LAB 1

HUZAIFA AEJAZ

011490453

GITHUB URL: https://github.com/HuzAejaz75

GITHUB PASSWORD: LightSpeed\_75

**Calculator:**

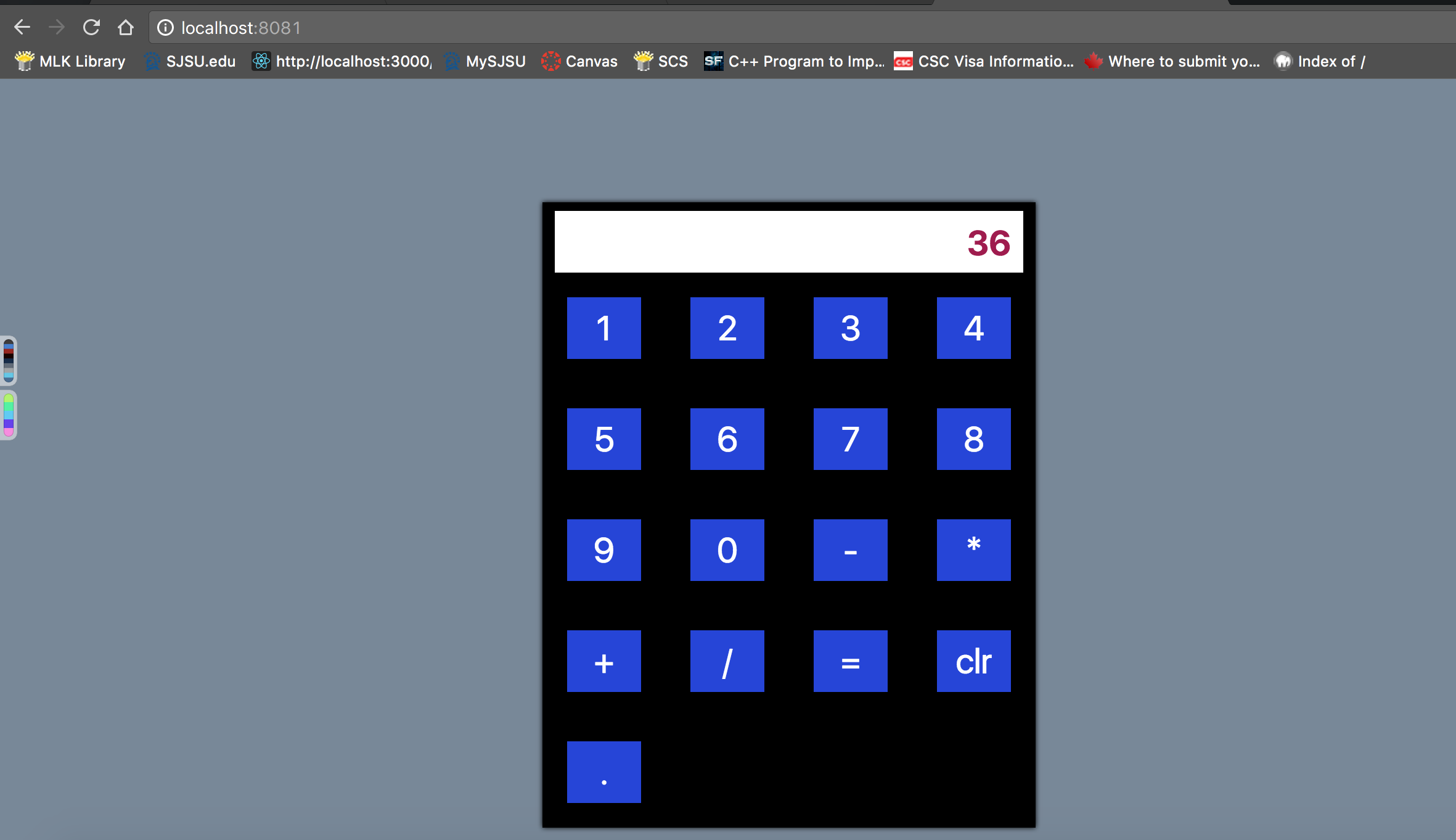
**Introduction:**

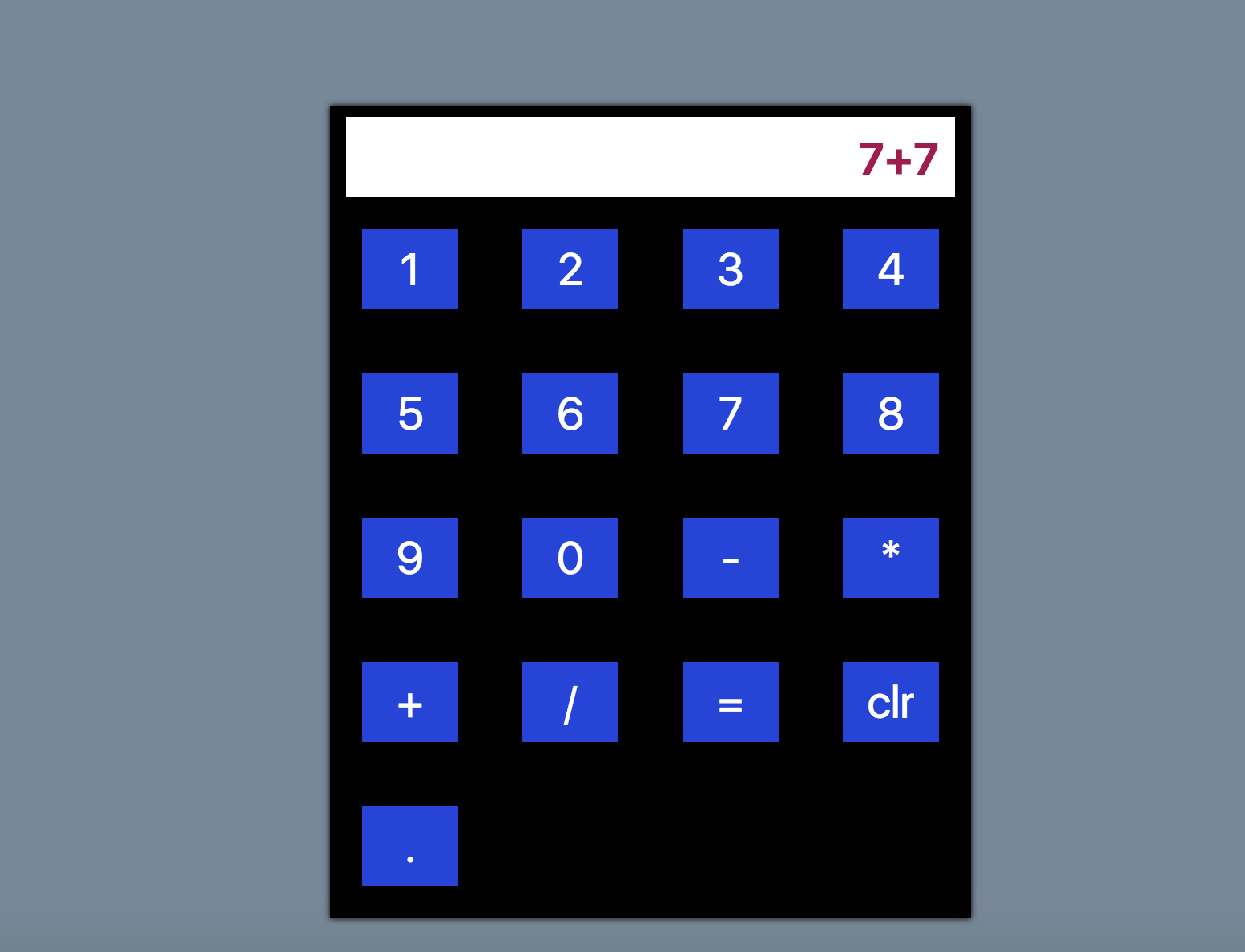
To design a calculator and dropbox using react and redux. Using mysql database as backend.

