CMPE 273-Lab 2

Dropbox

By

ADITYA PARMAR

Student ID – 011819964

**Introduction:**

* To developed Data storage like Dropbox where User can save their data and share with others.
* Use Kafka for fault tolerance and high throughput. Use PassportJS for security and persistent session.

**System Design:**

**Client –**

* Client is developed in React-Redux as JavaScript framework and HTML,CSS and Bootstrap for attractive UI.
* It interacts with Users and HTTP requests to sever as per user’s requirements.
* React – Redux wonderfully update components.

**Server –**

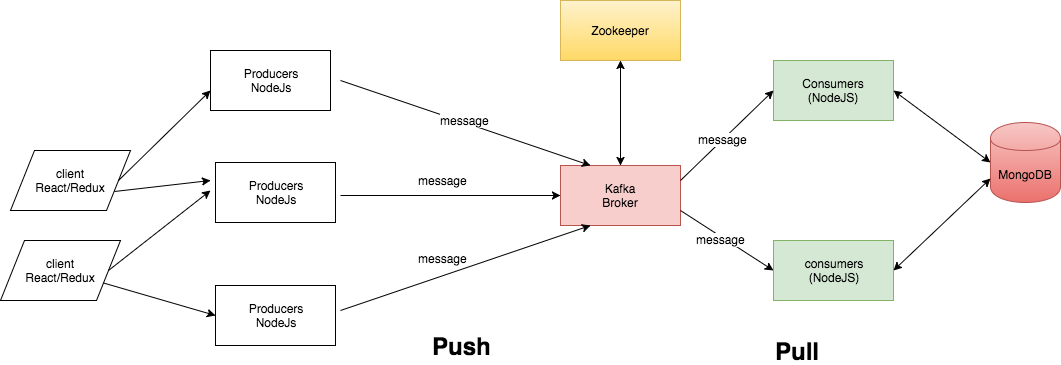
* The “Dropbox” application server handles the requests sent by the browser i.e. Client to the server.
* The server here consists of two components – Kafka and Node.js. The requests sent

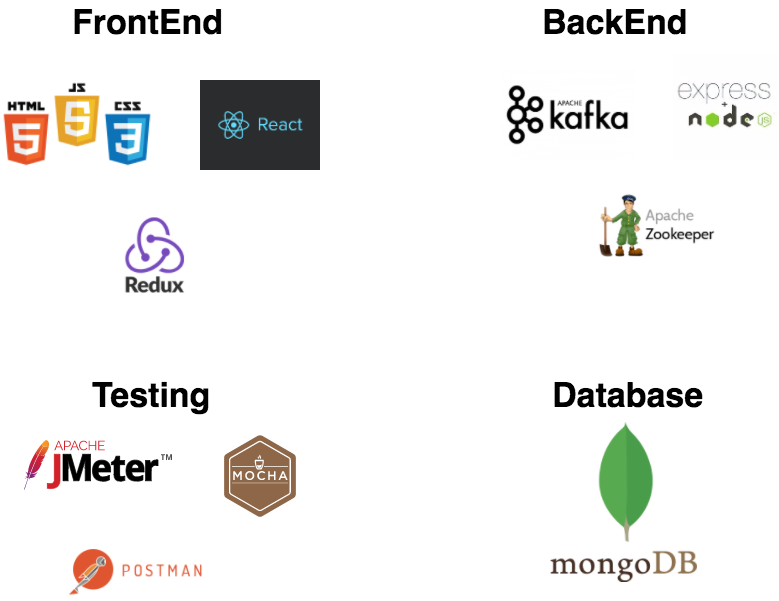
by the Client is received by Node server. The Functionality of node server is to send those requests to Kafka in form of messaging queue to process the received requests.

* The requests are sent to Kafka in form of topics along with the message. The task of Kafka is to process those requests sent by the Node Backend.
* Kafka server interacts with the Database i.e. Mongo DB in these case in order to process the requests like insertion, updating, delete.
* Now, after Kafka process the request, it becomes the producer to send the response back to Node backend. In these case, Kafka and Node backend becomes the producer and consumer simultaneously.
* After the requests are received by the Node backend, it sends the required response back to Client in JSON.
* This is how the whole architecture is created and implemented wherein Kafka acts

as a middleware between the requests and processes it in form of messaging queues.

