add member in a group.
  + Show members in a group.
  + Assign access permission to a directory.
  + Delete member from a group.
  + Delete a group.
* **Should perform connection pooling for databases.**

Testing for this Dropbox application will be done using JMeter and Mocha.

**System Design:**

**Server 1:**

The server 1 for this Lab 1 is to create a Node.js based server to perform calculator operations. Before getting started, we should install the required dependencies for our backend server. Initially, we should create a *package.json* file which consists of all **npm** packages which are needed for the server. To initialize our project, we will run the command:

npm init

This creates a package.json file in our folder. We will give details about the project:

|  |
| --- |
| { |
|  | "name": "server1", |
|  | "version": "1.0.0", |
|  | "description": "CMP273 lab1 - server1", |
|  | "main": "app.js", |
|  | "scripts": { |
|  | "start": "nodemon app.js", |
|  | "test": "echo \"Error: no test specified\" && exit 1" |
|  | }, |

The dependencies which we used are: *bodyparser, cors, express, express-validation, joi, lodash.* The usage of these dependencies are:

* Bodyparser – This will parse incoming request bodies in a middleware before the handlers. Body-parser exposes the body portion on ‘req.body’ by extracting it from incoming request stream.
* Cors – This is a package for providing a Express middleware that can be used to enable CORS.
* Express-validation - express-validation is a middleware that validates the body, params, query, headers and cookies of a HTTP request and response with errors is returned if any of the configured validation rules fail.
* Joi – This is used to validate if the parameters of a HTTP request are matched with our required parameters i.e., numbers. If inserted parameters are not numbers, this package will throw errors if it is not a number.
* Lodash – This is very helpful for iterating arrays, objects, & strings, manipulating & testing values, creating composite functions.

To install a dependency, we use the command:

* npm install body-parser –save

For the server side, we used four end points using ‘get’ function for our four calculator operations. These can be accessed using:

* app.get(‘/add’)
* app.get(‘/subtract')
* app.get(‘/multiply’)
* app.get(‘/division’)

**Client1:**

We built an API where the user will input two values on client side and the client will take the values and pass it to the server side where the operations are taken place and an output is produced. And this output is received back at the client side and displayed. We used Reactjs for the client side of the Calculator App.