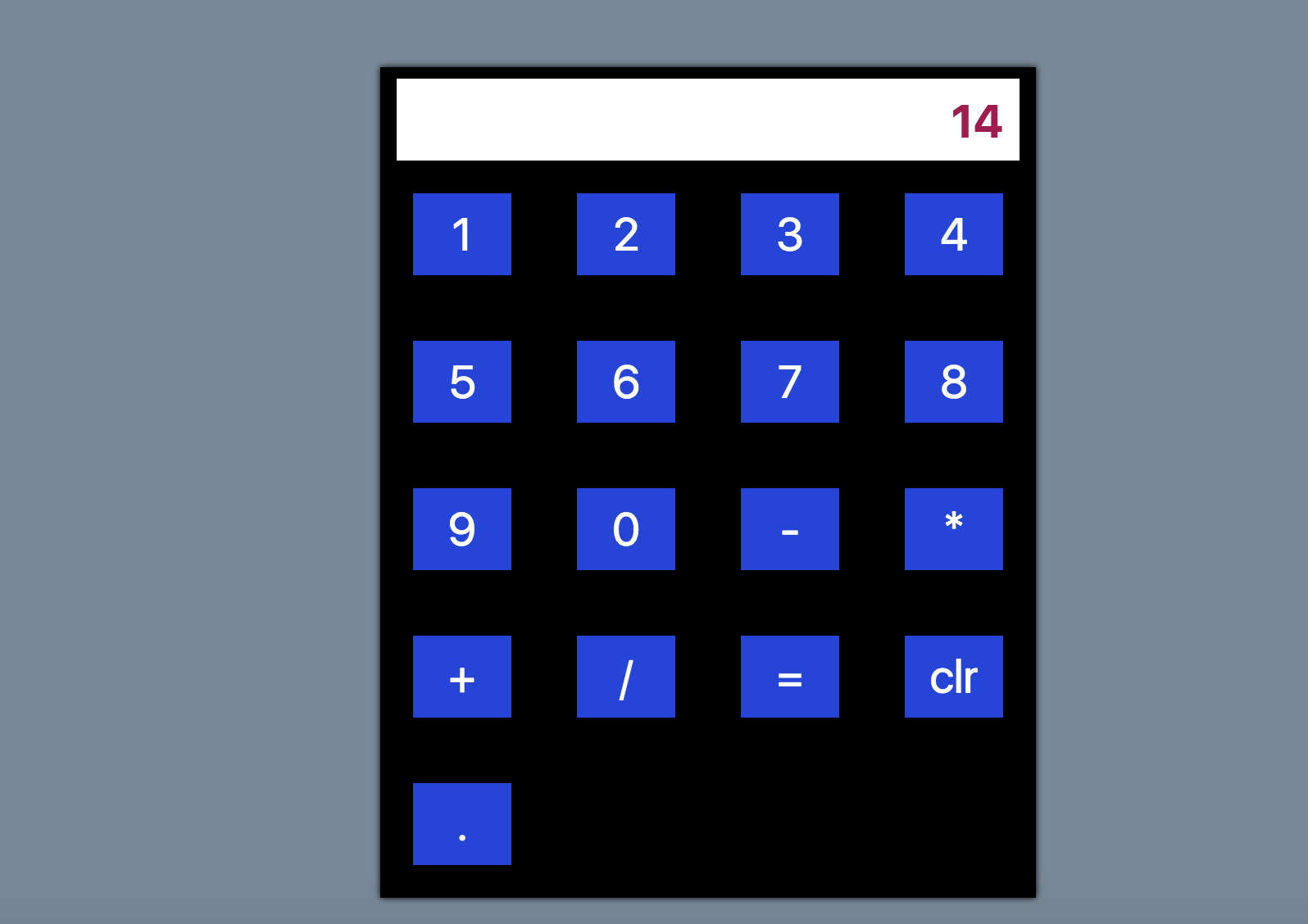
**System Design:**

The system is split into client and server, client is based on react and redux with jsx/typescript as the templating language for dropbox/calculator respectively. The actions file dispatches the action which is captured by the reducer which updates the state which is displayed in the client.