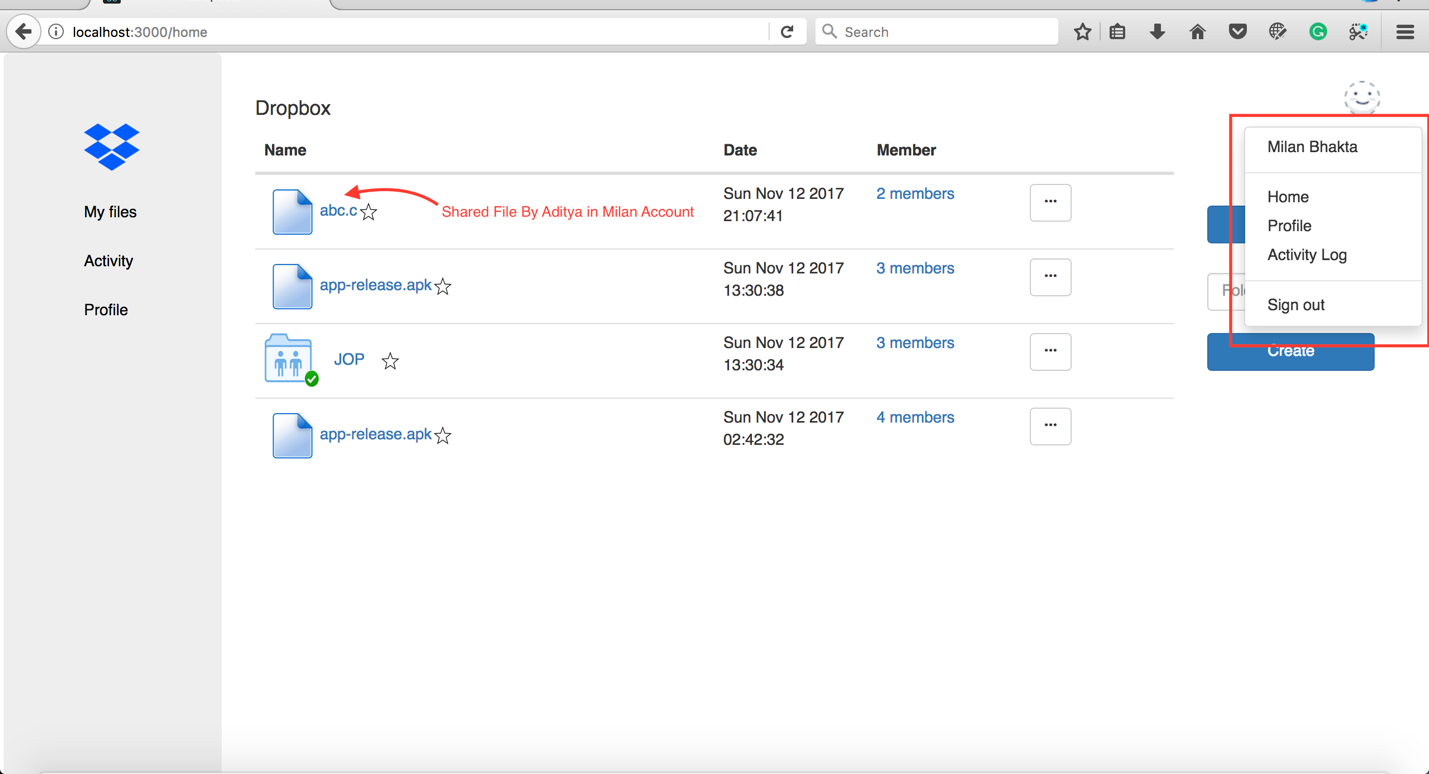
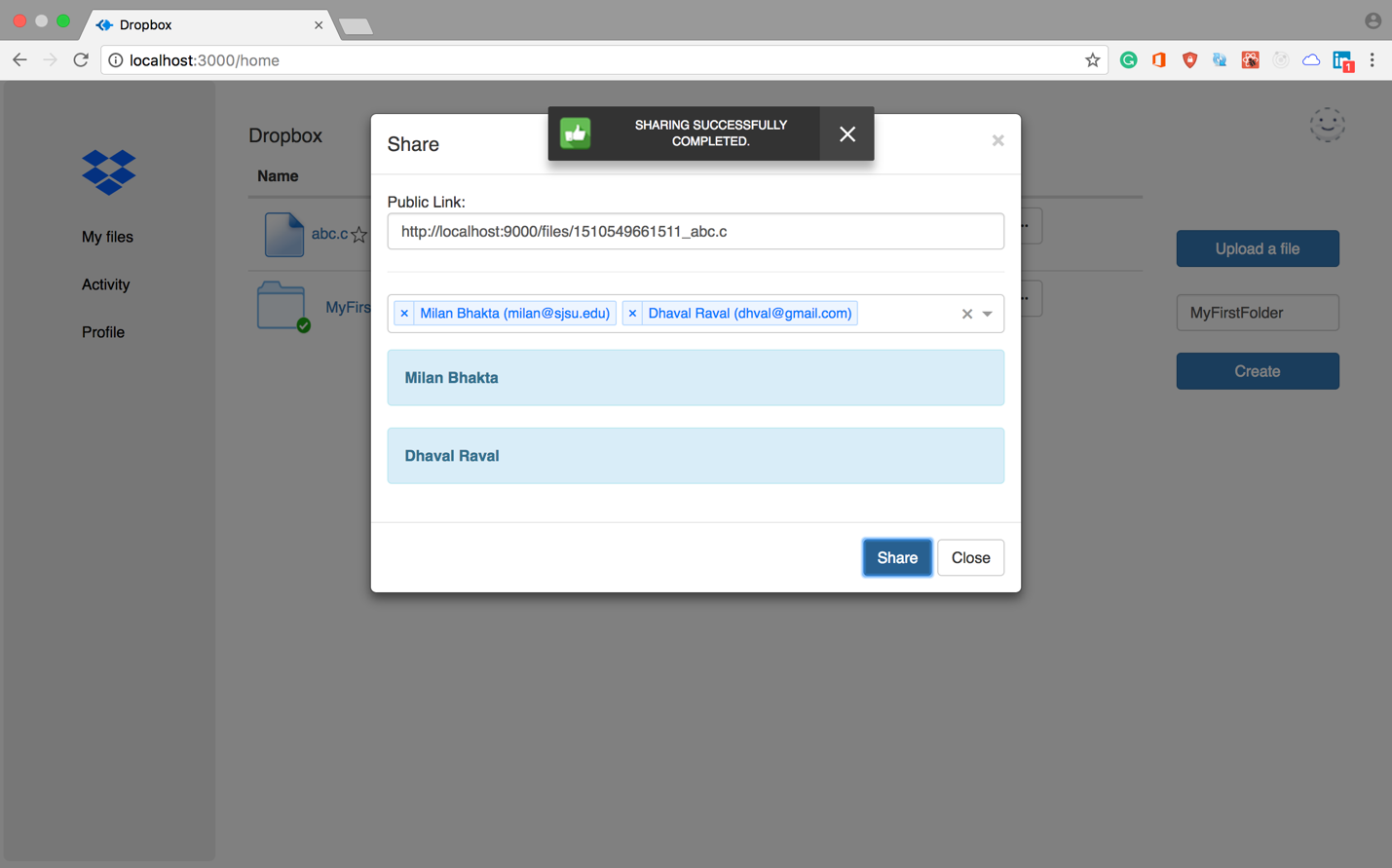
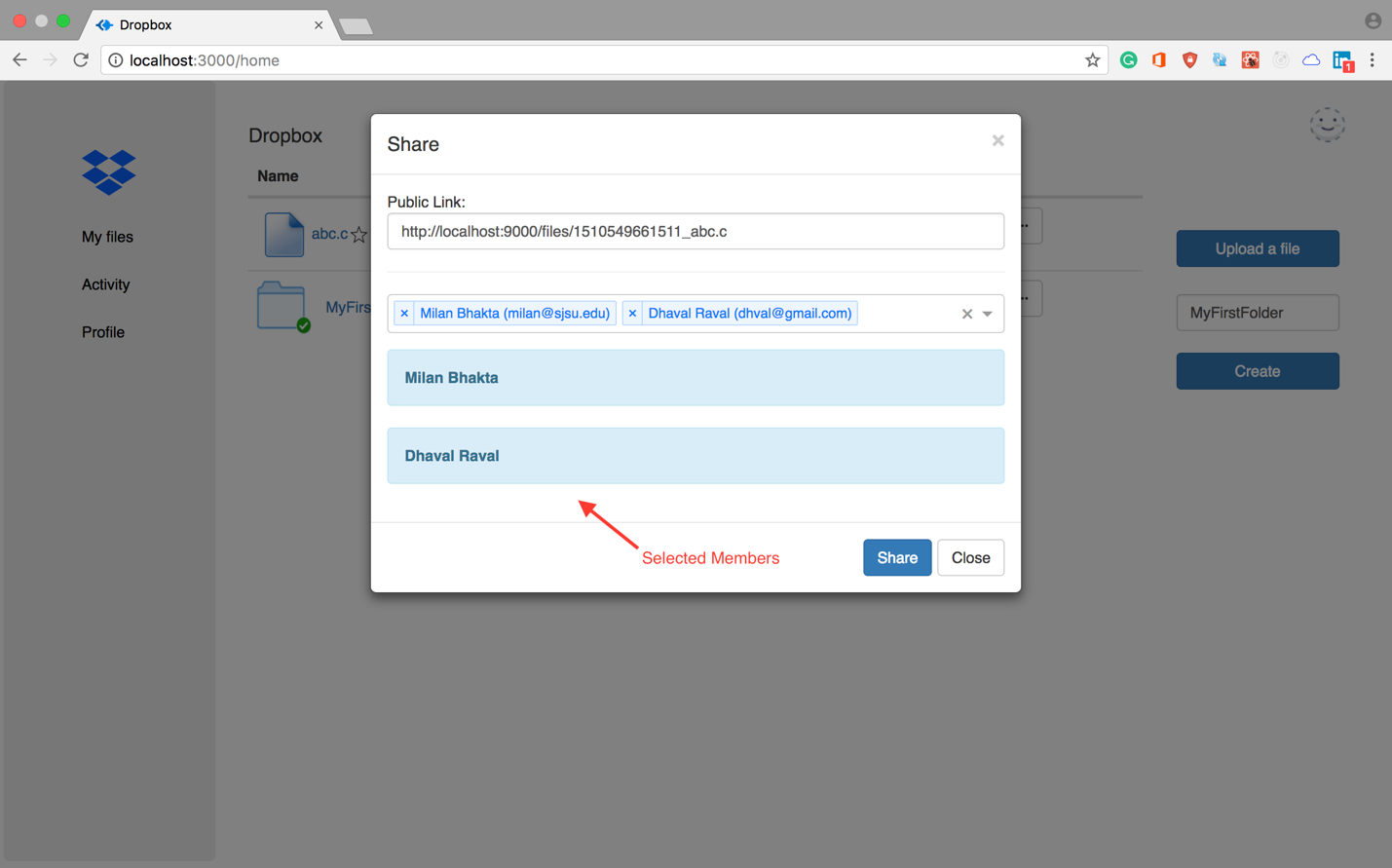
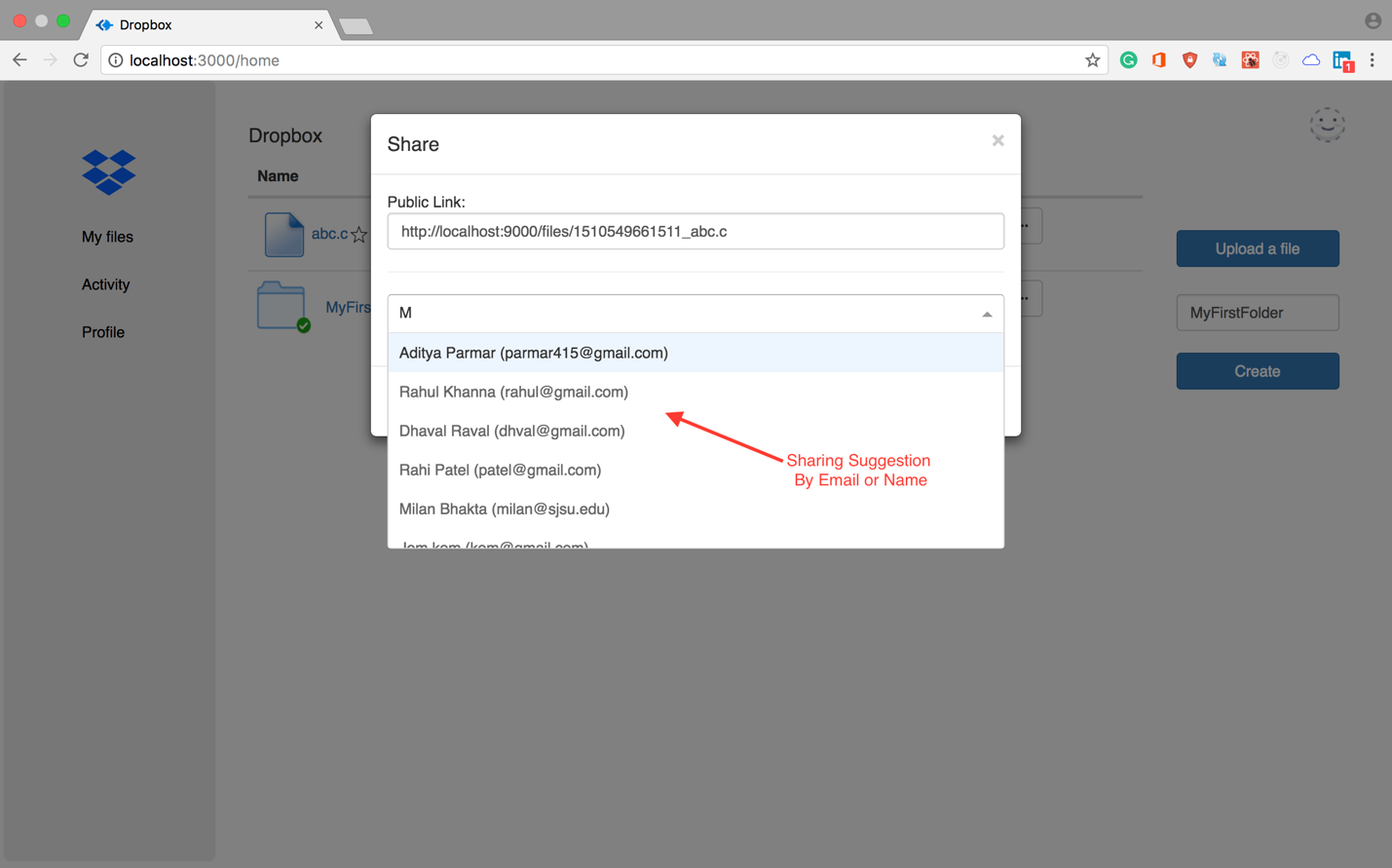