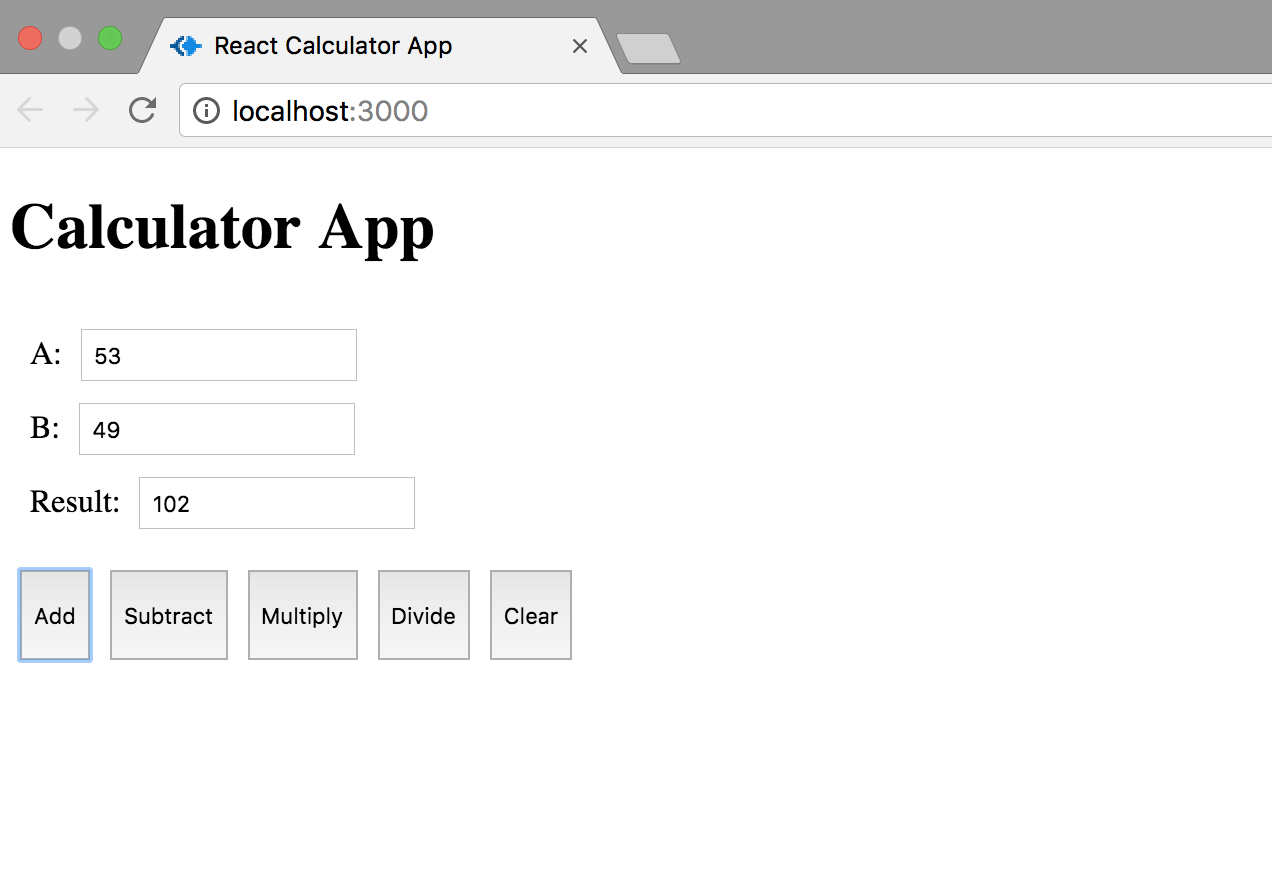
**Results:**

**Calculator App:**

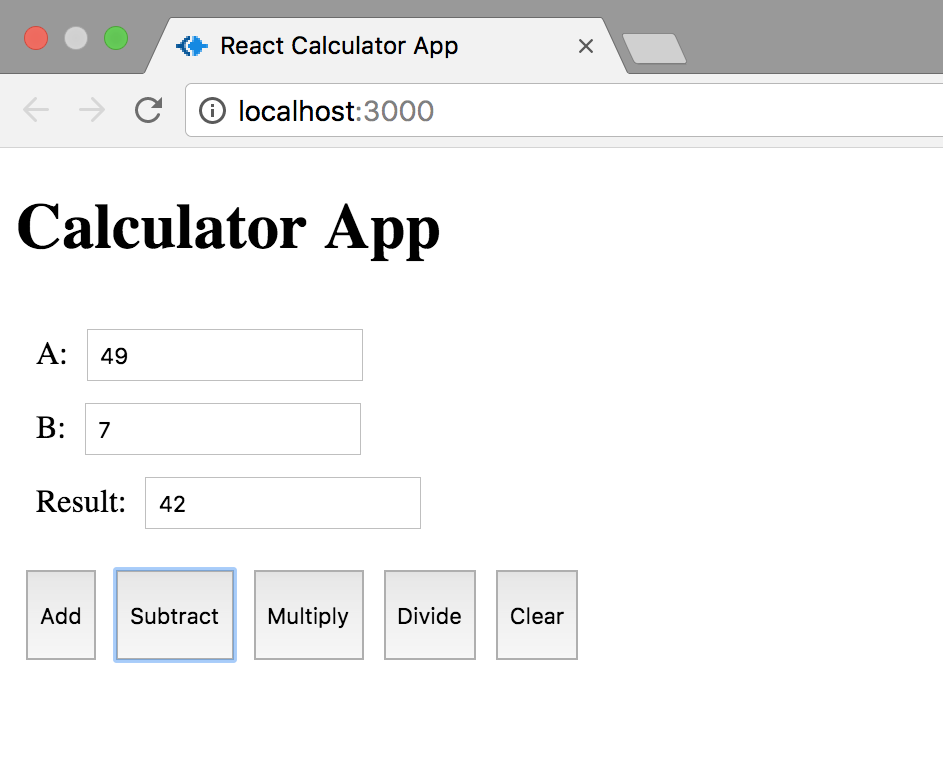
To start the application, go to the directories server1 and client1 and use:

* npm install
* npm start

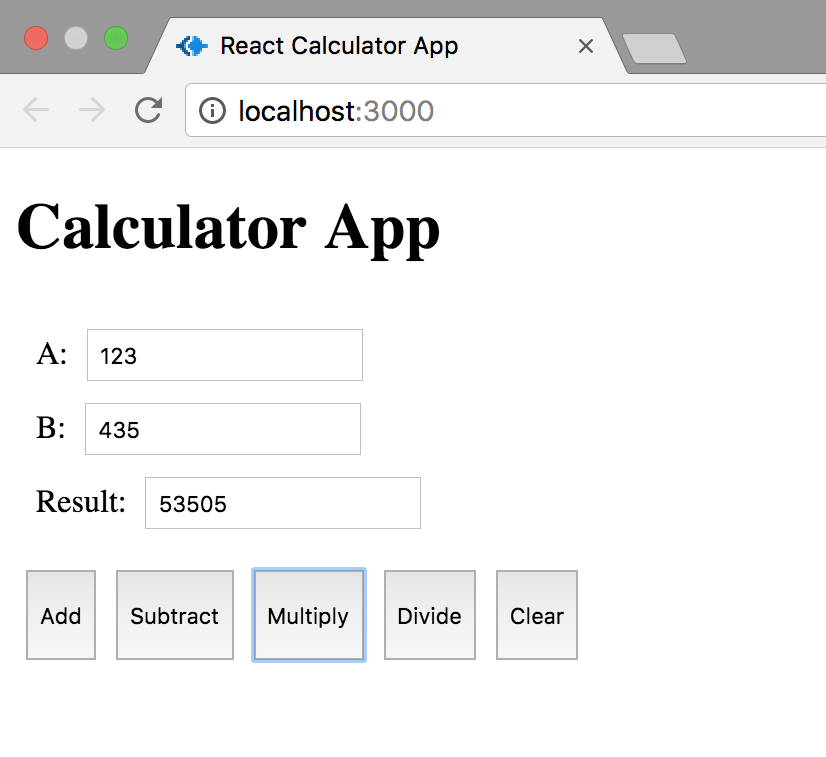
The front-end of the calculator application will be shown as below:



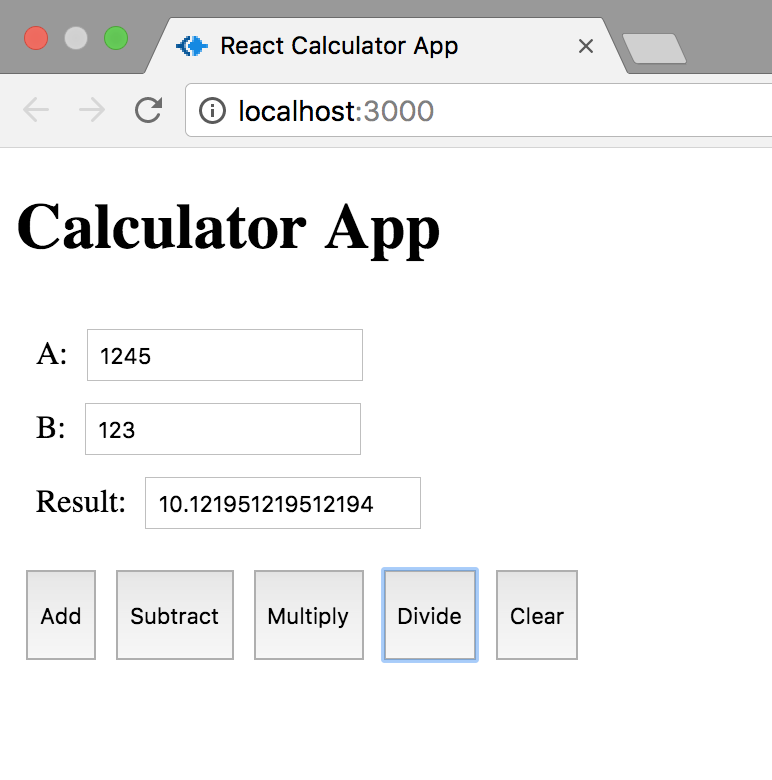
**Addition Operation**

****

**Subtraction Operation**

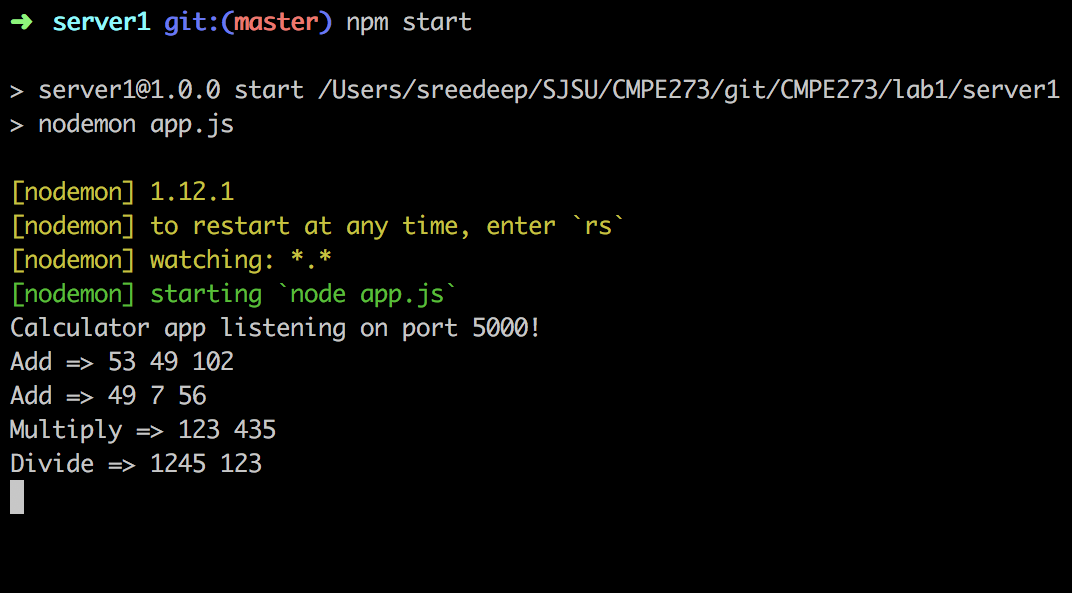
****

**Multiplication Operation**

****

**Division Operation**

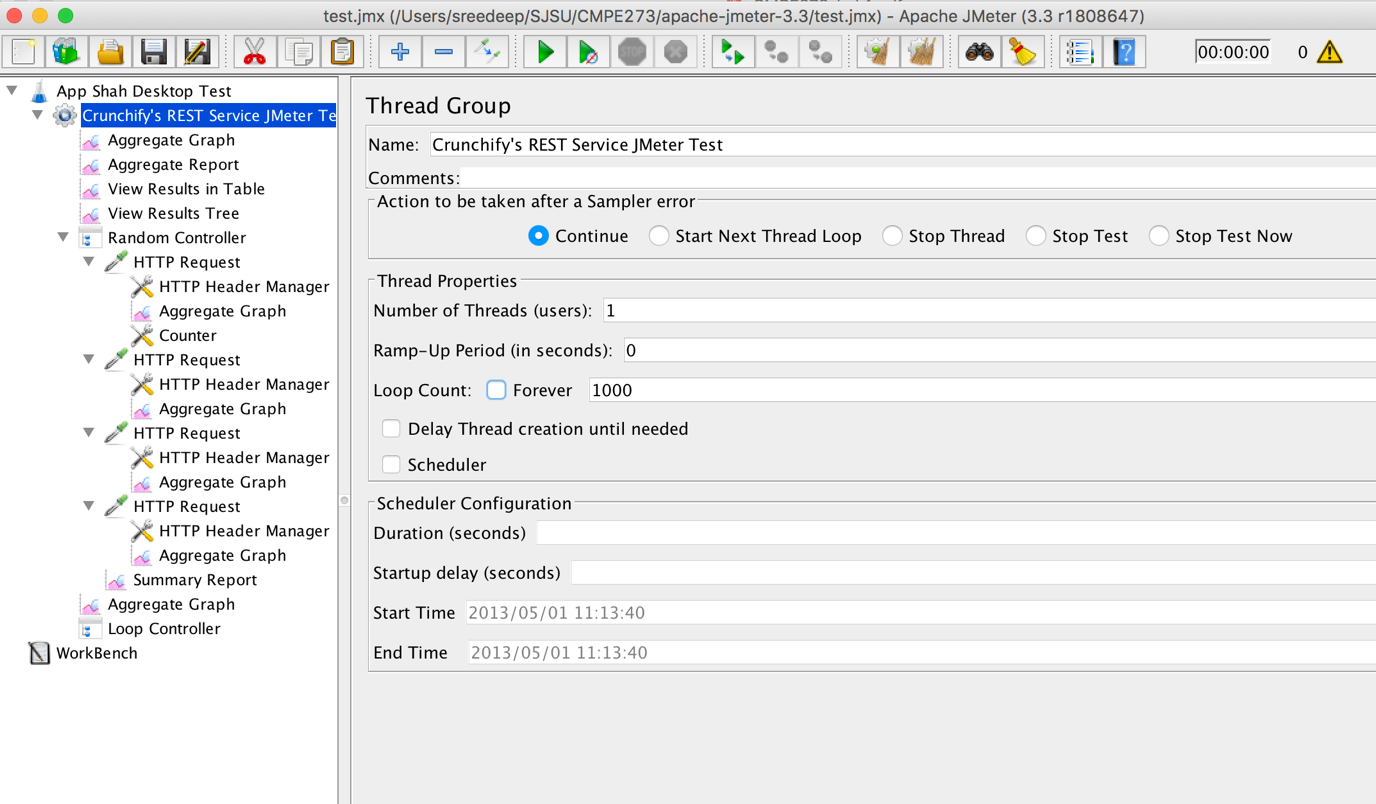
As we can see, all the four operations are working properly on the client side, now we will see how the operations are being carried out on the server side.



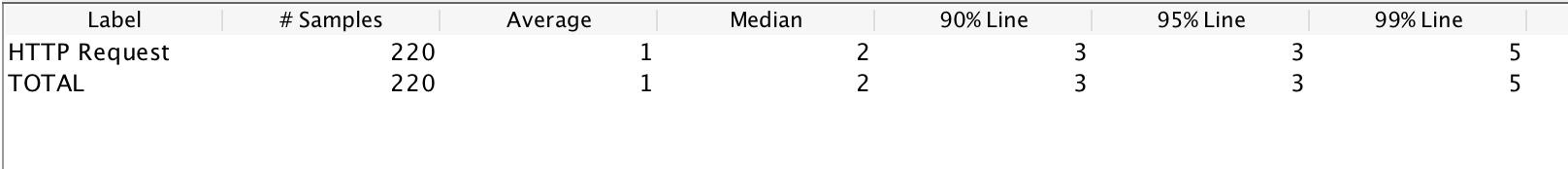
All the operations for our Calculator App are working effectively. Now, we will test the average time required for each of these operations using JMeter.

**Performance:**

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

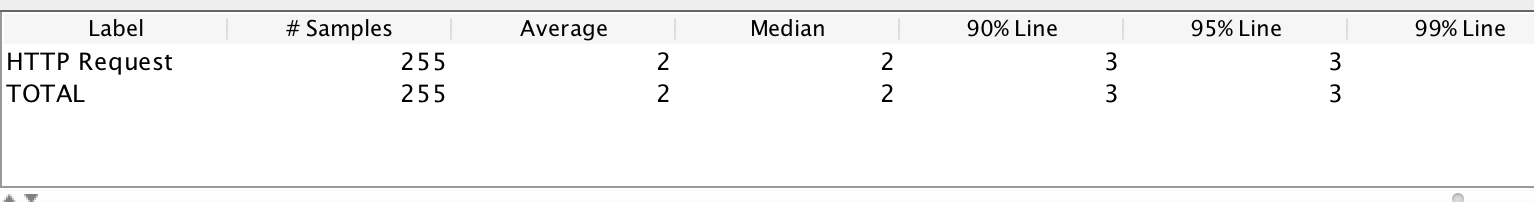


As we can see from the above image, for a single user we are calling 1000 calculator calls on randomly selected tasks. We are sending the parameters randomly in the range of 0 to 1000 using: ${\_\_Random(0,1000)}. Then, the average times are generated after starting the automated testing process:



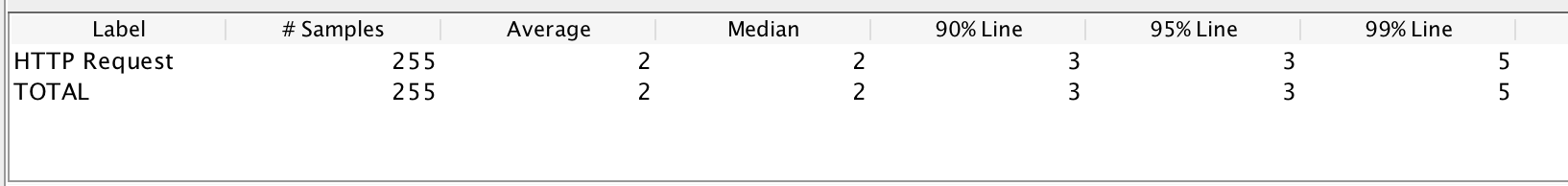
**Addition Operation**

The average time for addition operation for 1000 calculator calls is **1ms**.