**Results:**

****

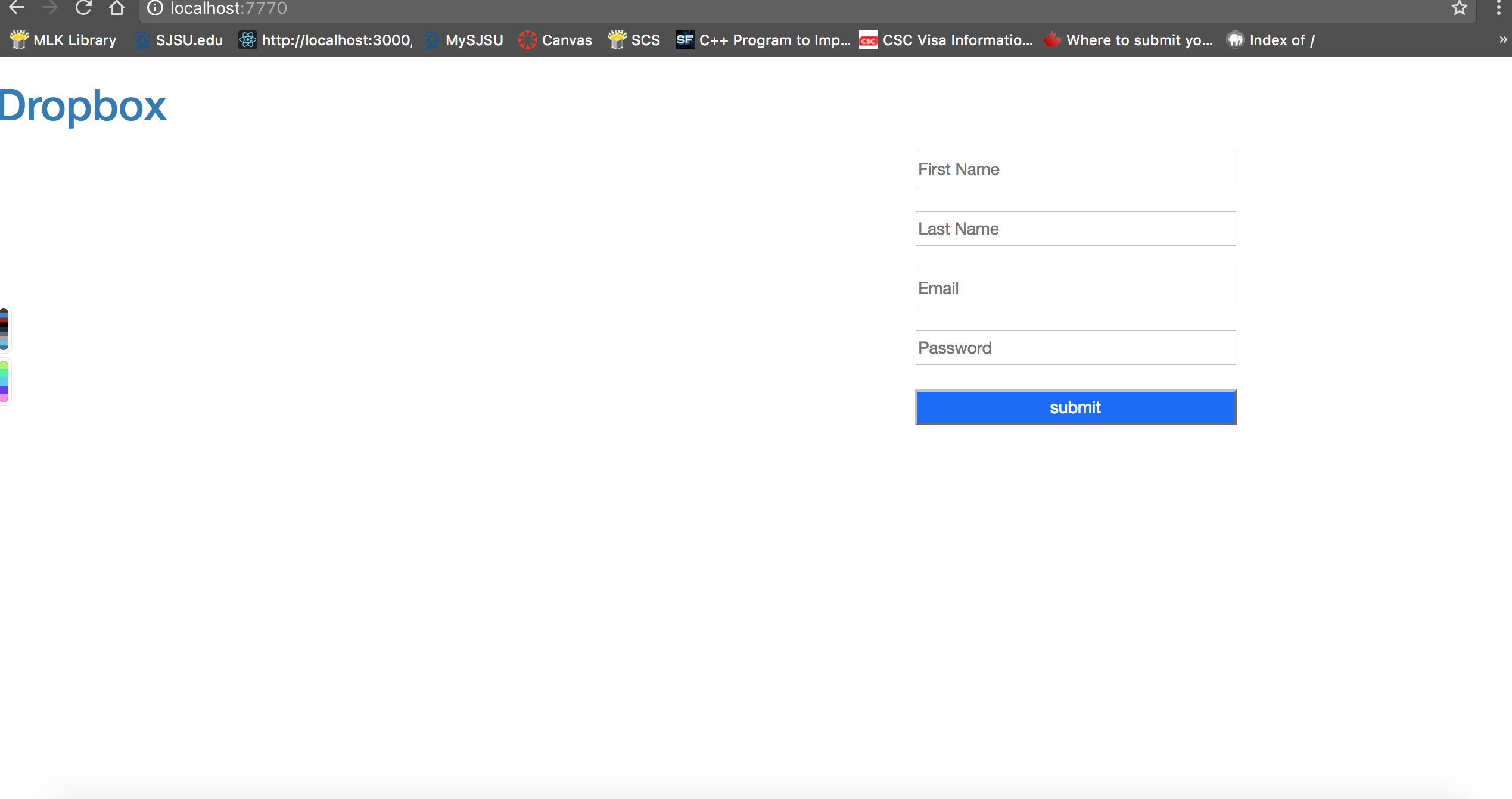
****

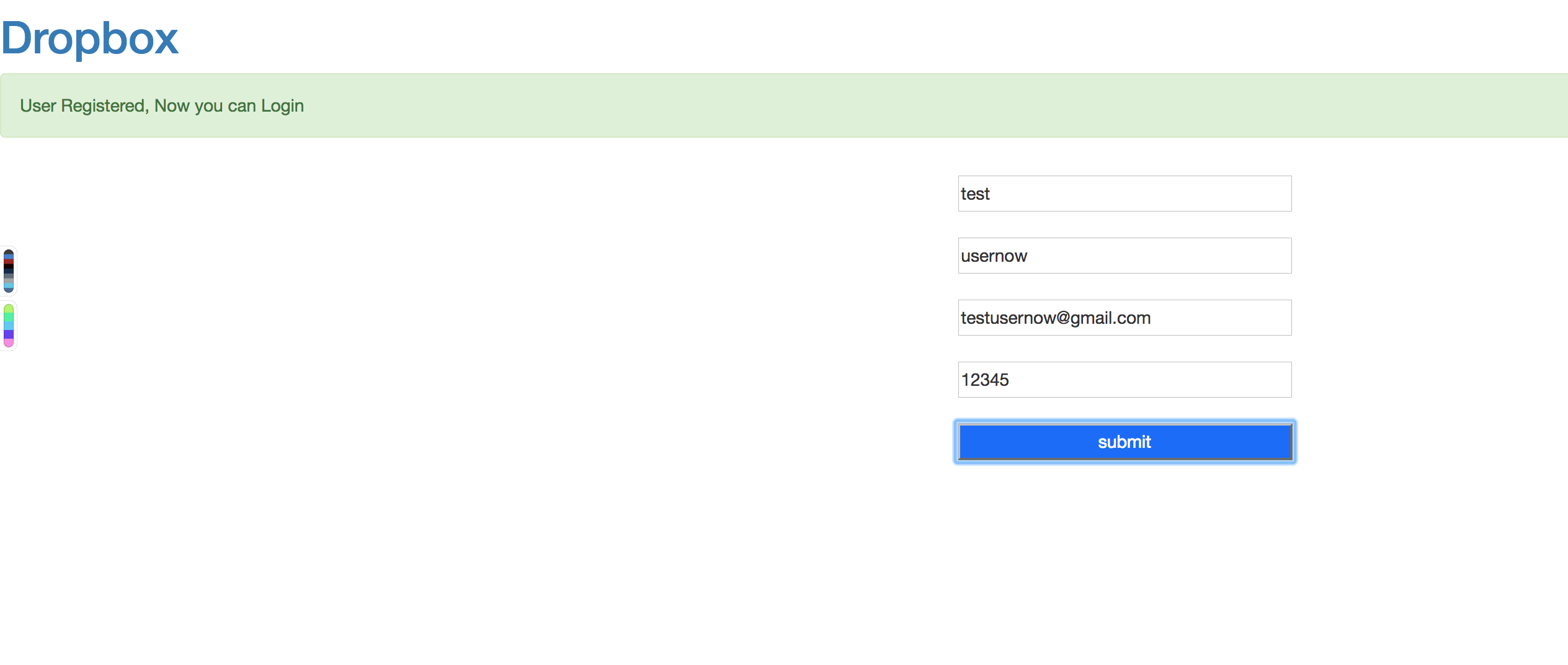
****

DROPBOX:

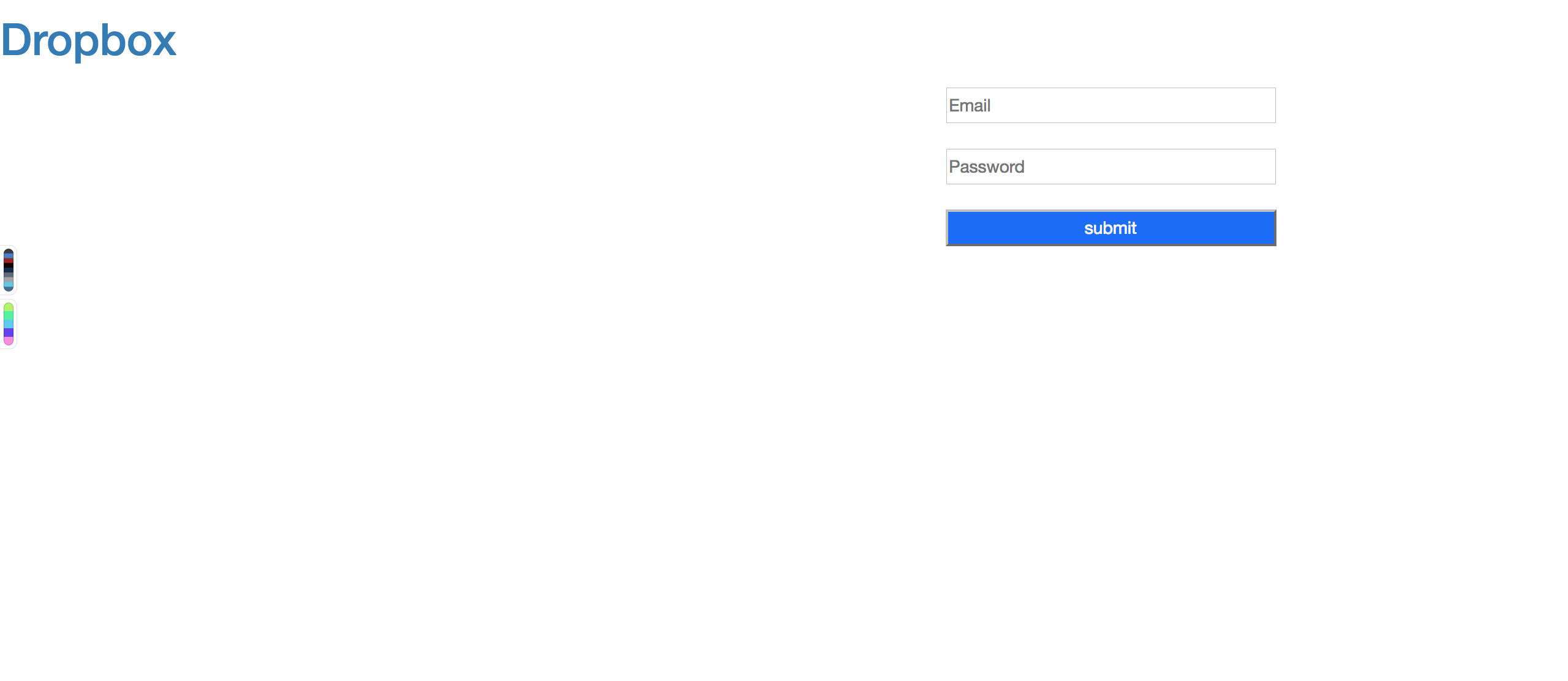
Results:

Register page:

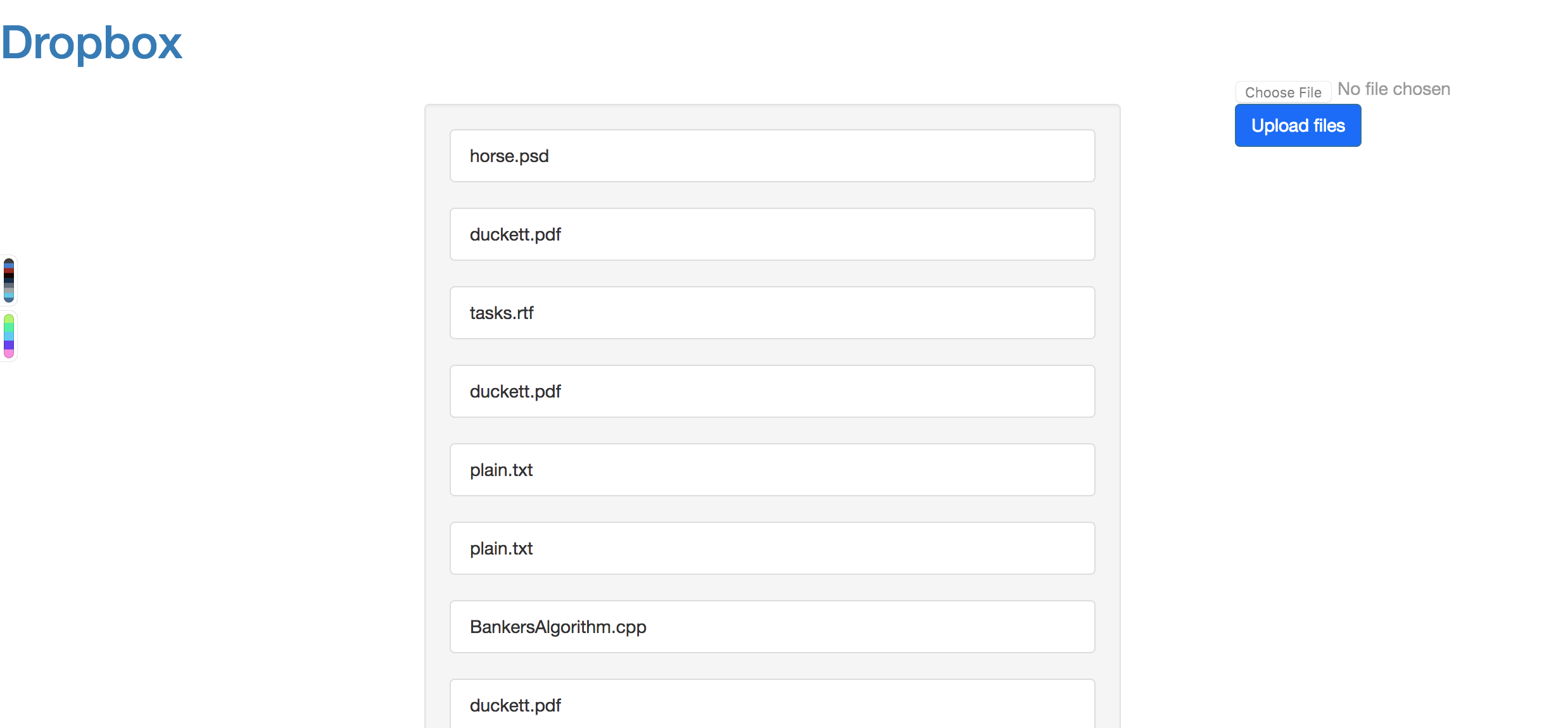




Login:



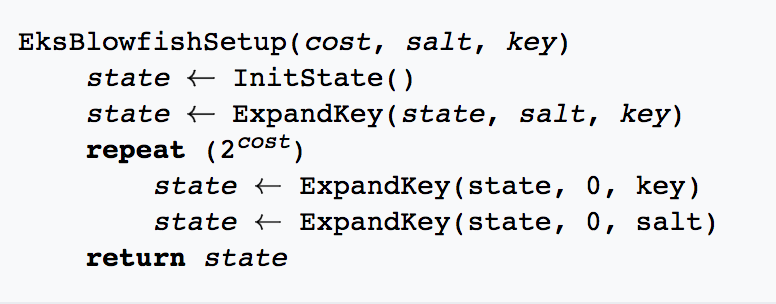
Filedashboard



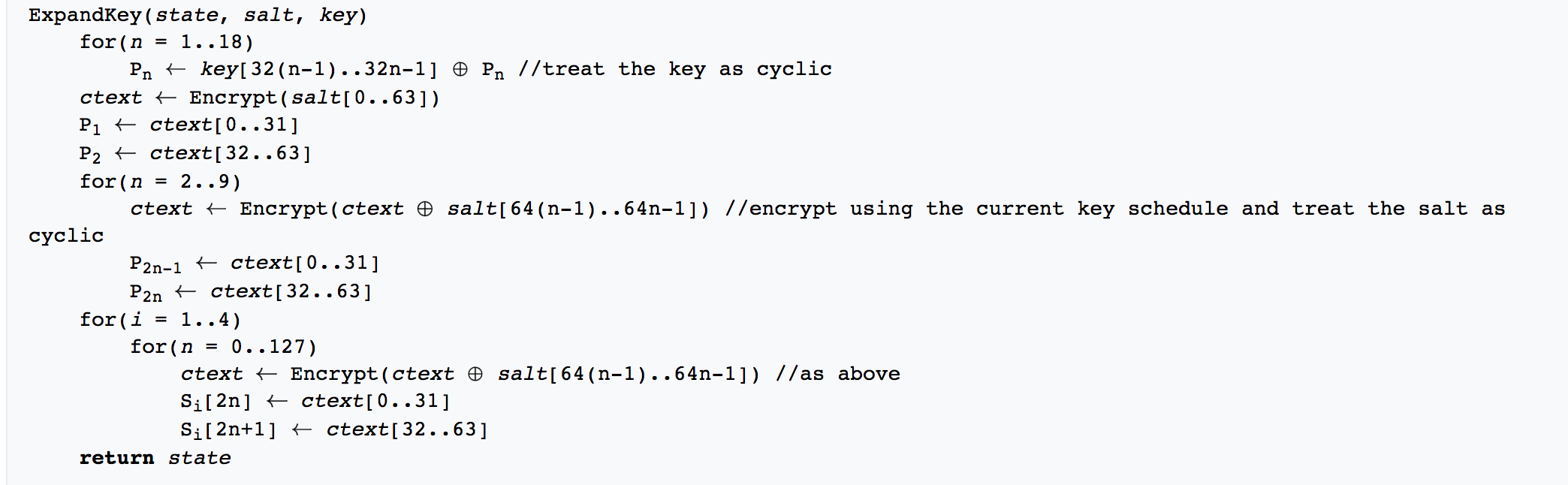
Explain the encryption algorithm used in your application. Mention different encryption

algorithms available and the reason for your selection of the algorithm used.

I have used Bcrypt.js for the encryption algorithm.



The expand key function is as shown below:



Bcrypt.js runs on “Eksblowfish" key setup algorithm. The initState works on original blowfish algorithm, populating the p-array and S-box entries with the Pi in hexadecimal.

The chief feature of Bcrypt.js is that it is run two times with the same plain text secret. This is because of the use of salt and hence even if two users have same password they will have a different hash.

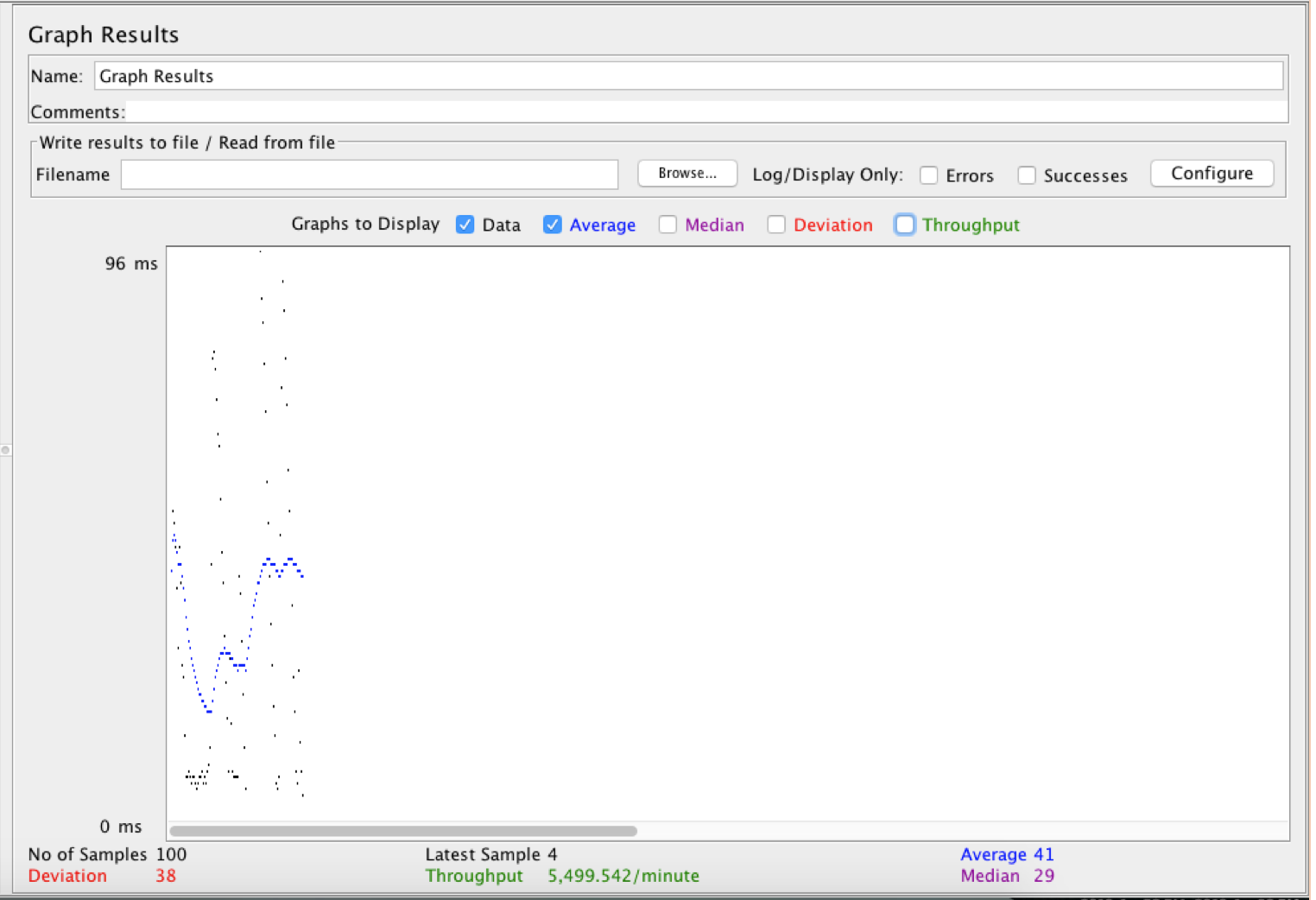
Compare the results of graphs with and w