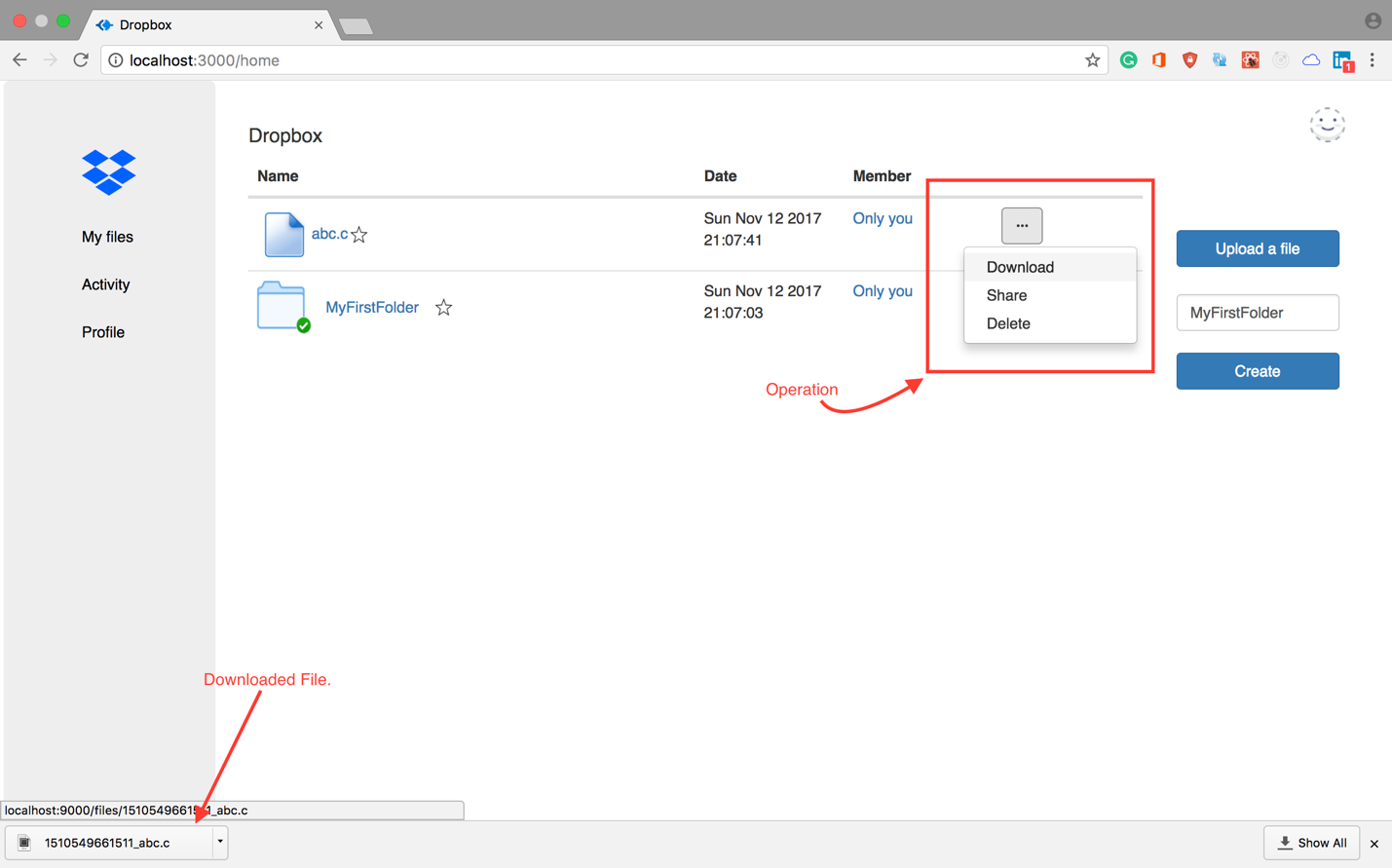
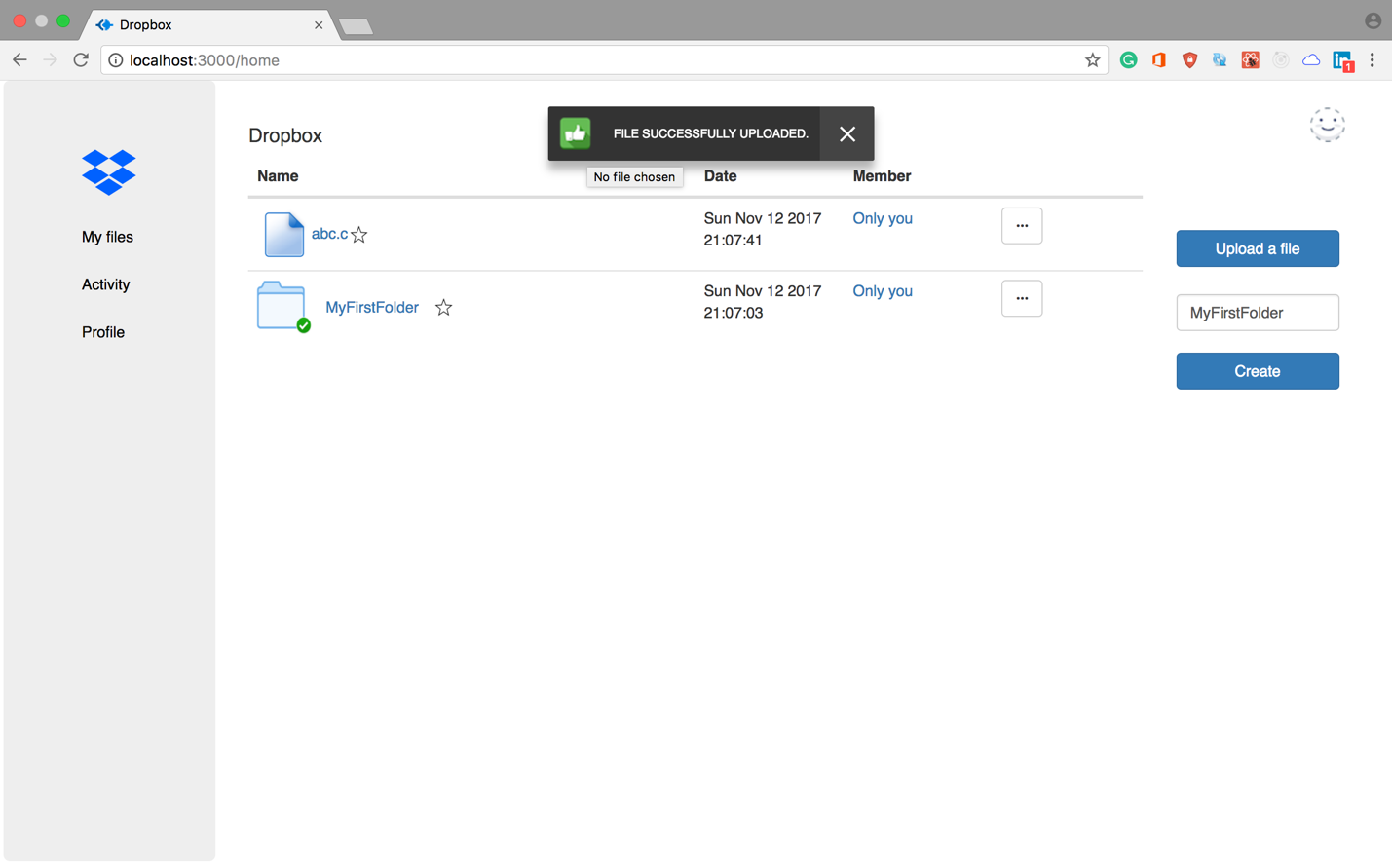
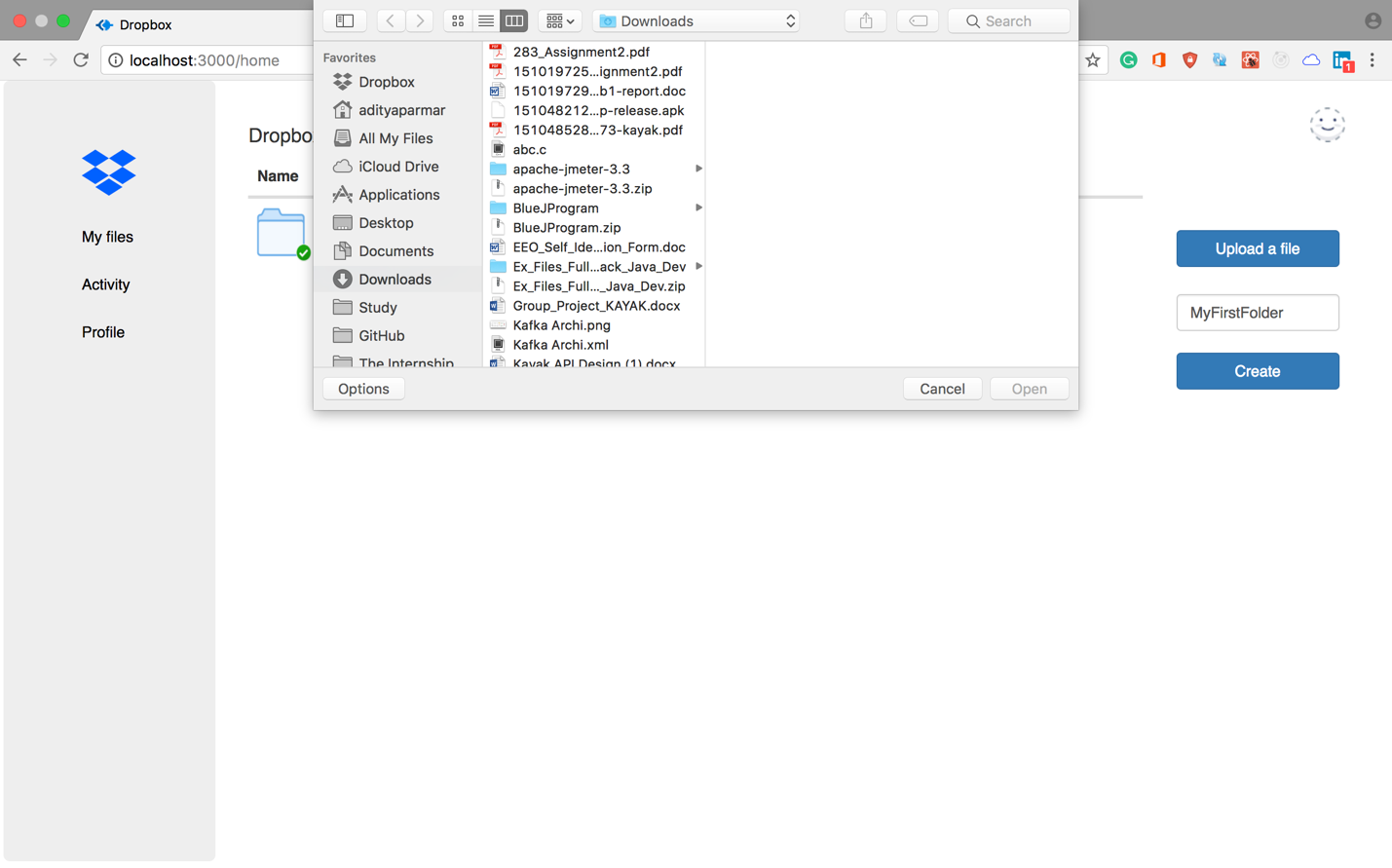