****

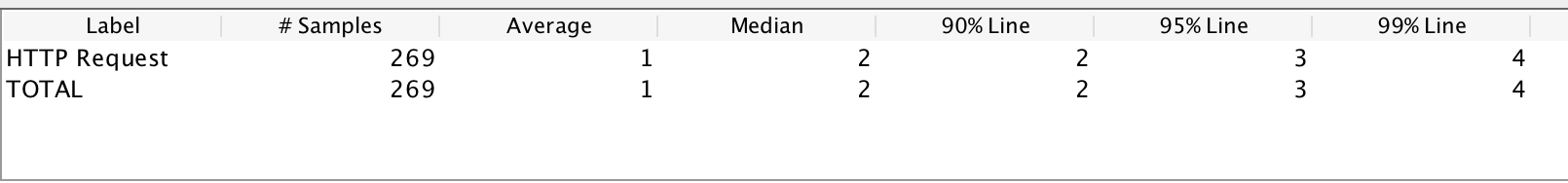
**Subtraction Operation**

The average time for subtraction operation for 1000 calculator calls is **2ms**.



**Multiplication Operation**

The average time for multiplication operation for 1000 calculator calls is **2ms**.

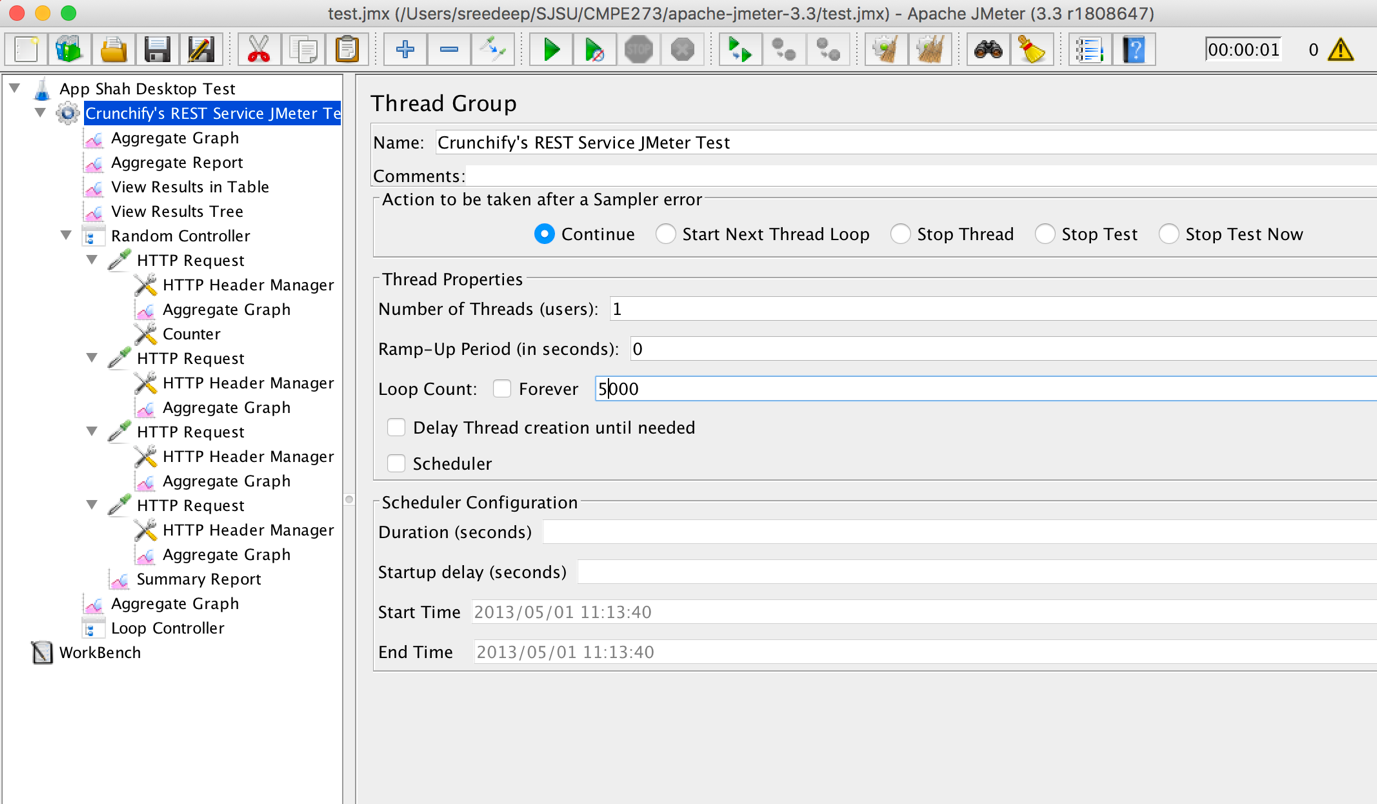


**Division Operation**

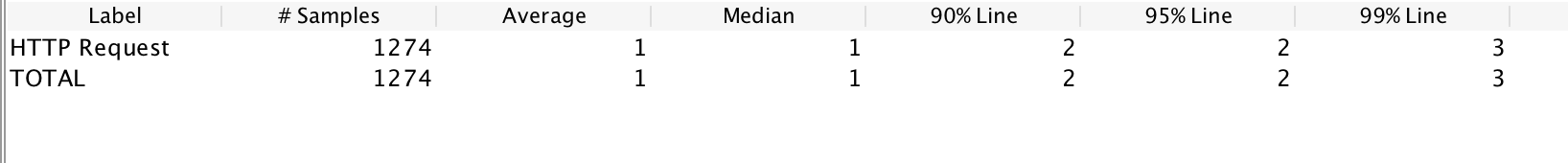
The average time for division operation for 1000 calculator calls is **1ms.**

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

In this, we will be carrying out 5000 calculator calls on randomly selected tasks and we will print the average time to perform each operation.