ithout connection pooling of database. Explain

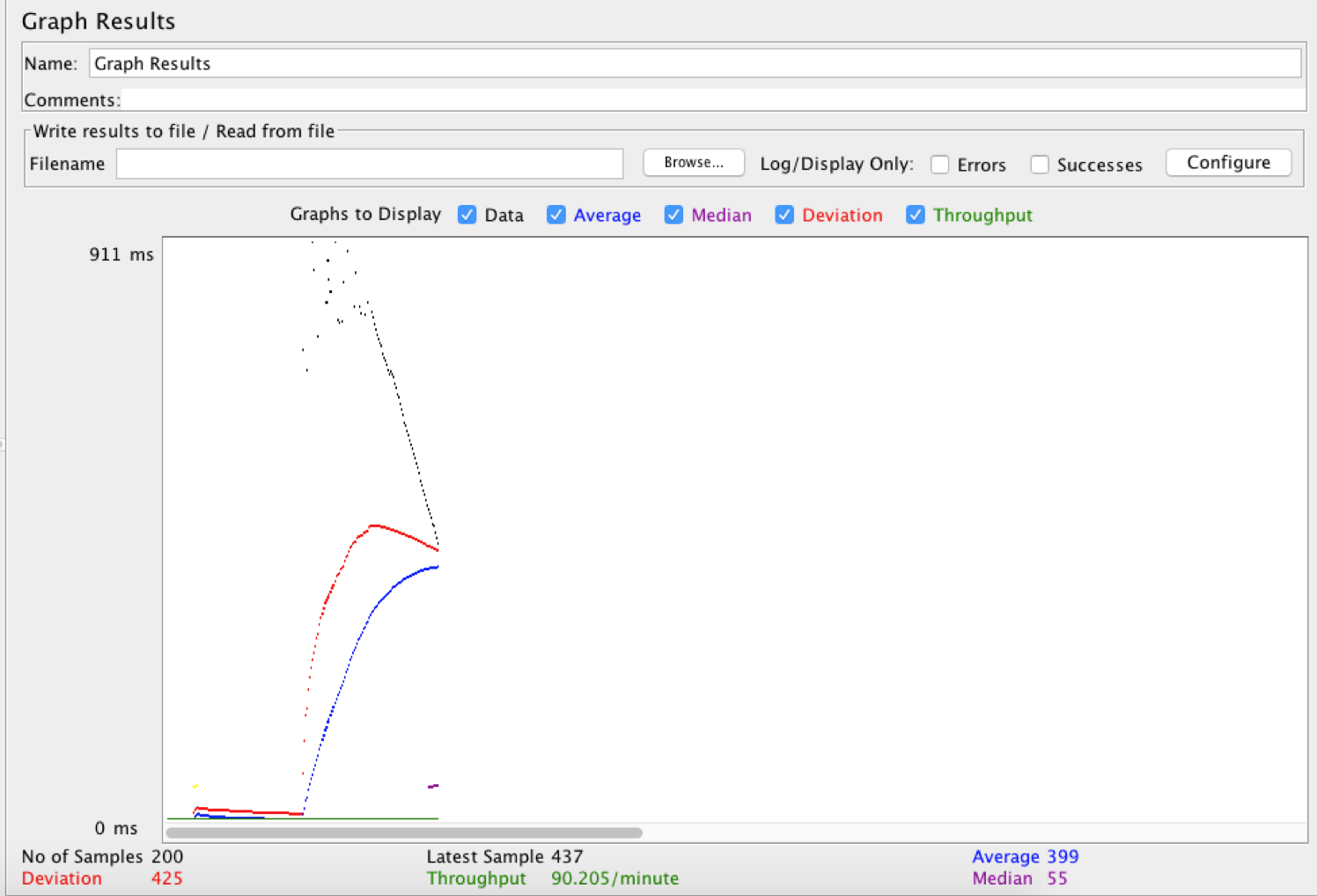
the result in detail and describe the connection pooling algorithm used in your

code.

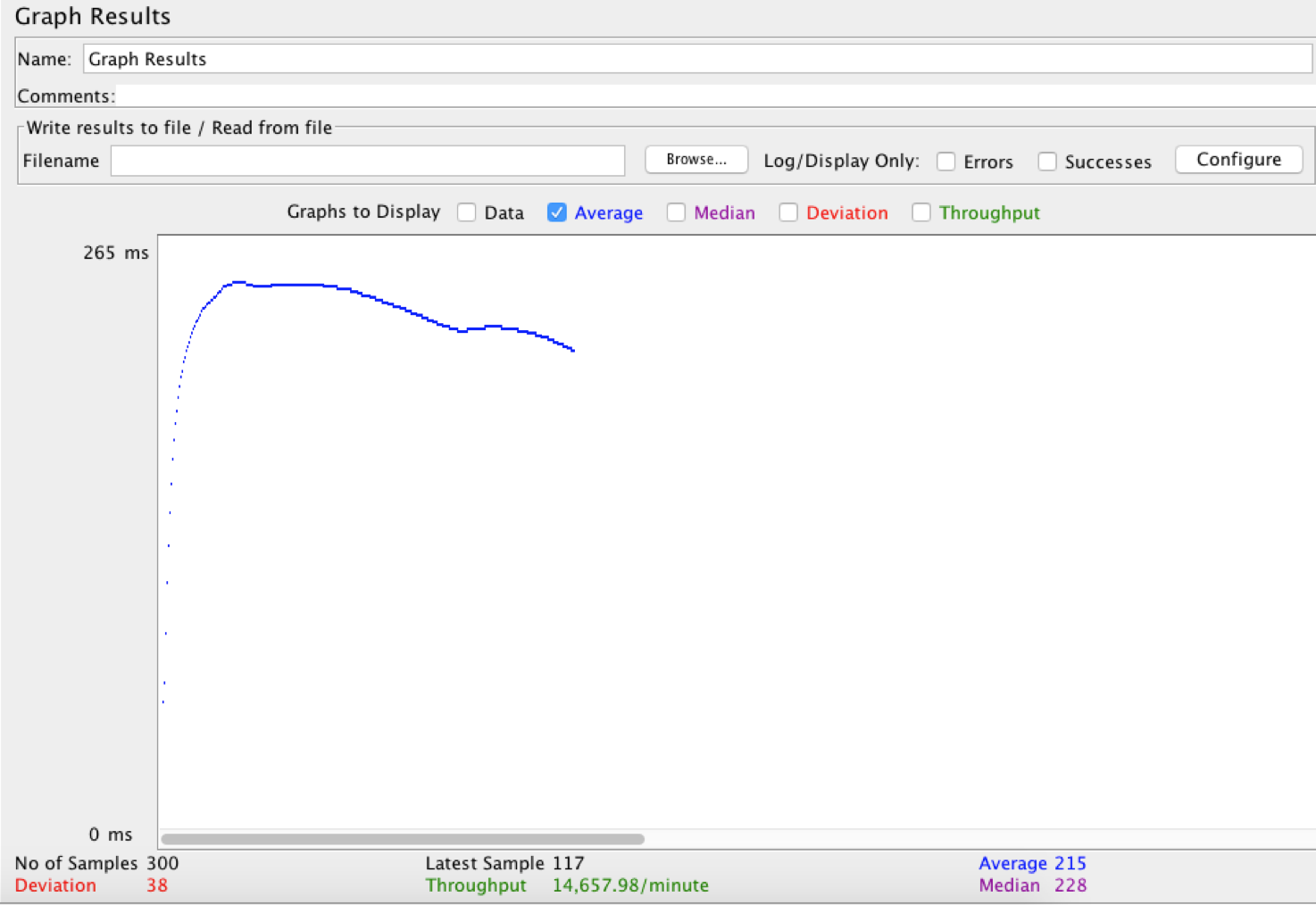
1. Without connection pooling : 100 users



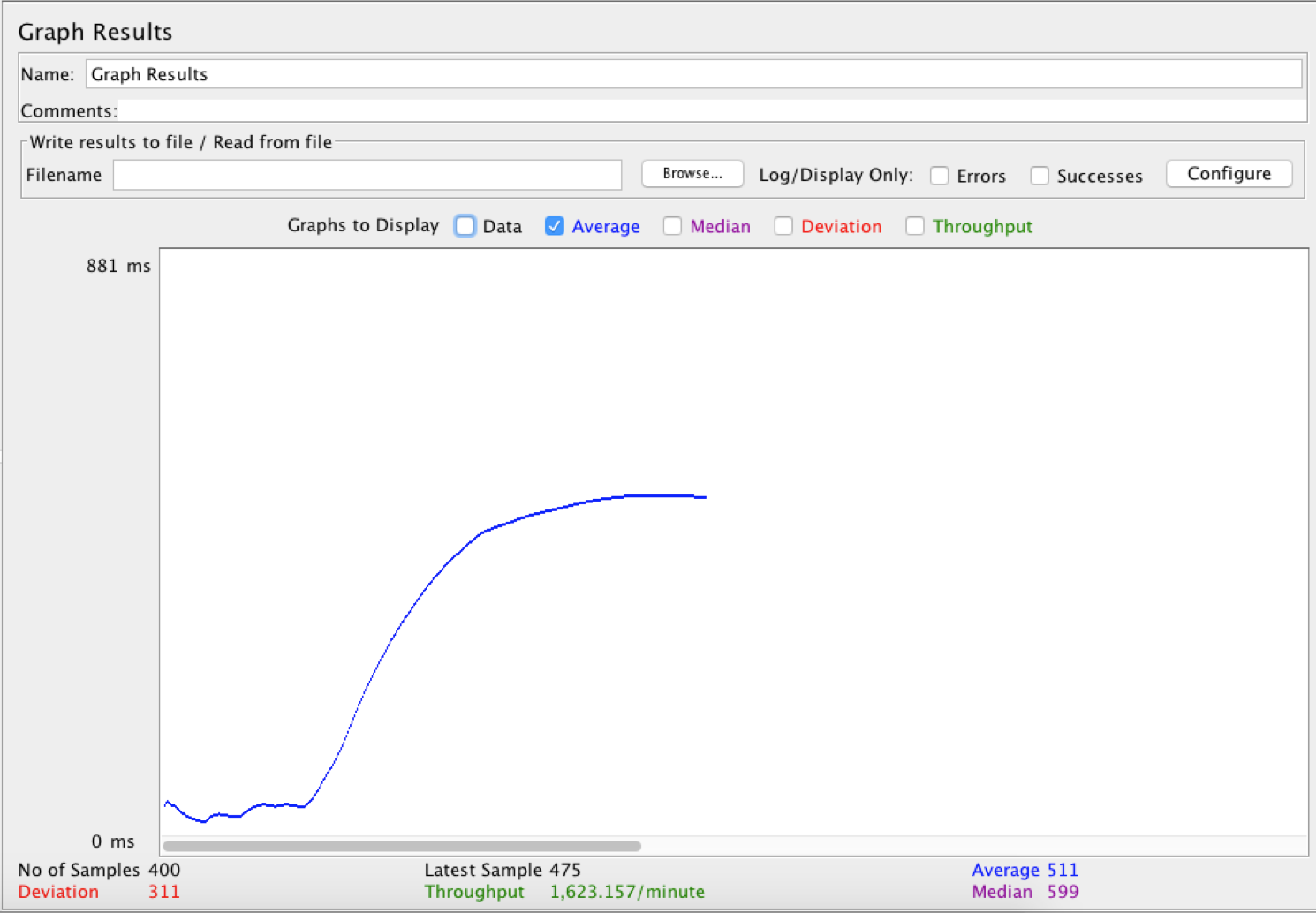
1. Without connection pooling :200 users