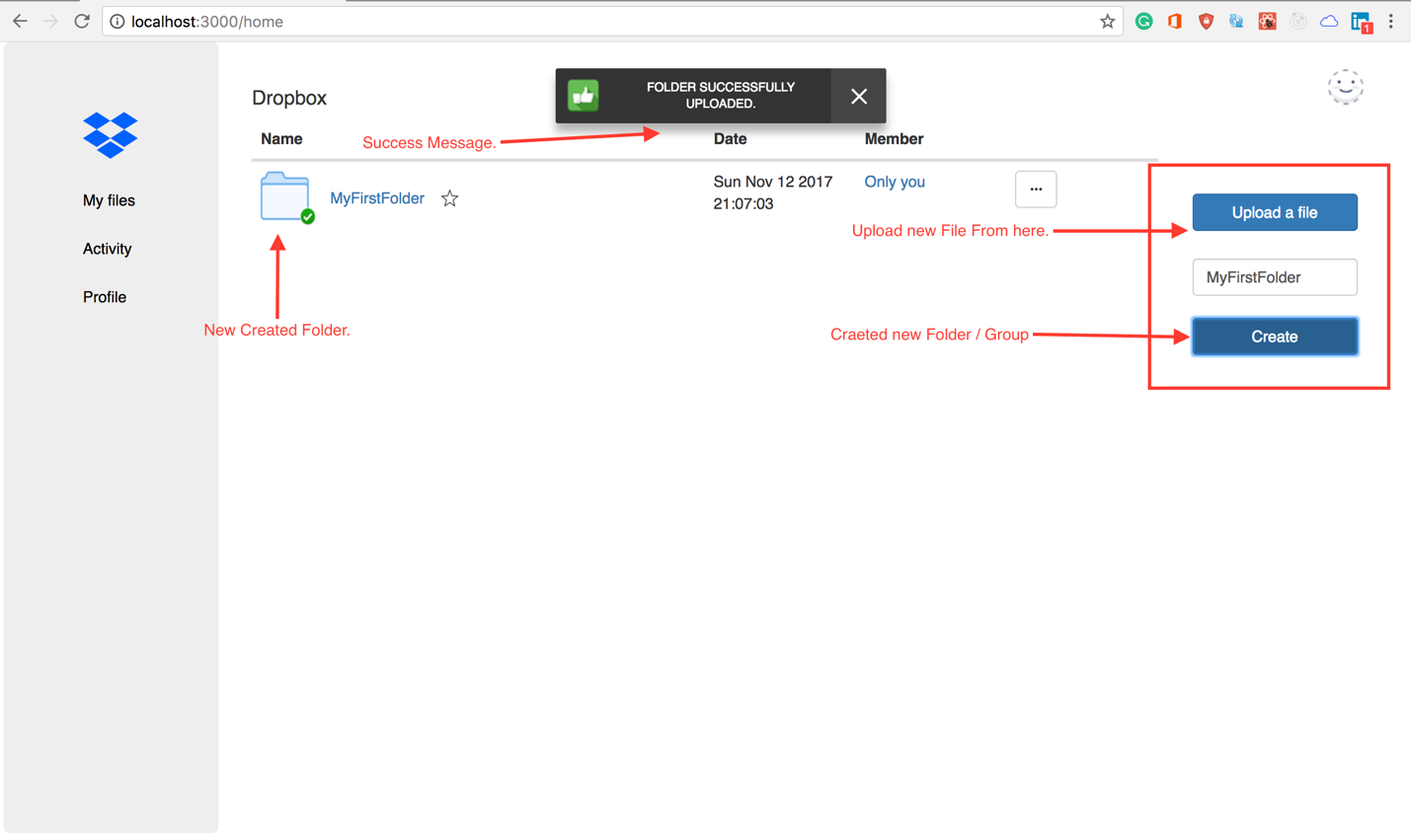
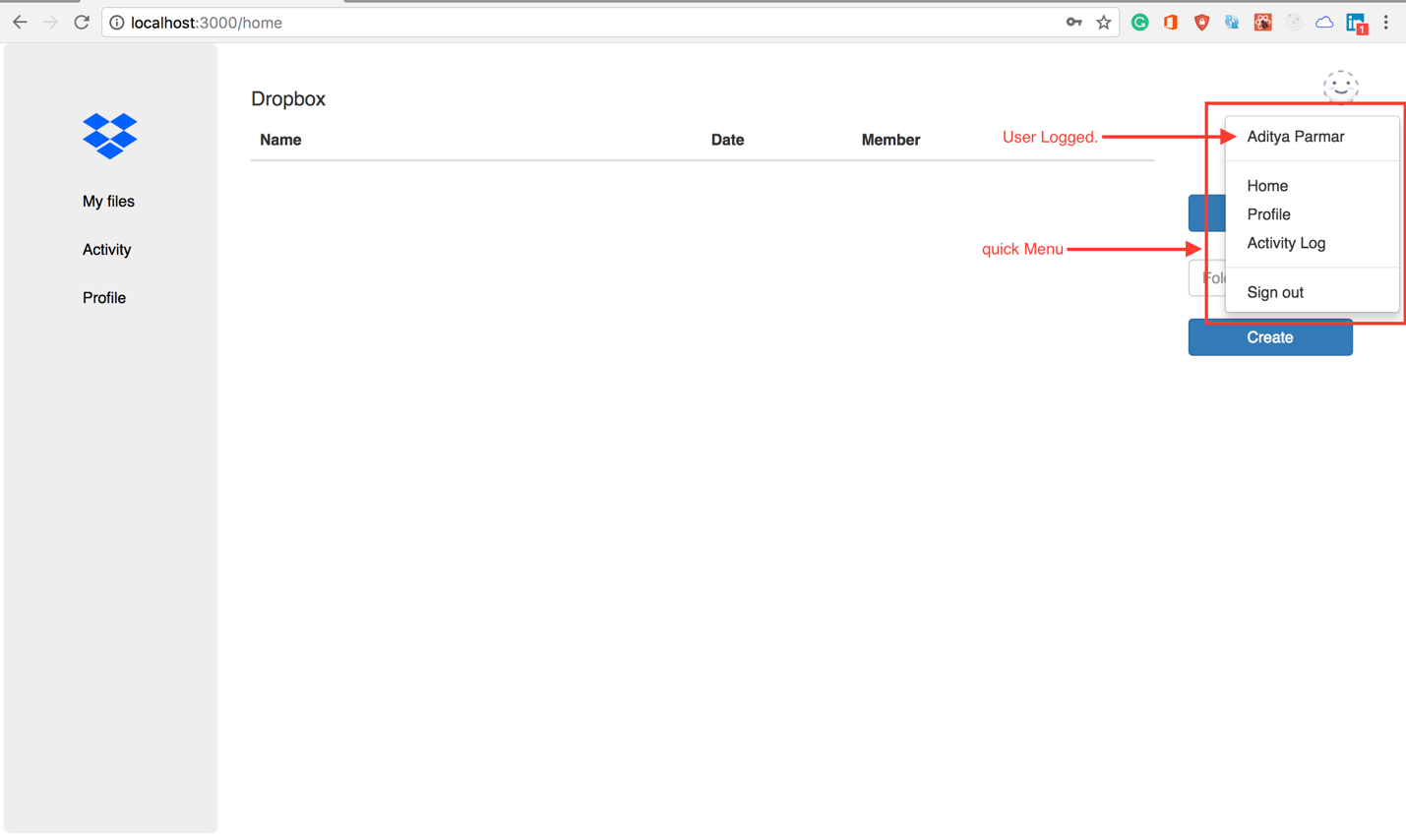
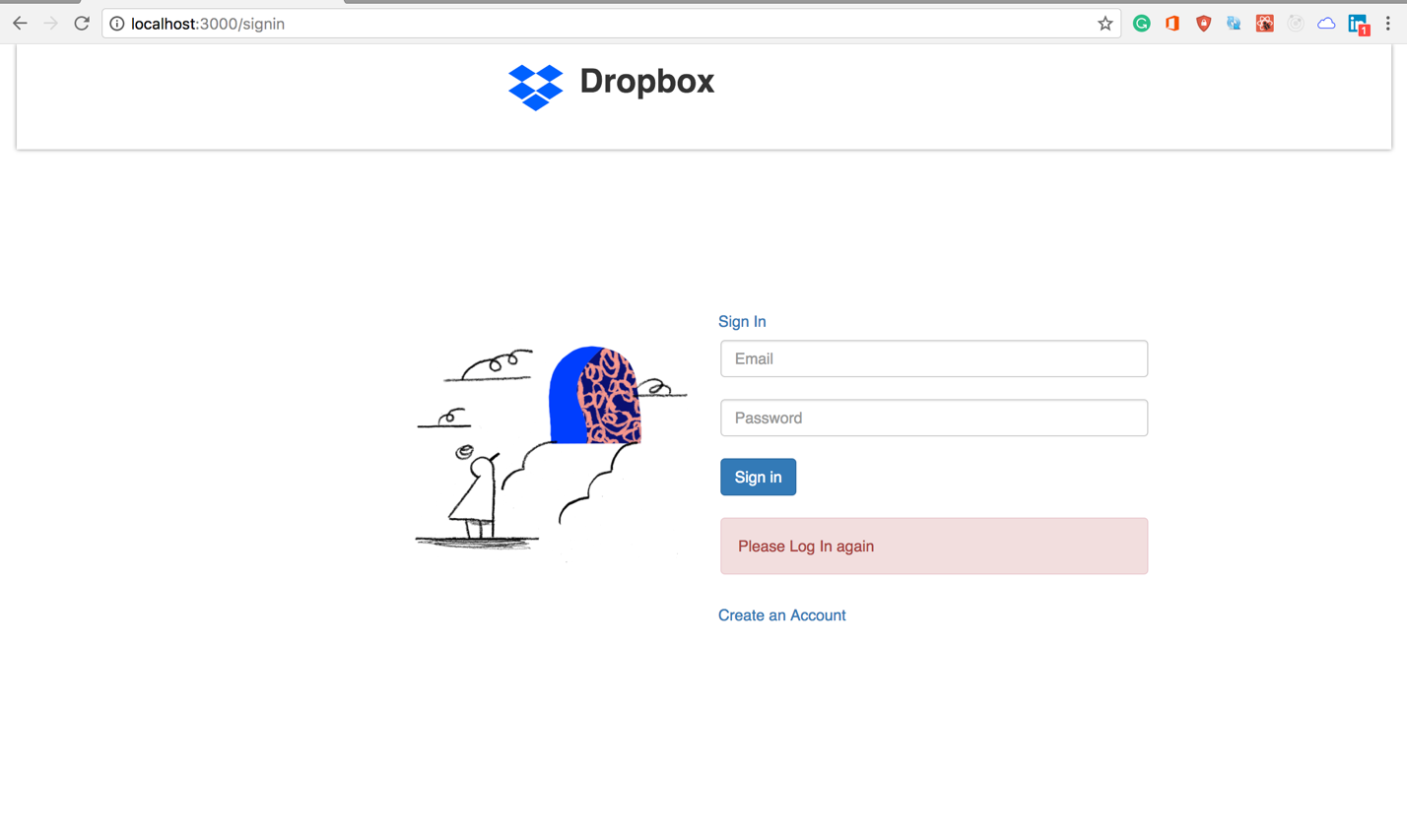
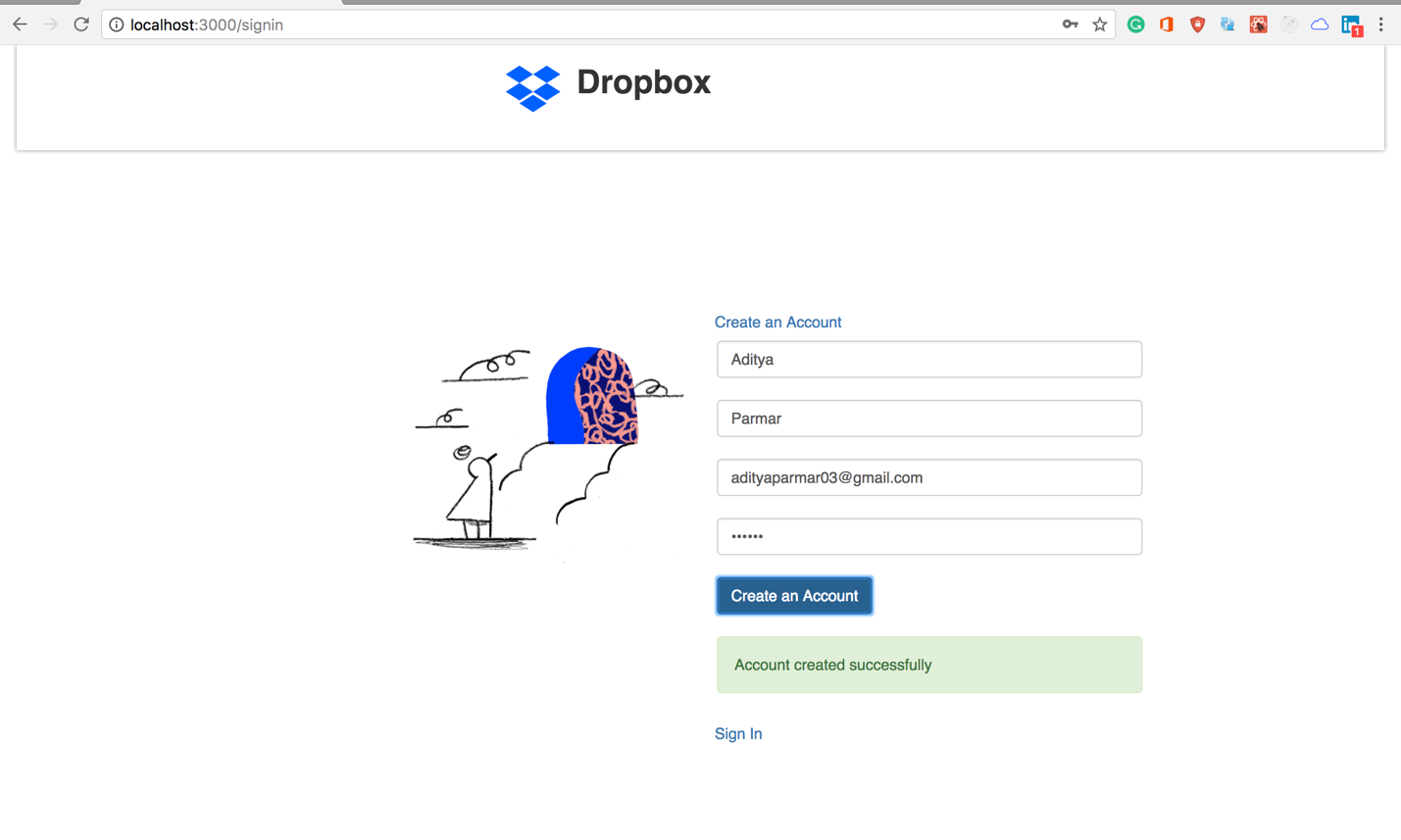