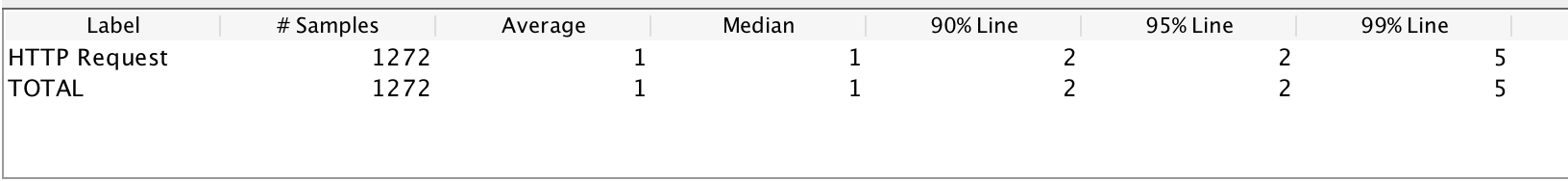


Let’s find the average time for each operation for 5000 calculator calls.



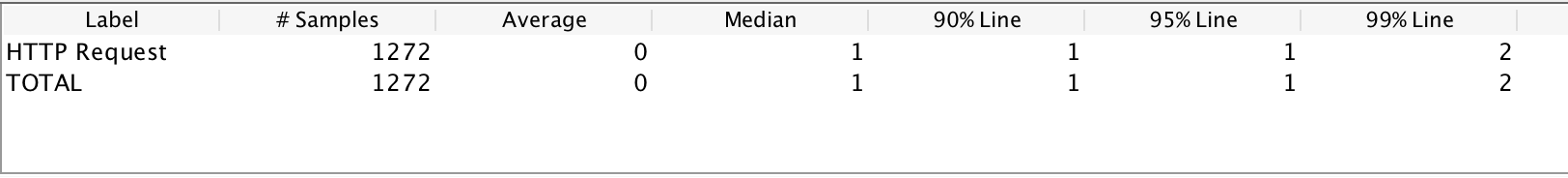
**Addition Operation**

The average time for addition operation for 5000 calculator calls is **1ms**.



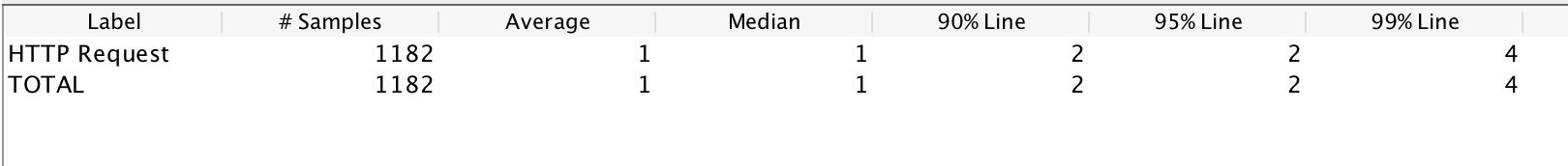
**Subtraction Operation**

The average time for subtraction operation for 5000 calculator calls is **1ms**.



**Multiplication Operation**

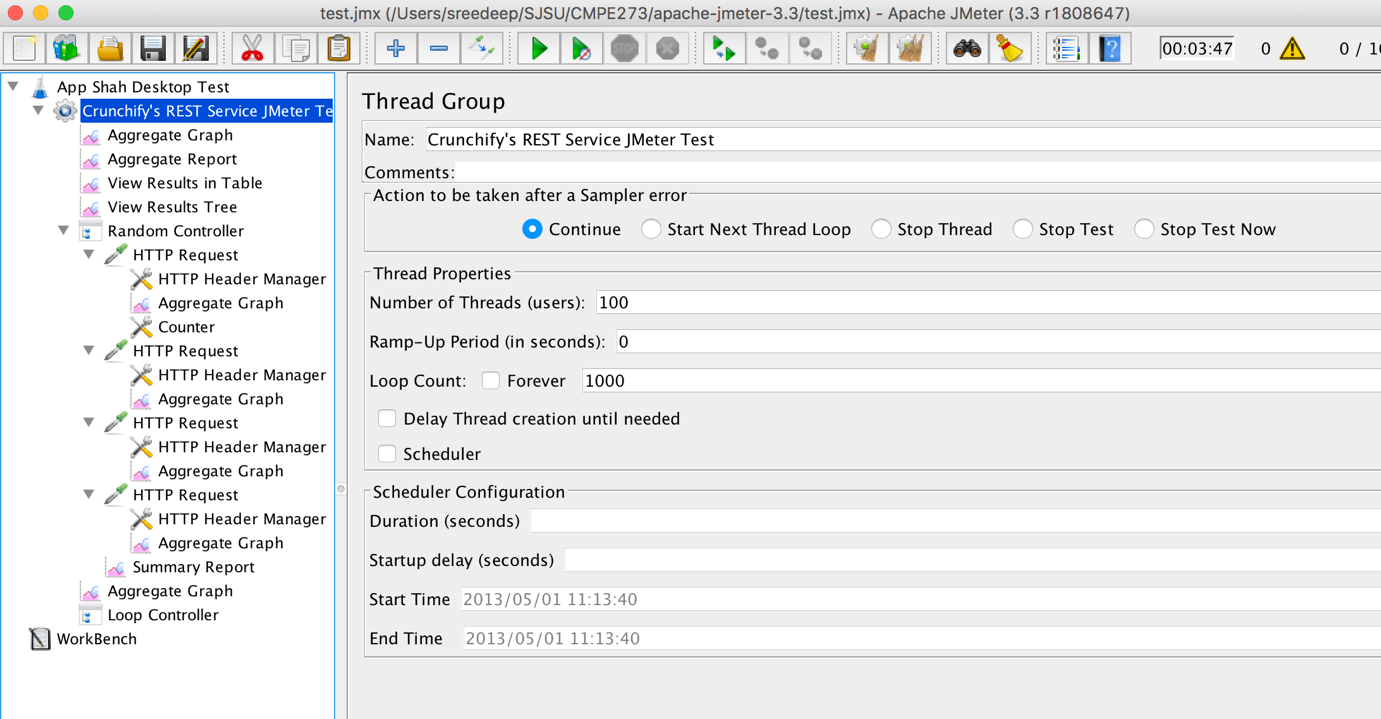
The average time for multiplication operation for 5000 calculator calls is **0-1ms**.



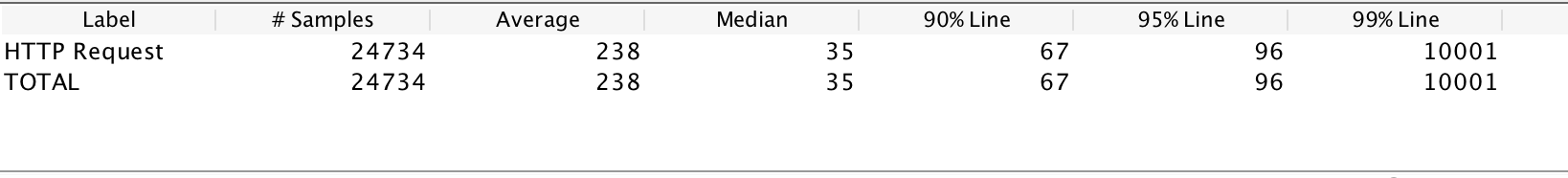
**Division Operation**

The average time for division operation for 5000 calculator calls is **1ms**.