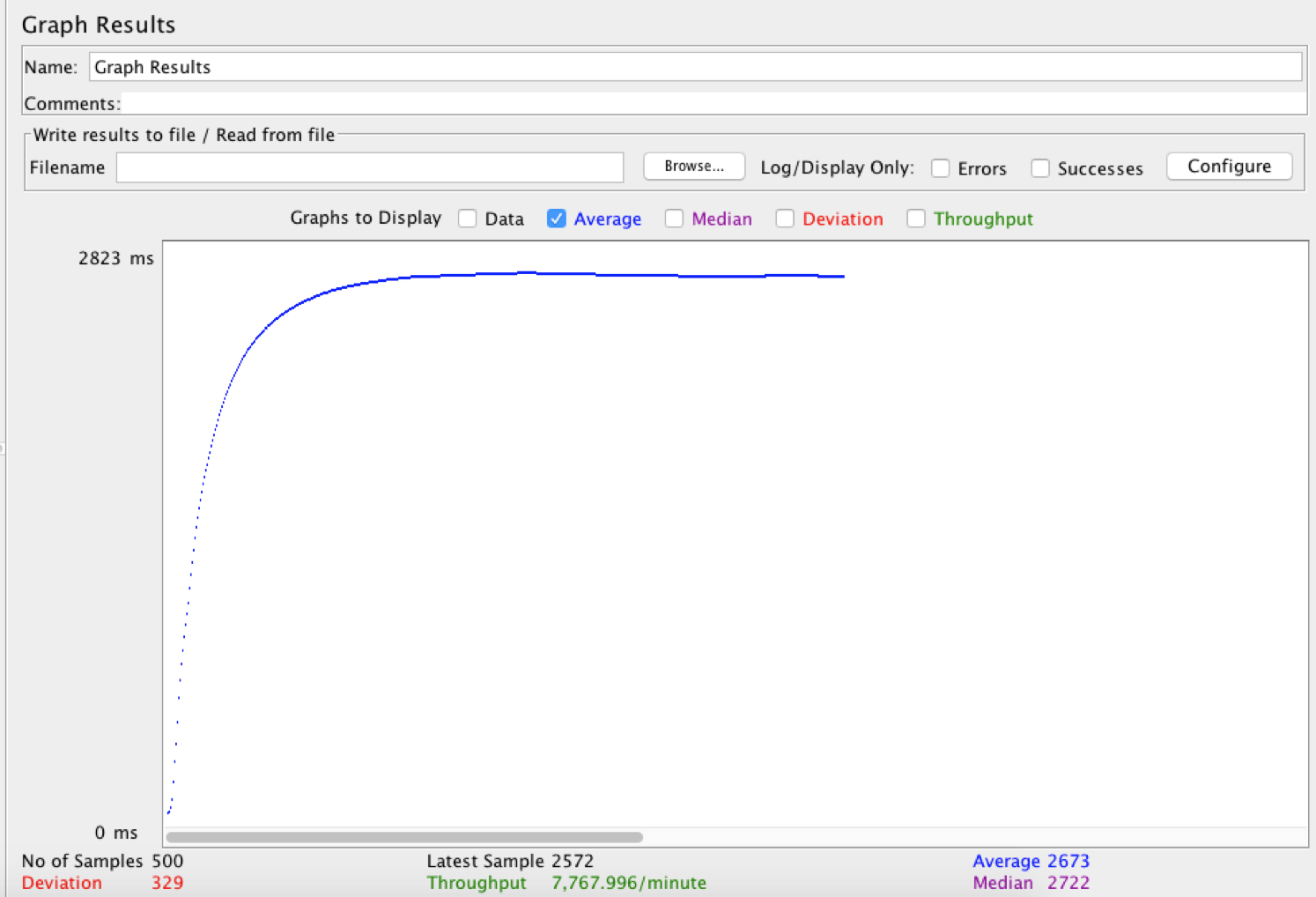
1. Without connection pooling: 300 users:



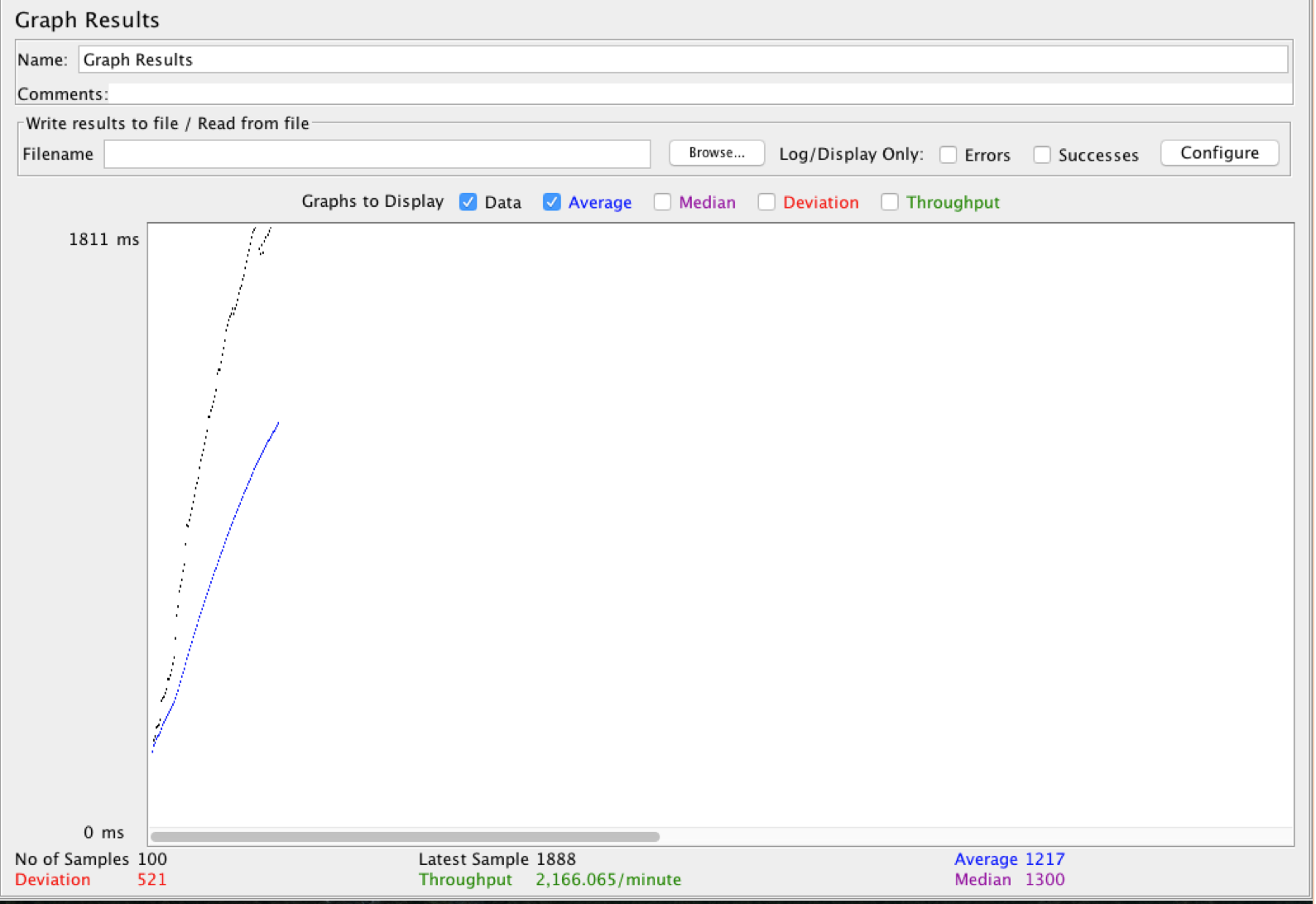
without connection pooling : 400 users



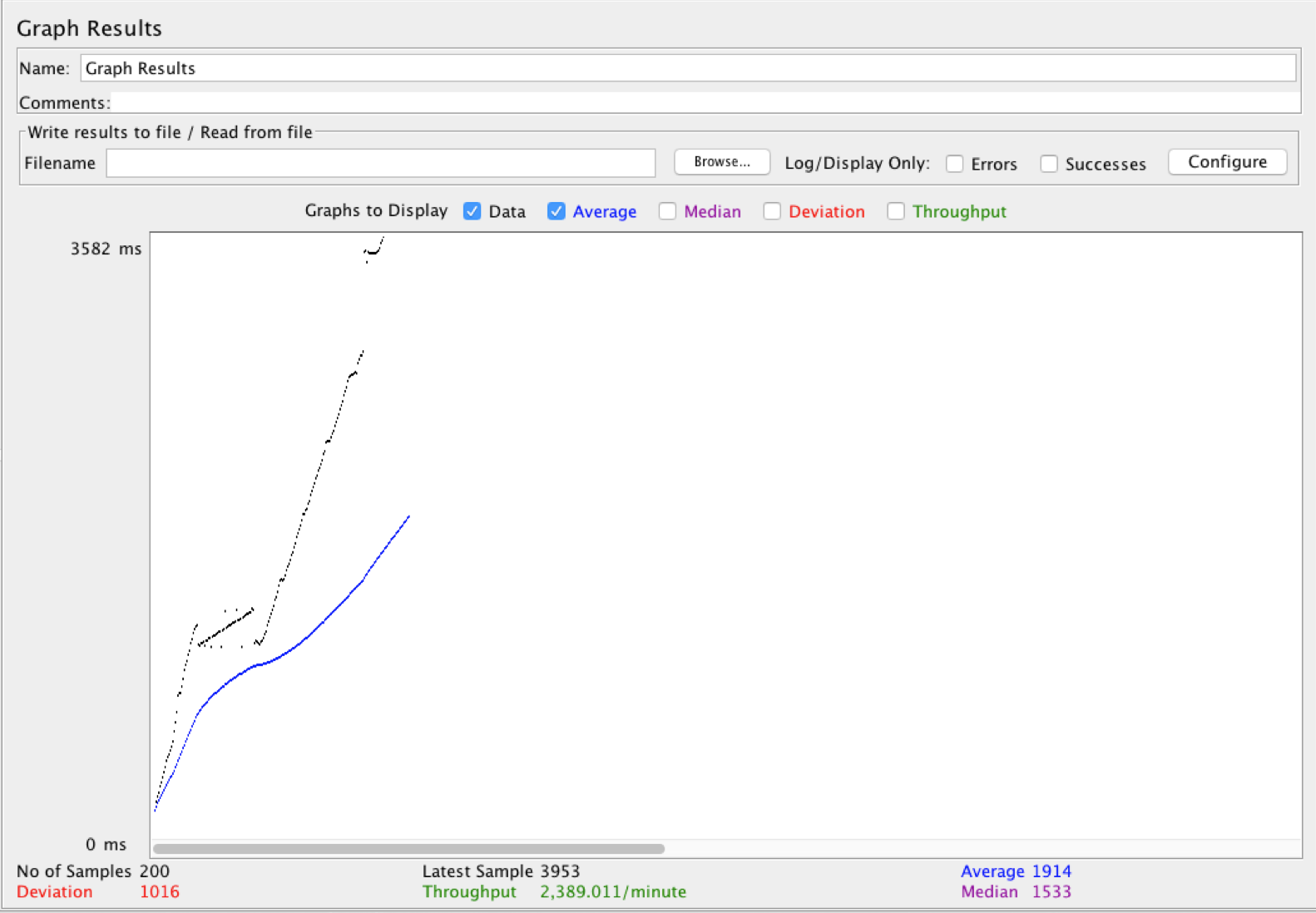
Without connection pooling : 500 users



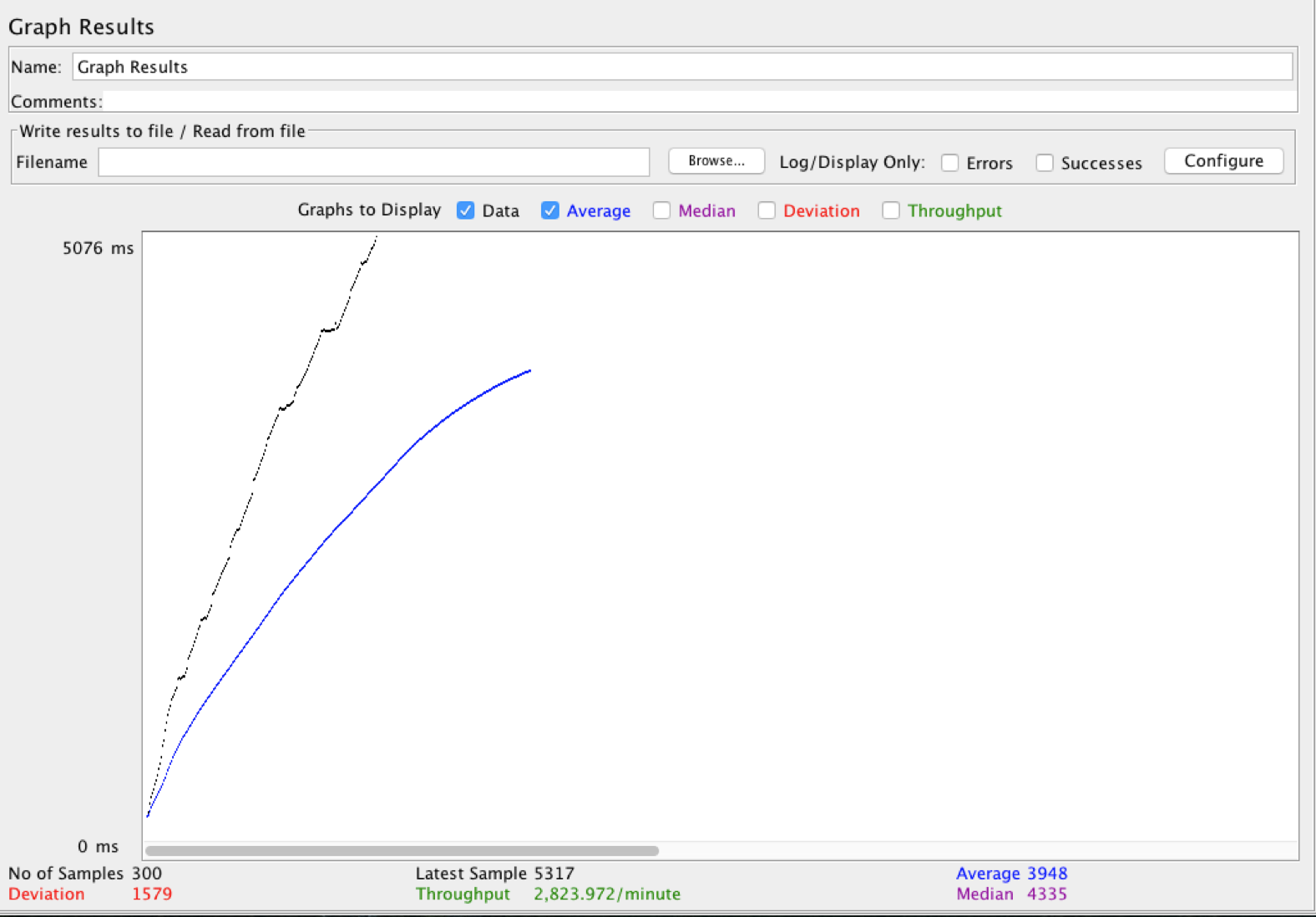
**With connection pooling – 100 users:**