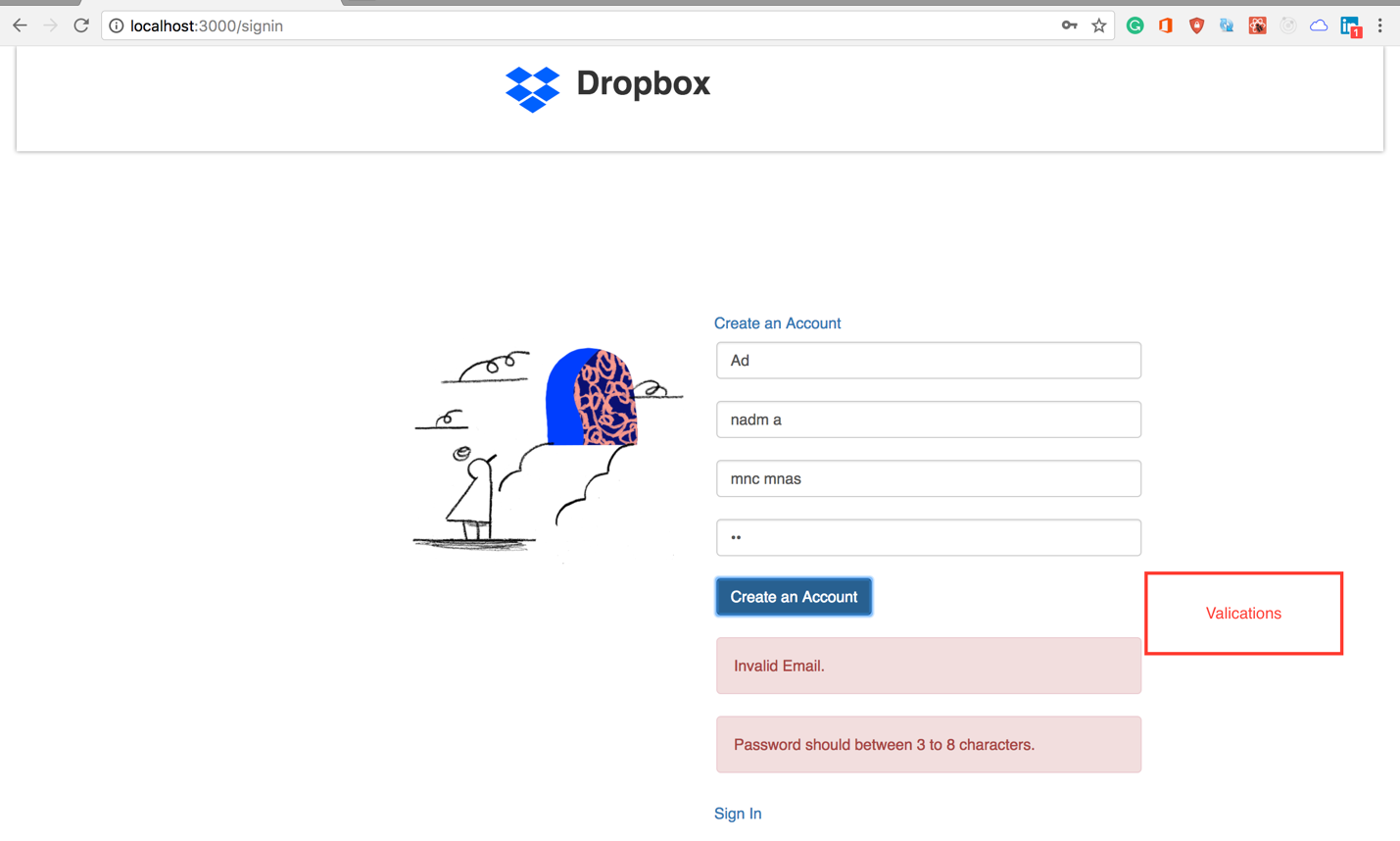
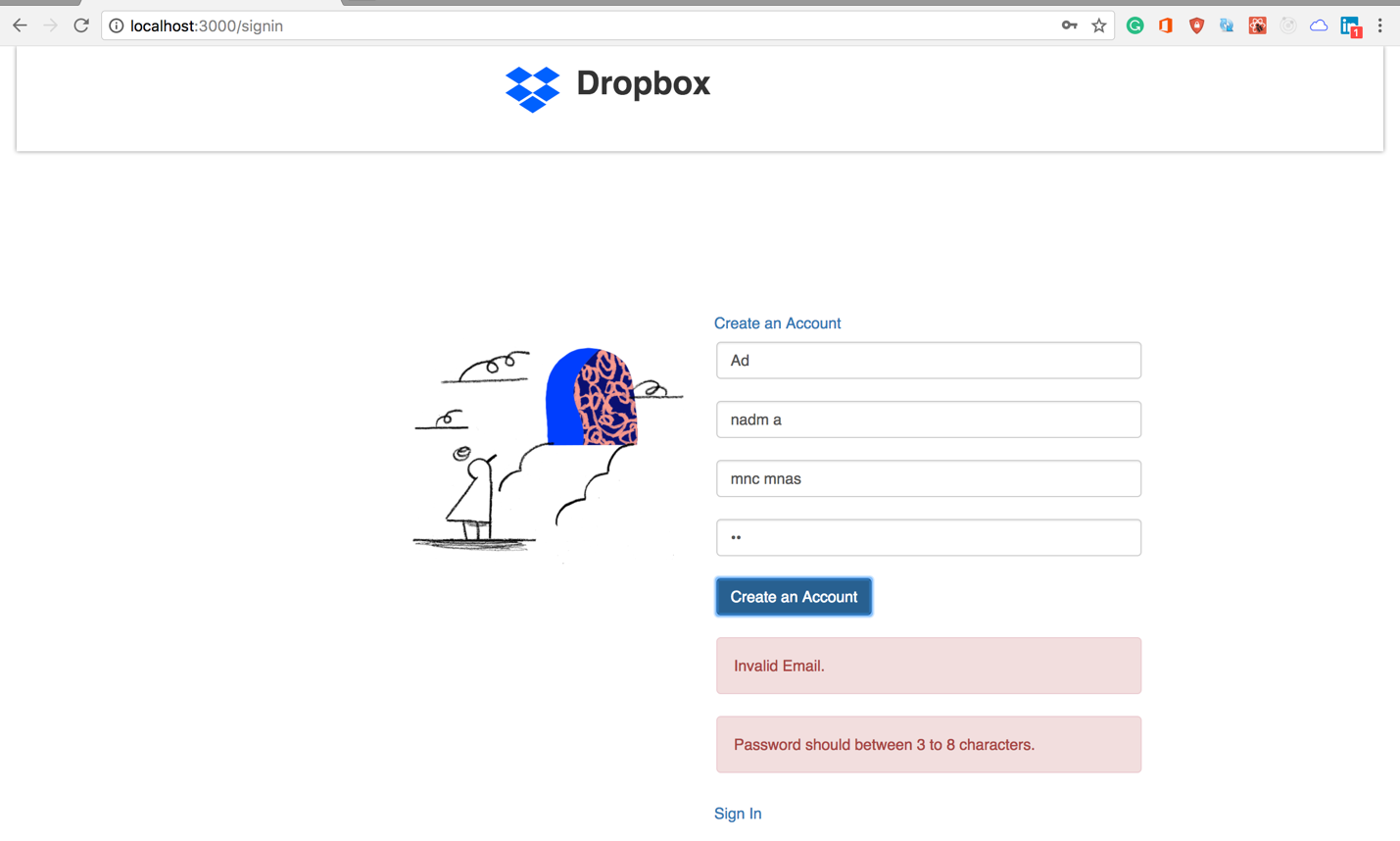
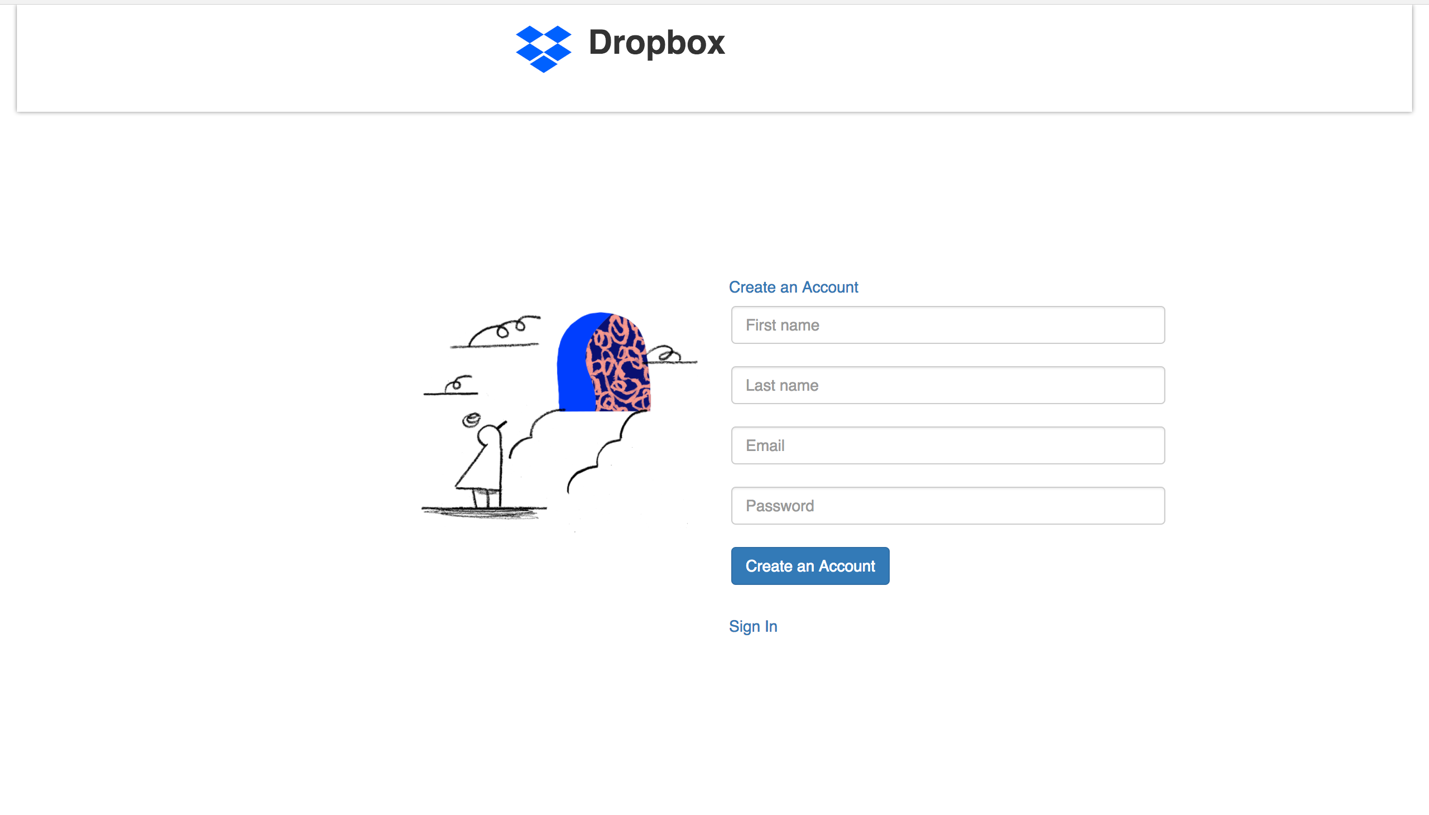
**Architecture Diagram**