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

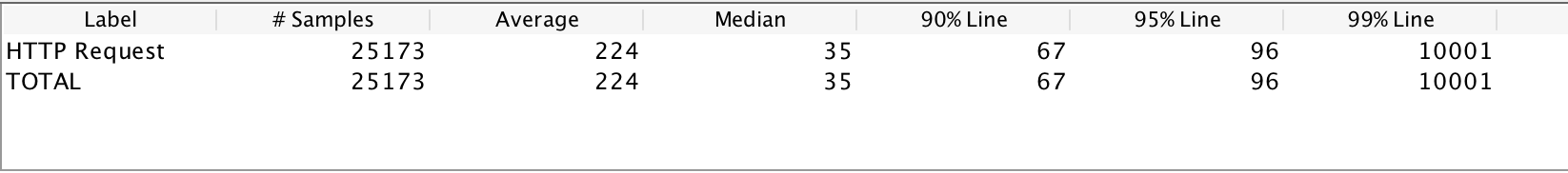
****

Here, we will invoke 100 concurrent users with 1000 calls each to calculator on randomly selected operations and print the average time to perform each operation.



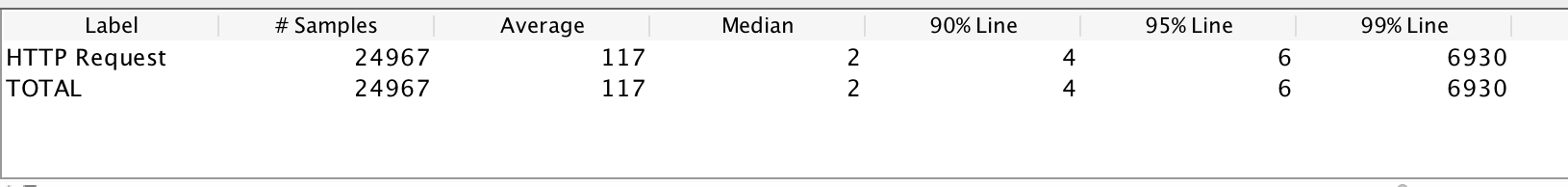
**Addition Operation**

The average time for 100 concurrent users with 1000 calls for addition operation is **238ms**.



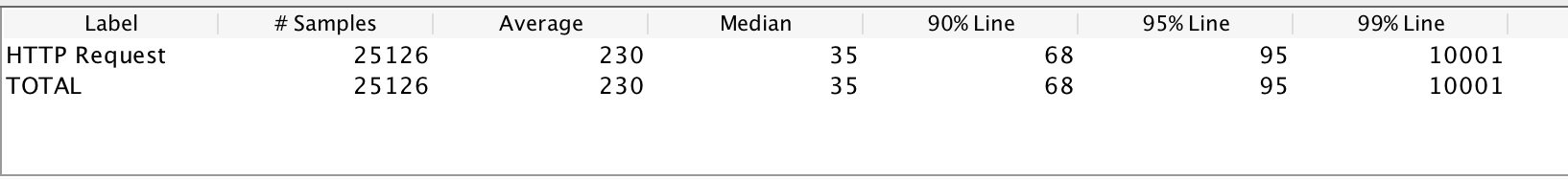
**Subtraction Operation**

The average time for 100 concurrent users with 1000 calls for subtraction operation is **224ms**.



**Multiplication Operation**

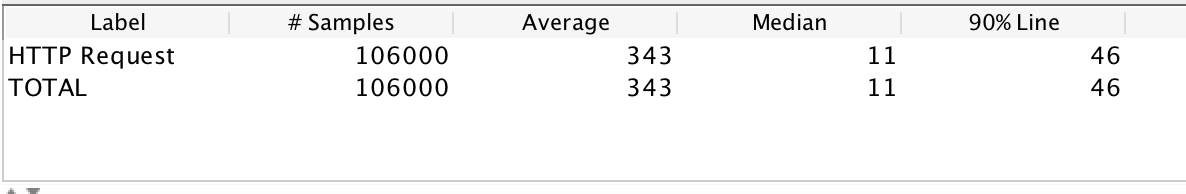
The average time for 100 concurrent users with 1000 calls for multiplication operation is **117ms**.



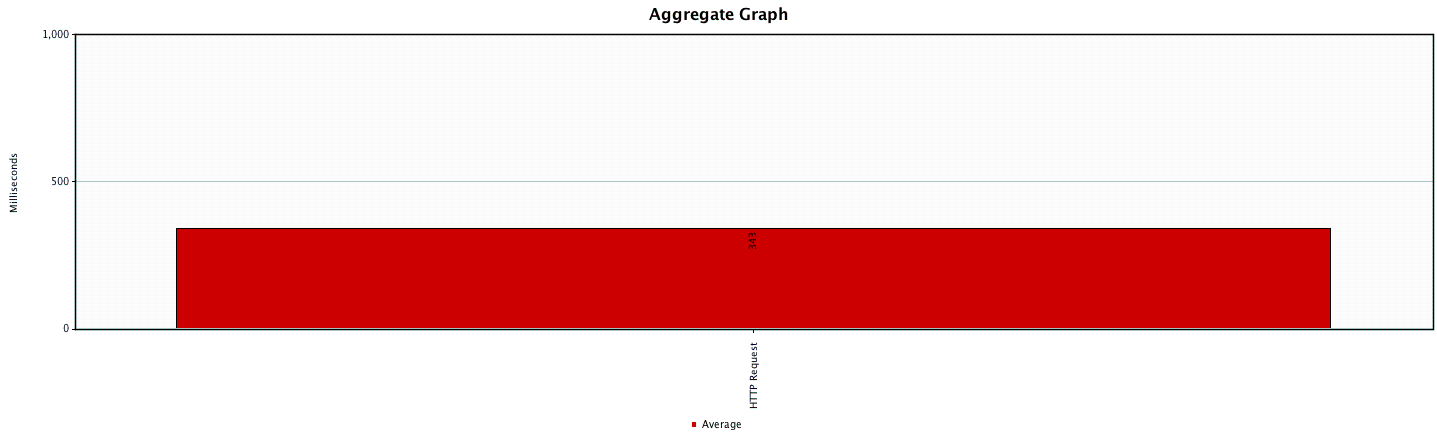
**Division Operation**

The average time for 100 concurrent users with 1000 calls for division operation is **230ms**.