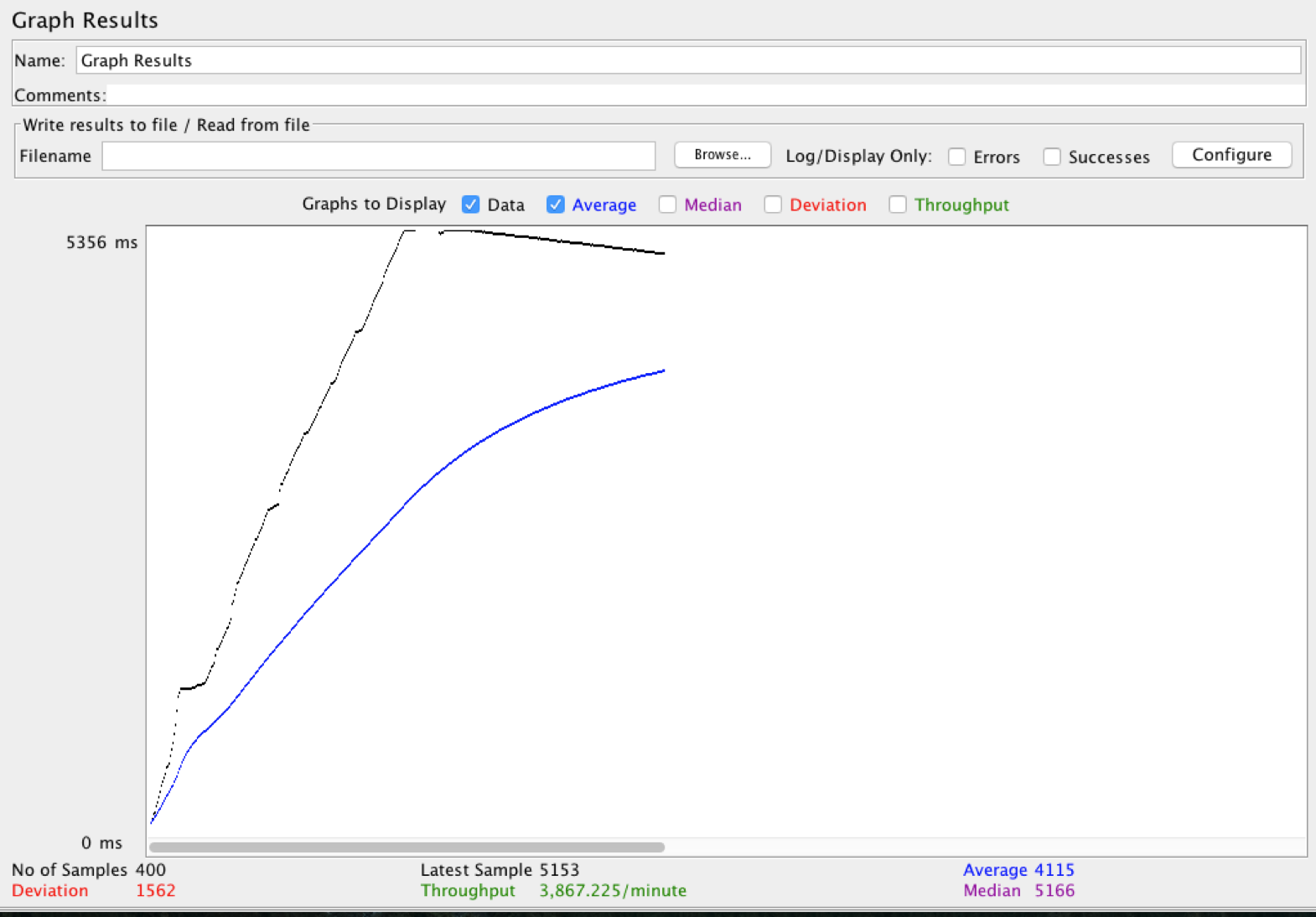
With connection pooling: 200 users



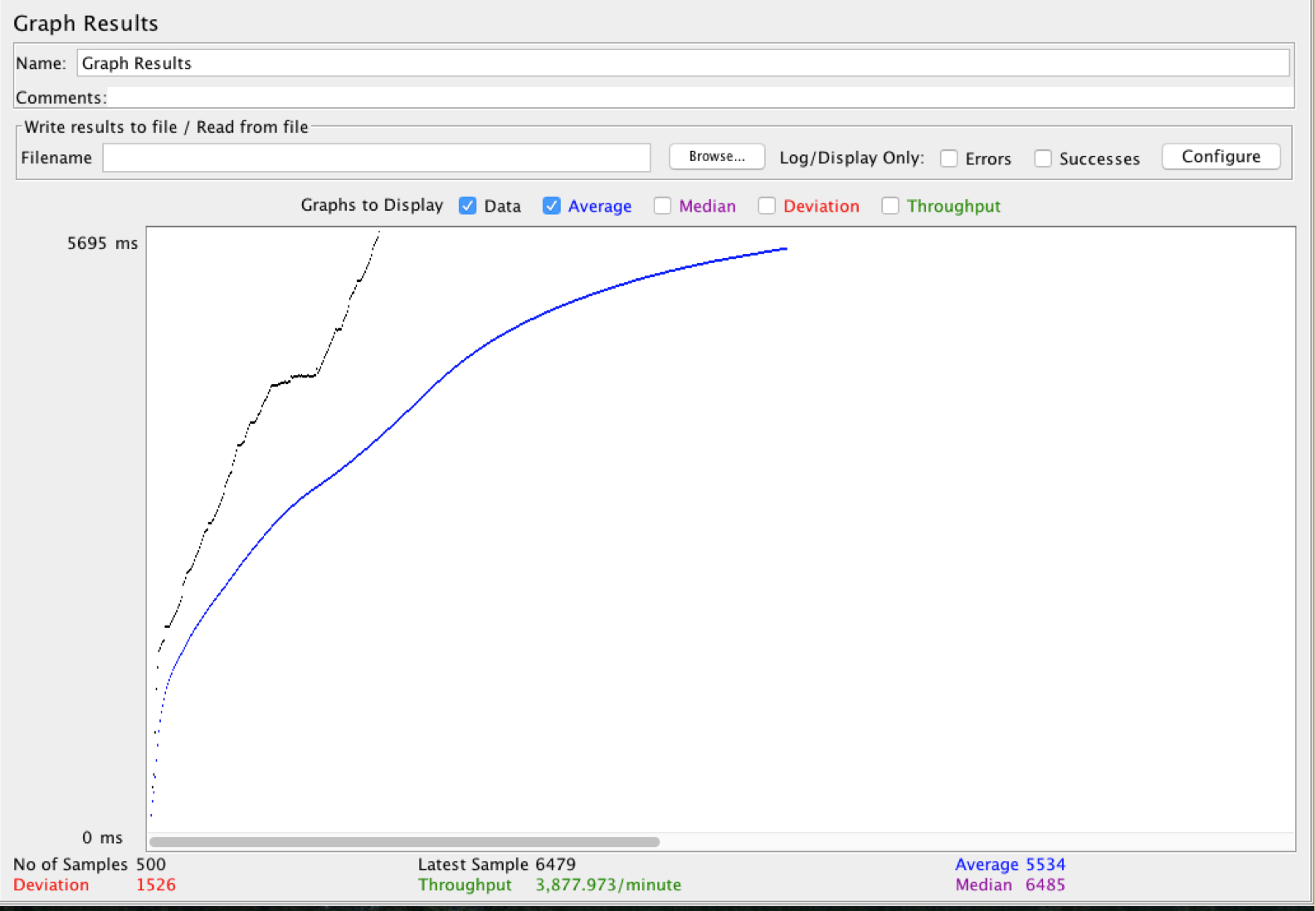
With connection pooling : 300 users:



With connection pooling: 400 users:



**With connection pooling – 500 users**



From the graphs, we can see that connection pooling gives better performance compared to without using connection pooling. I have used 50 connections and use the mysql.createPool function to set them.

1. **How would you implement Request Caching? Explain in detail. No need to implement a function – use pseudo code or detailed explaination?**

In distributed systems when calls are made to servers to fetch data, it can lead to performance issues, as the requests get called every time and have to move to and fro from the clint and this leads to delay. With request caching, we save time and delays by saving every request that is made to the server and reusing them as is needed, this leads to significant improvement in performance. Here we first check the

new Request()

if !(any new changes to the database ) {

the cached request needs to be scanned{

if new request == cached Request{

if (cacheRequest.time> systime() – x){

response

}else{

response = request.send()

cacheRequest.timestamp = systime()

return response

}{

}else{ //a new cache is created here

cache Request = new chached\_request

response = request.send()

cacheRequest.timestamp = systemtime()

return response

}

}

Is your session strategy horizontally scalable? If

No, it is not horizontally scalable as I am using local-storage npm package to check the user credentials and JWT token and not using session manager.