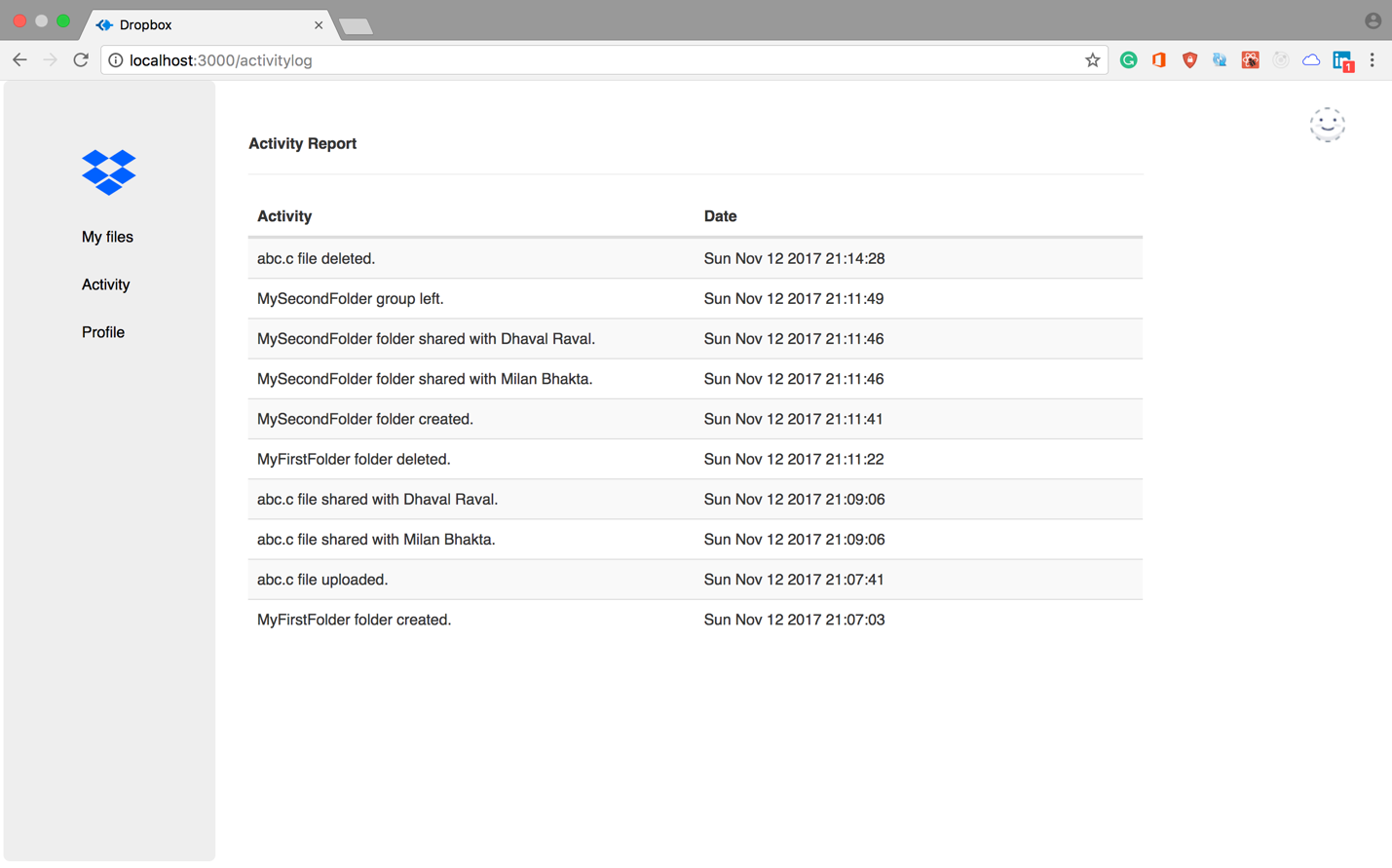
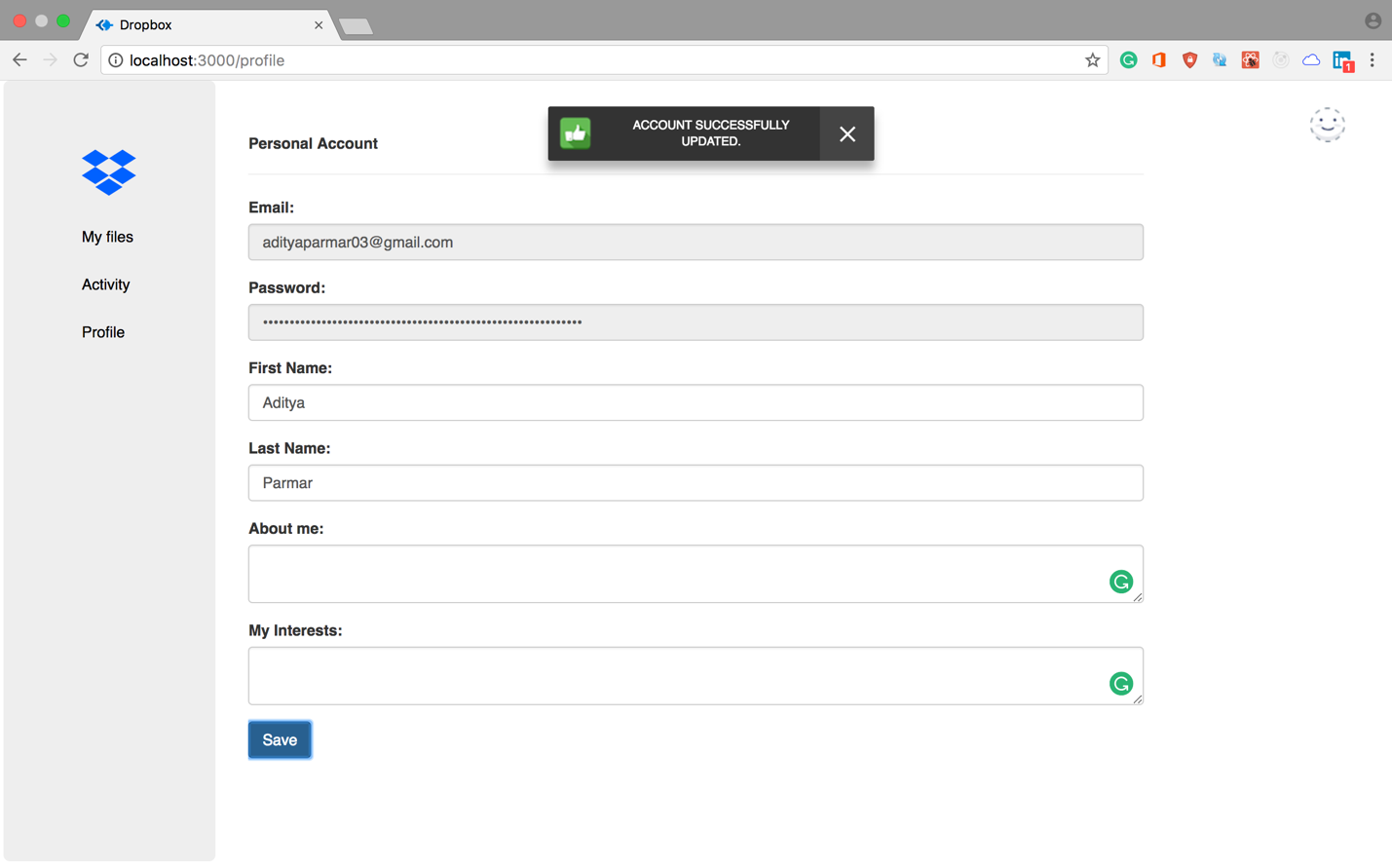
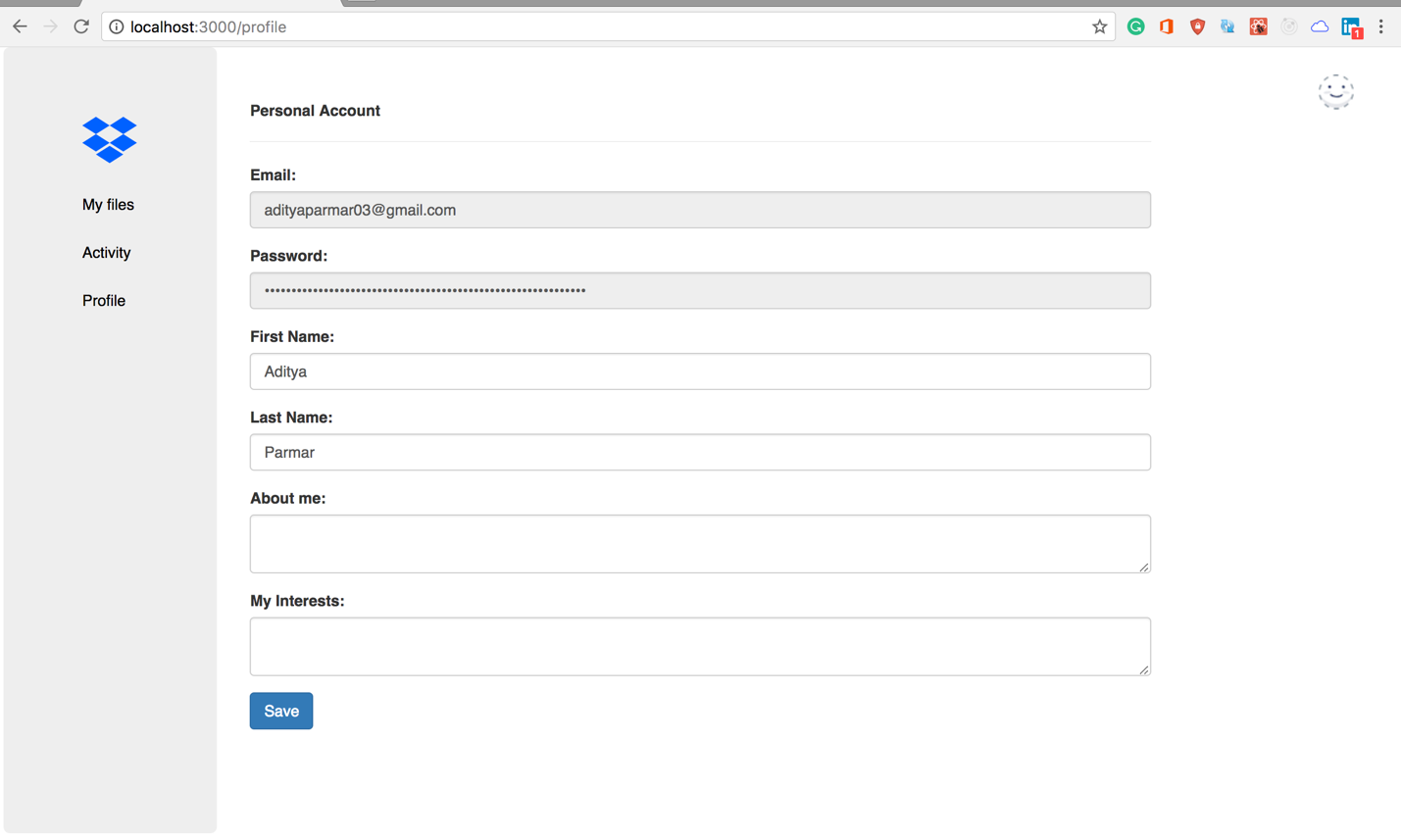
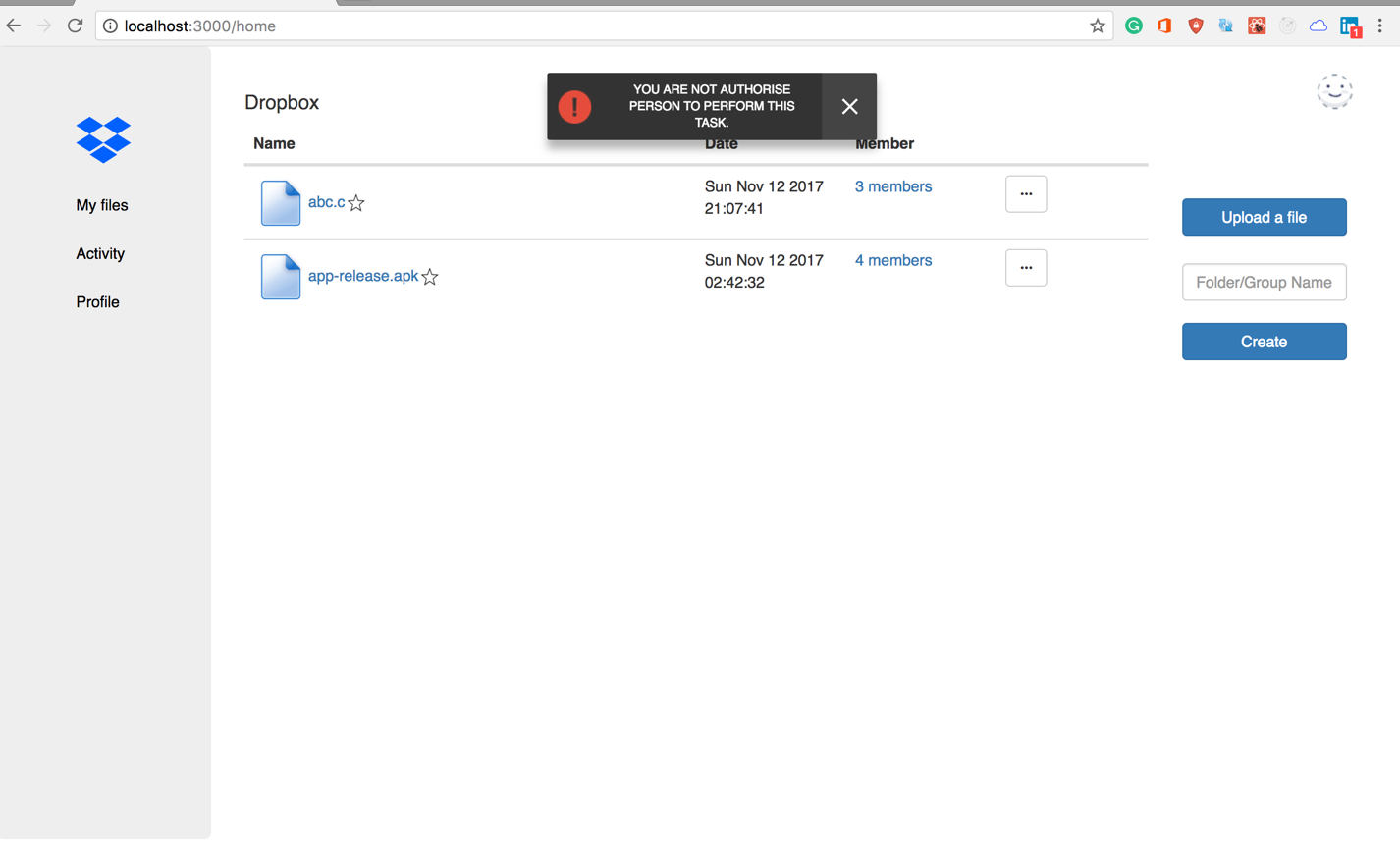
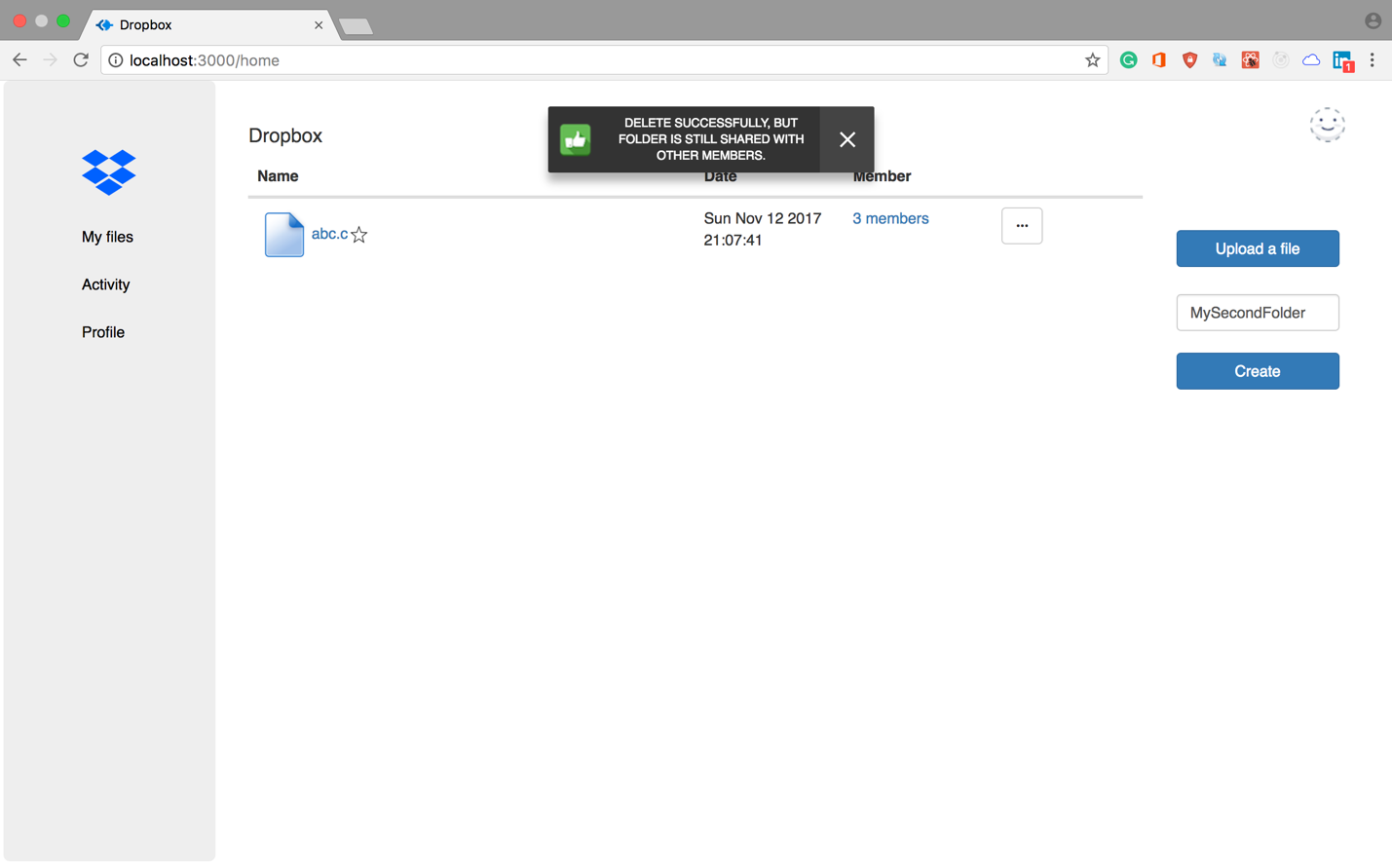
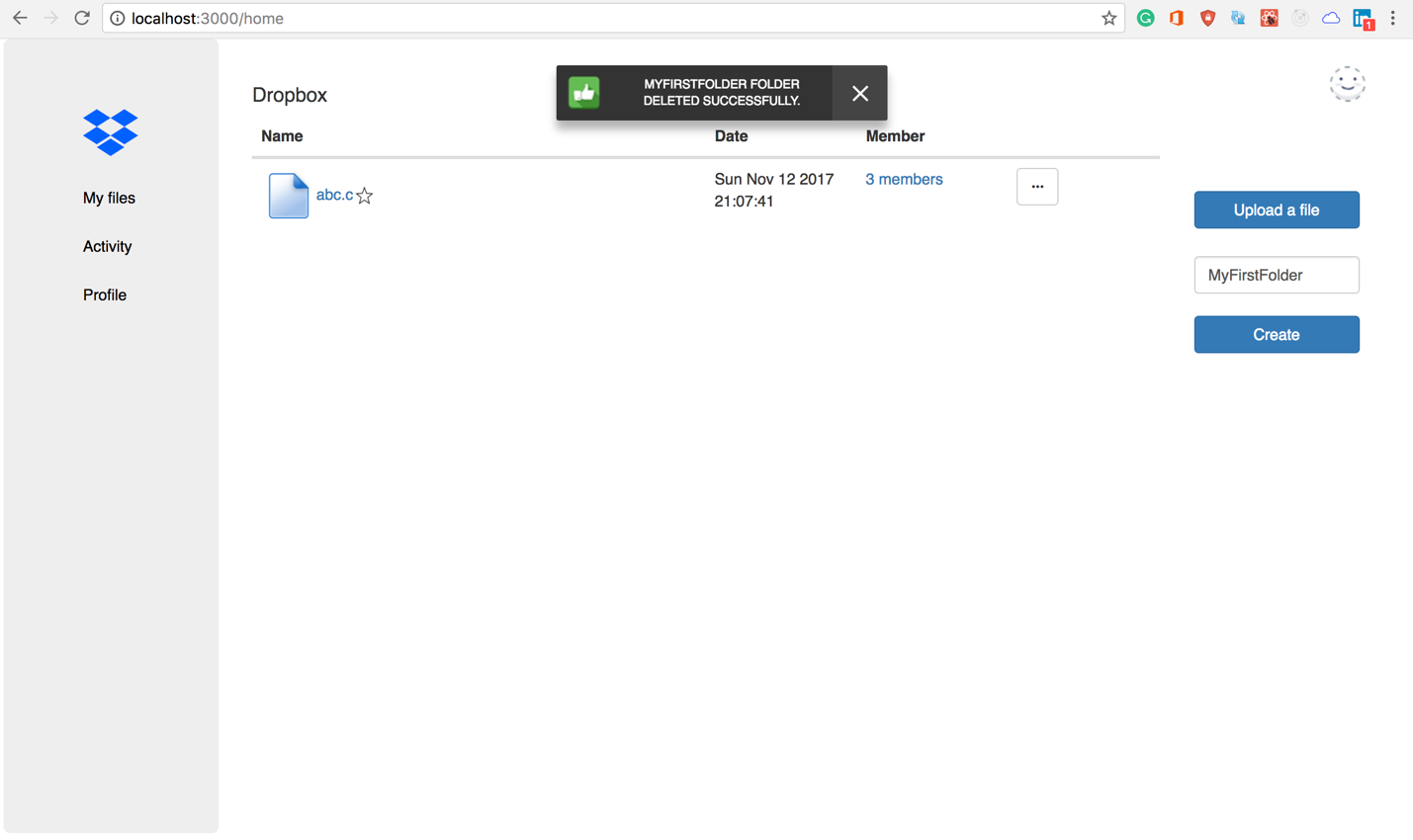
****