**Q. Draw a graph with average time and include it in the report.**



As we can see, for all the samples the average time took for all the operations is **343ms**. The aggregate graph of the average time for all the operations will be:



**Aggregate Graph**

**System Design:**

**Server 2:**

The server 2 of this lab is to create a Node.js based server to create a “Prototype of the Dropbox application”. The description of the package.json file is as follows:

{

"name": "server2",

"version": "1.0.0",

"description": "CMP273 lab1 - server2",

"main": "app.js",

"scripts": {

"start": "nodemon app.js",

"test": "echo \"Error: no test specified\" && exit 1"

},

The dependencies which we used are: *body-parser, client-sessions, cors, express, express-validation, joi, lodash, multer, mysql, node-cmd, nodemon. The usage of the other dependencies are:*

* Client-sessions: client-sessions is a middleware that implements sessions in encrypted cookies.
* Multer: This is a middleware which is primarily used for uploading files.
* Nodemon: Nodemon is primarily used to reconnect to the server automatically after making any changes without re-establisihing the connection.

There are several end-points for the server 2 and they are described below:

* **POST '/users/signup'**

This is the end-point which is used for a user to sign up. The parameters for this end-point are Email, password, First Name, Last Name, Overview, Work and Interests.

* **POST '/users/signin'**

This is used for the user to sign in to their Dropbox account. The parameters for this end-point are Email and password.

* **POST '/users/signout'**

This end-point is used for the user to sign out of their account.

* **GET '/users/info'**

This end-point can be accessed by using get method. The function of this is to read the info of a certain user.

* **POST '/files/upload'**

The use of this end-point is to upload the files into their account.

* **GET '/files'**

This is used to display all the files.

* **GET '/files/:id'**

This is used to display a particular file using their ID rather than displaying all the uploaded files.

* **POST '/groups/create'**

This is used to create groups in the Dropbox.

* **POST '/groups/:id/members'**

This is used to create members in a particular group. The group can be accessed using the group ID.

* **GET '/groups/:id/members'**

This end-point is used to display all the members in a particular group. This group can be accessed using the group ID.

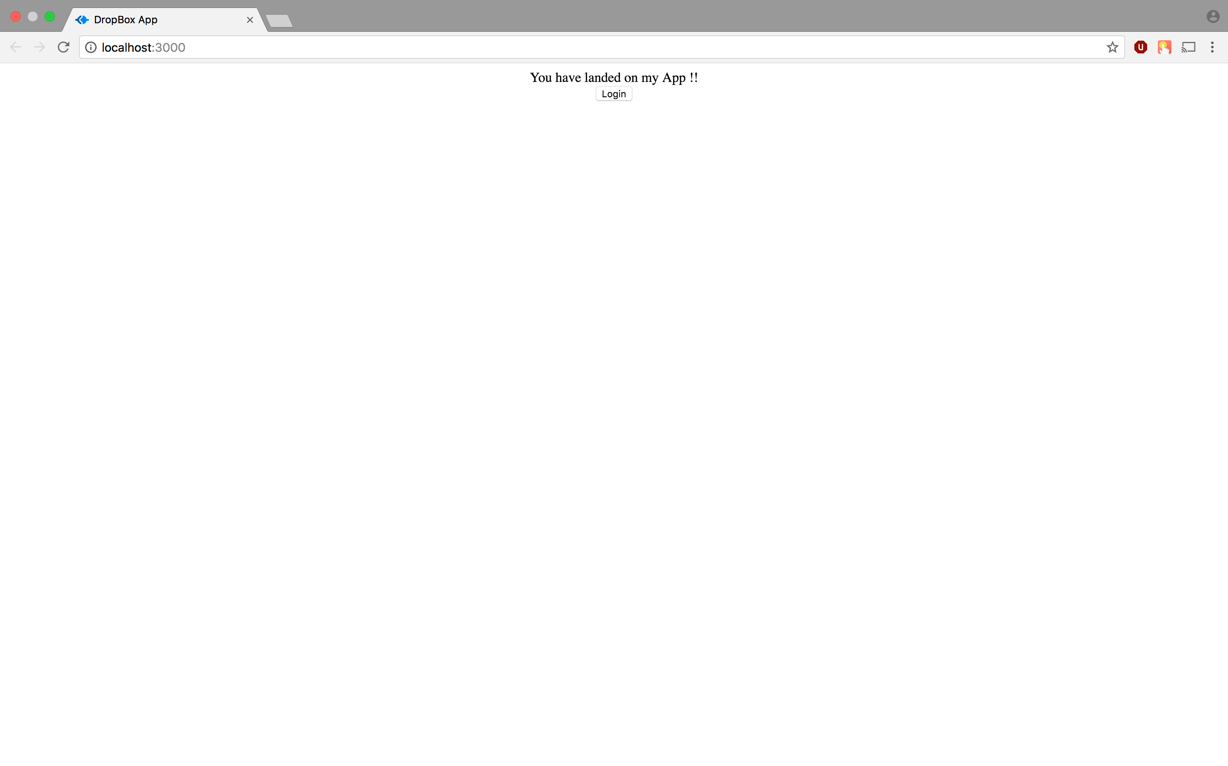
* **DELETE '/groups/:id/members'**

This end-point is used to delete members from a group. The group can be accessed using the group ID.