****

**User**

**Interface**

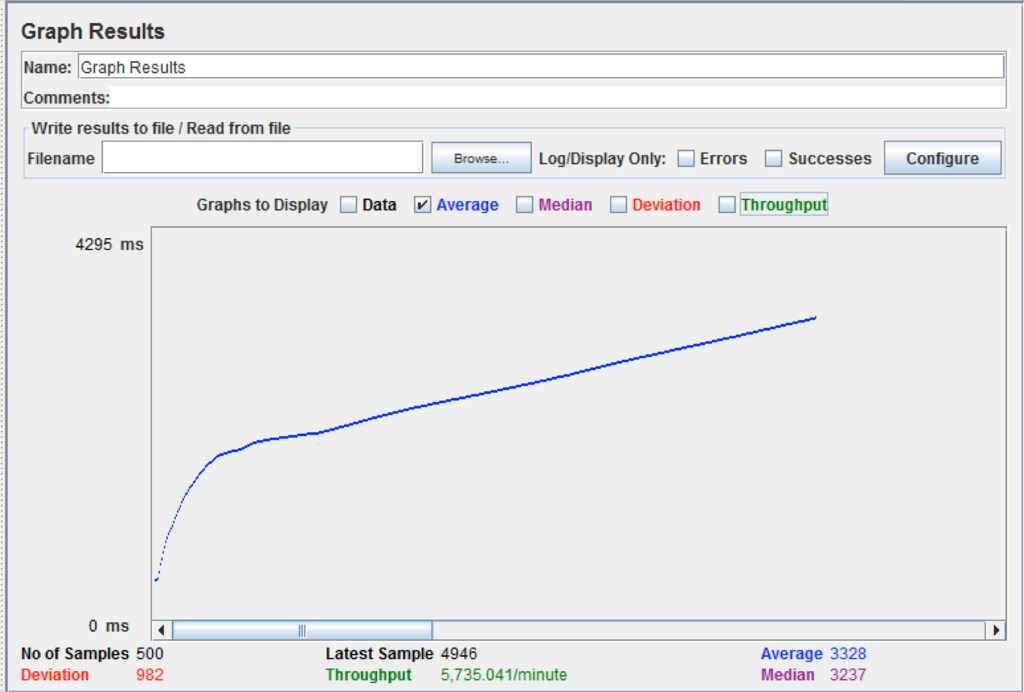
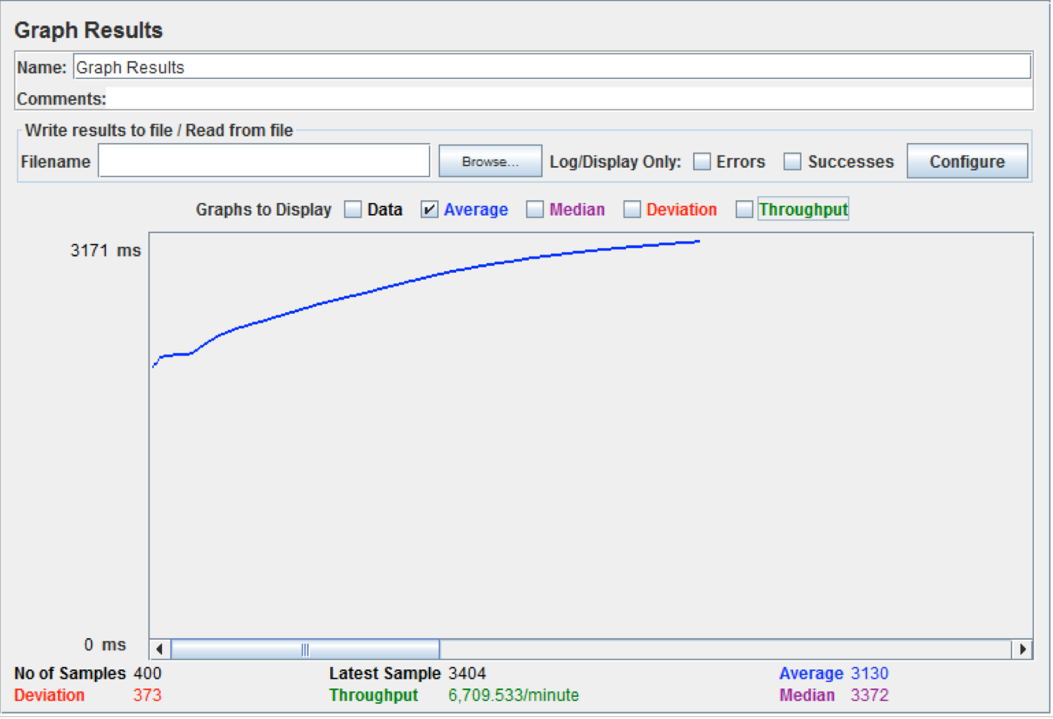
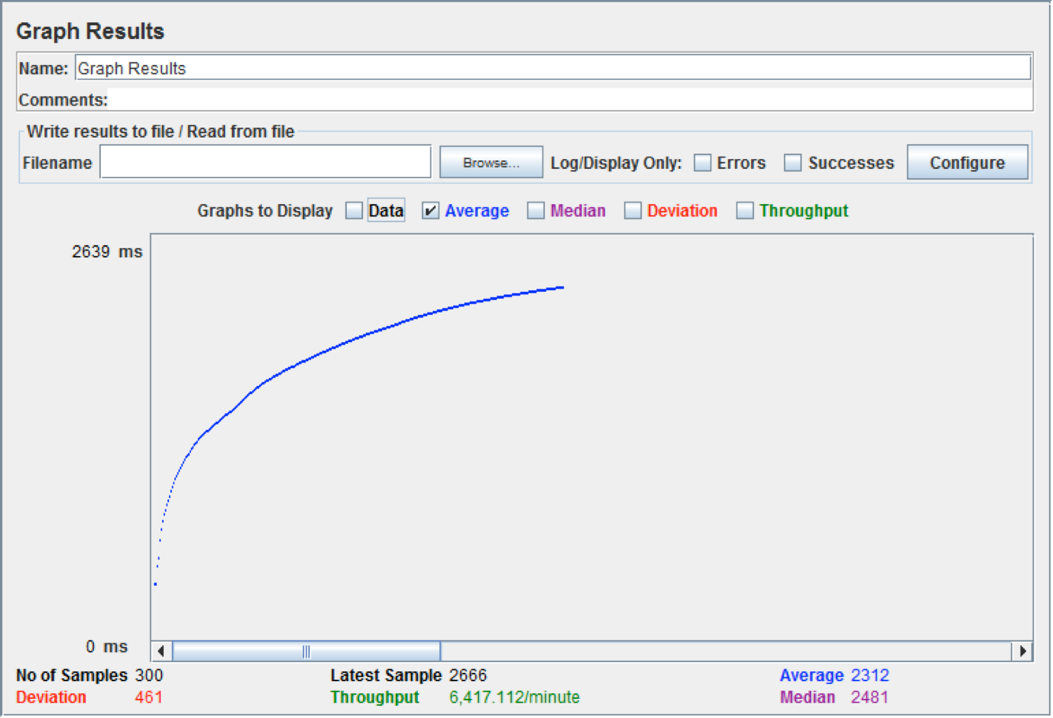
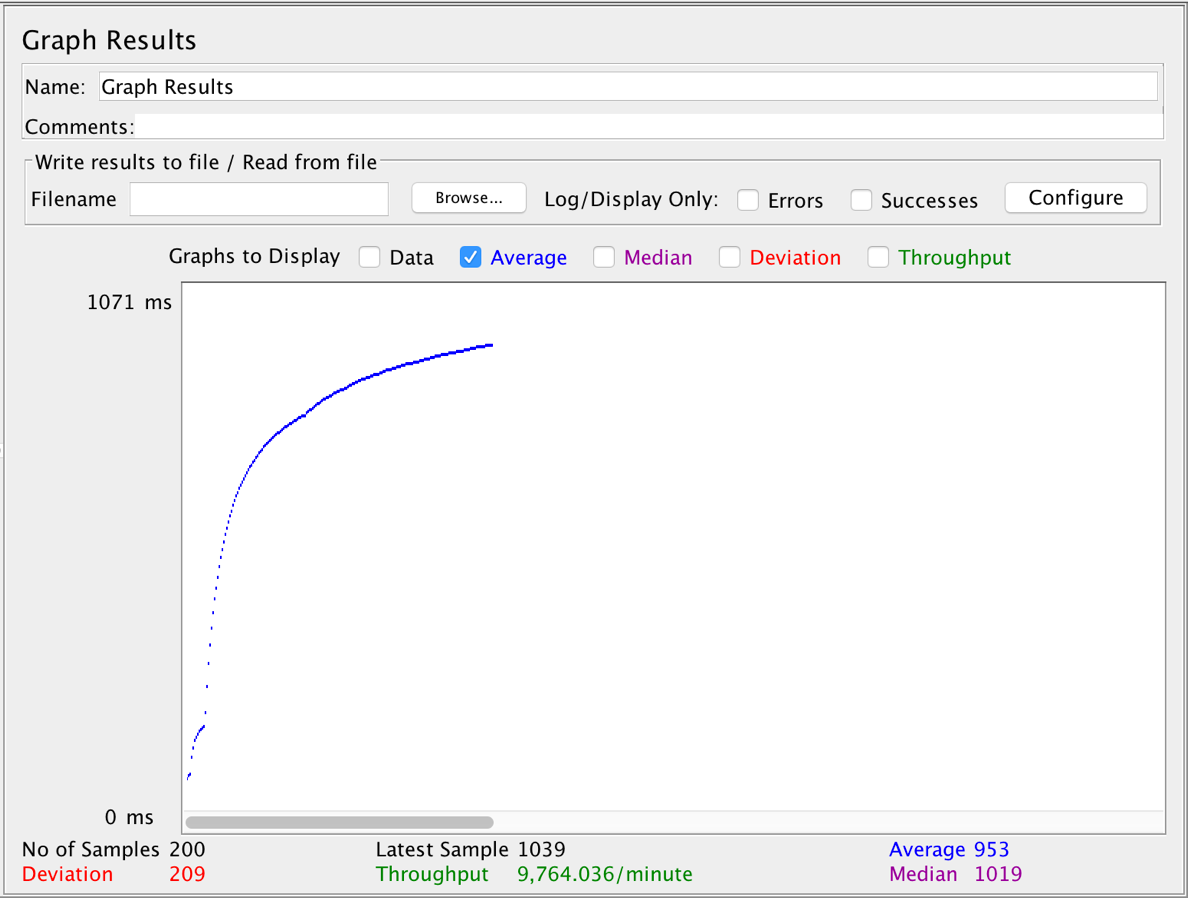
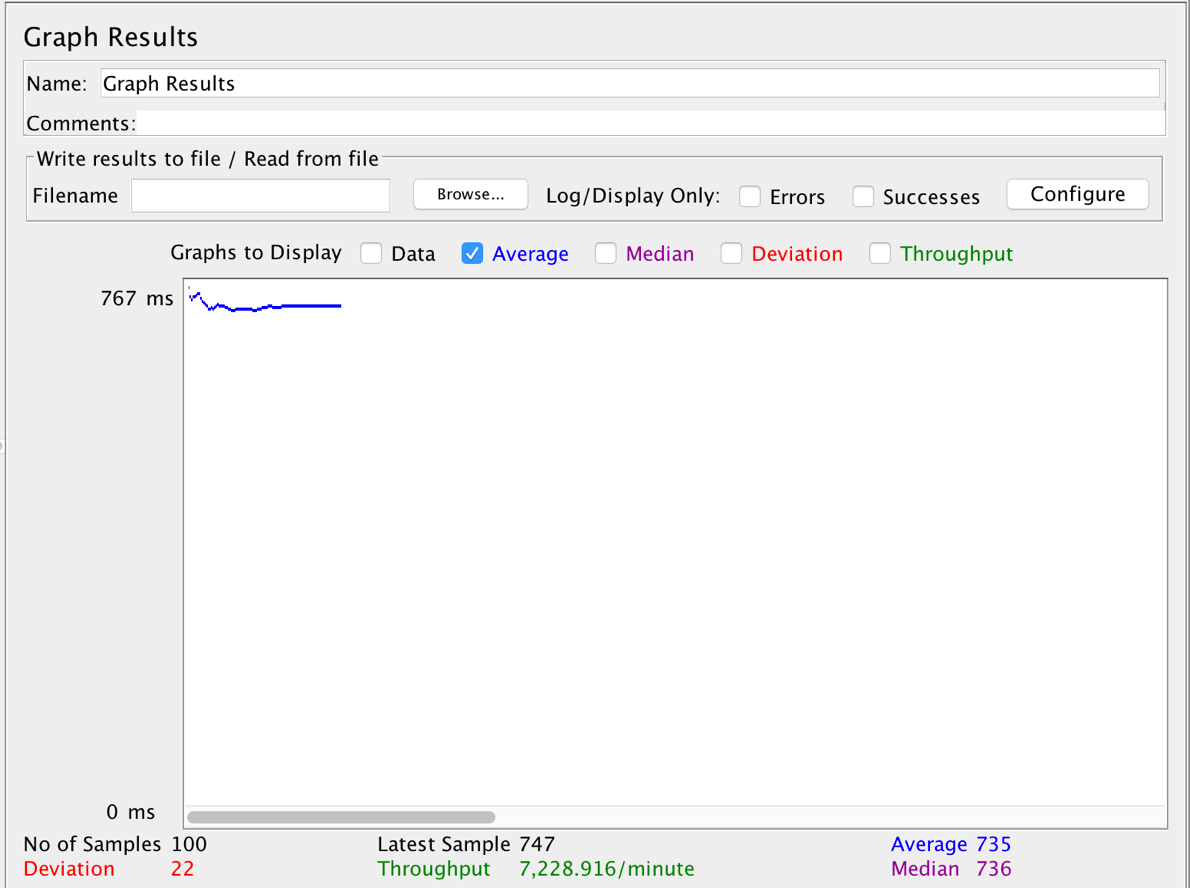
****