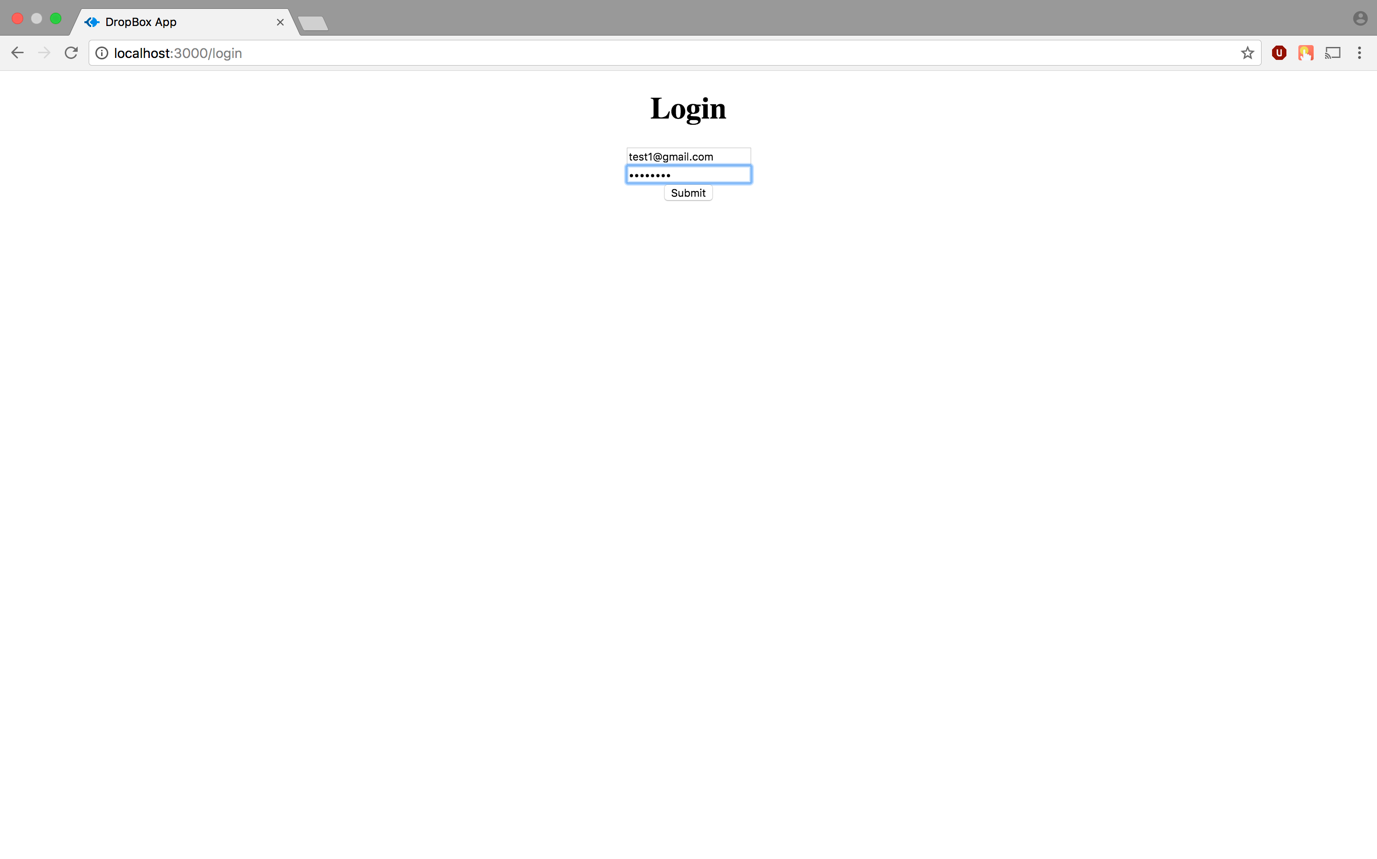
* **DELETE '/groups/:id'**

This end-point is used to delete a group. The group can be accessed using the group ID.

**Client 2:**

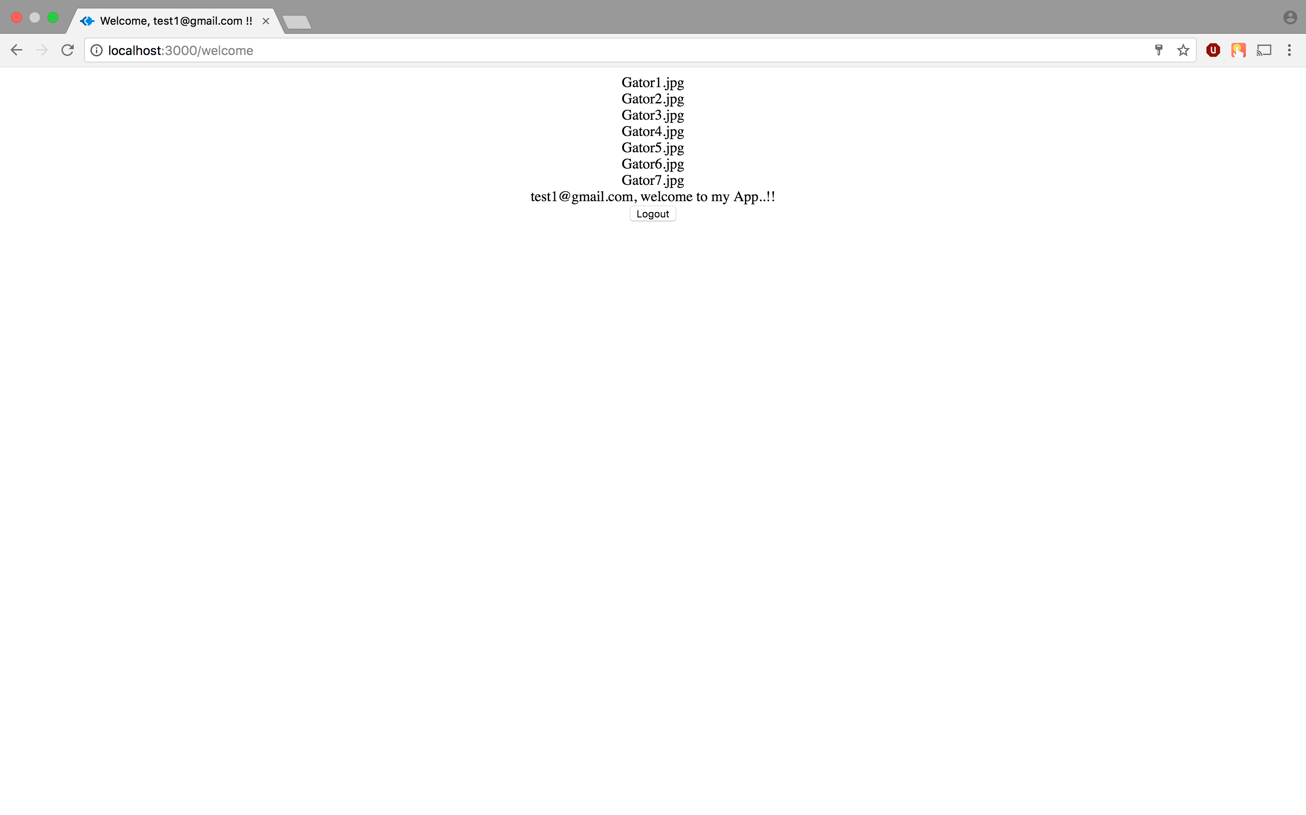


The above image displays the login page for the user.



**Login Page**

The below image shows the list of files uploaded by the user.



**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.**

There are many kinds of encryption algorithms such as symmetric key encryption algorithms, asymmetric key encryption algorithms, cryptographic hash functions. Examples of symmetric key encryption algorithms are the AES suite, where the same key is used to encrypt and decrypt the message. Examples of asymmetric key encryption algorithms are the RSA Suite, PGP etc, where different keys are used to encrypt and decrypt the message. Examples of cryptographic hash functions are the MD suite, the SHA suite etc where the message is hashed to produce a digest which cannot be used to produce the original message. In my application, I used SHA256 to encrypt the password and the reason for using SHA256 is because it provides better security (64 bit) and computational efficiency over other cryptographic hash functions such as MD5 and SHA1.

**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.**

The connection pooling algorithm maintains a pool of mysql connections, where all the requests are handled by reusing these connections. When every connection in the pool is used, the algorithm creates a new connection if the total number of connections is within the prescribed limit.

**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 the process of caching the results for select queries in memory. This eliminates the need to query the database for frequent identical queries and improves the performance of the database server. The most commonly used caching algorithm is the Least Recently Used (LRU) cache algorithm where the cache entries are sorted according to their last access times. If the cache is full, the last access entry is removed to insert the newest query.

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

Session handling in my application is horizontally scalable. The sessions in my application are handled by a npm module called client-sessions. The sessions are stored in the form of cookies in the browser and contain relevant details about the user. These sessions/cookies are stored on the browser devices and are passed with every request to the server. These sessions/cookies also contain an expiry time after which the sessions are recreated.