****

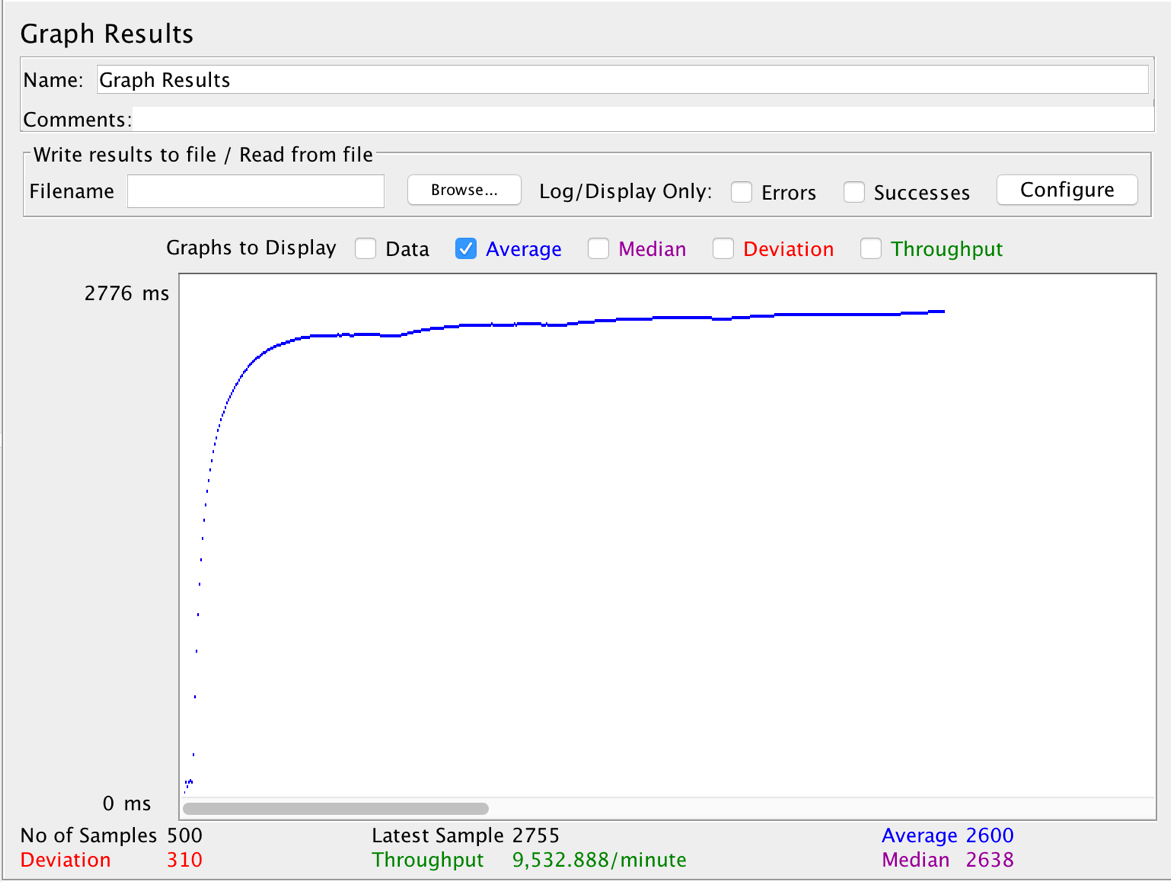
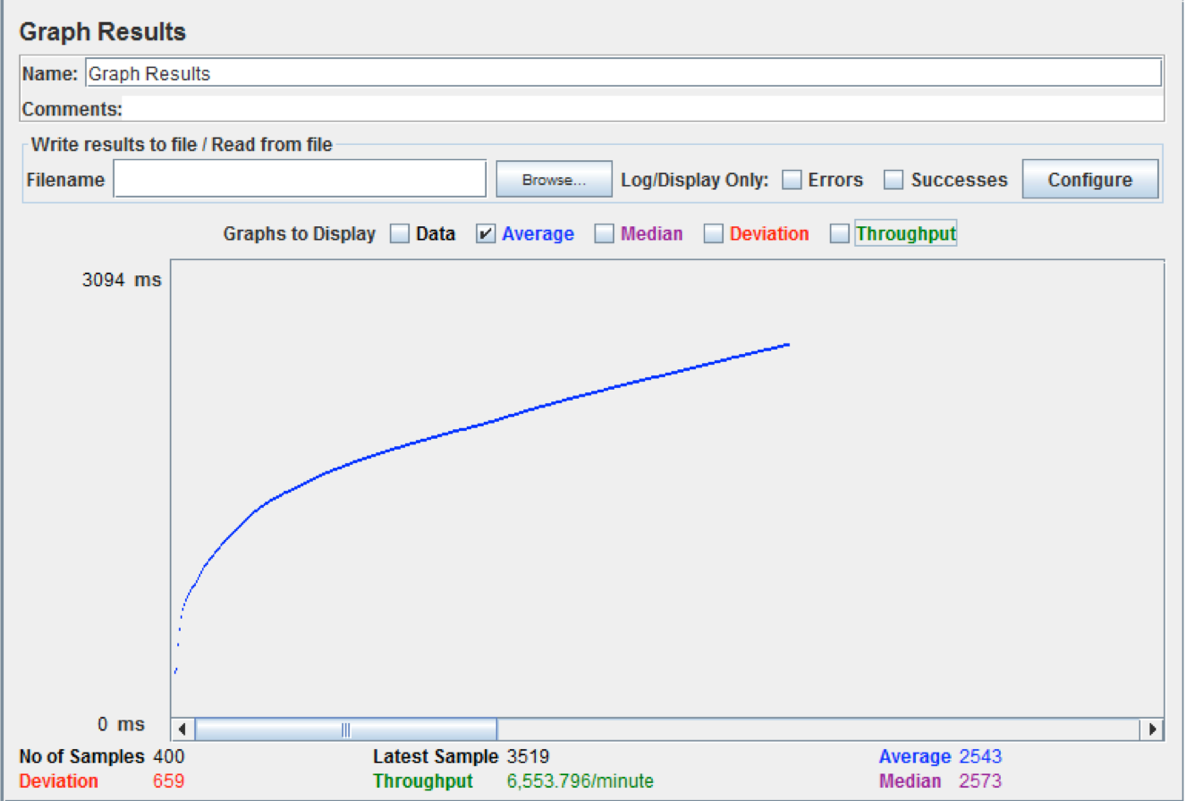
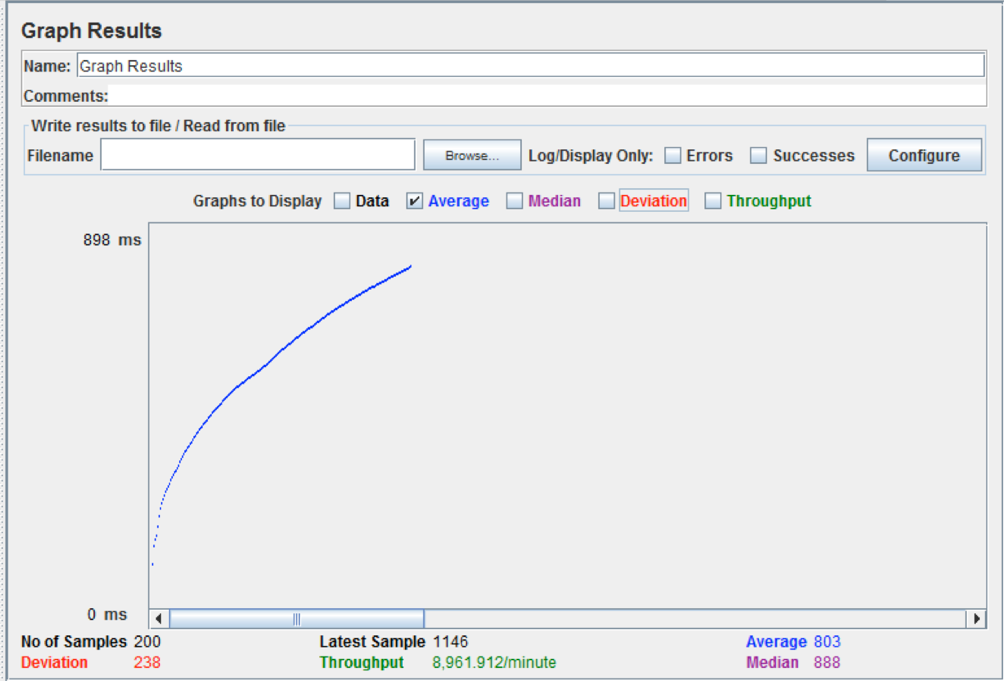
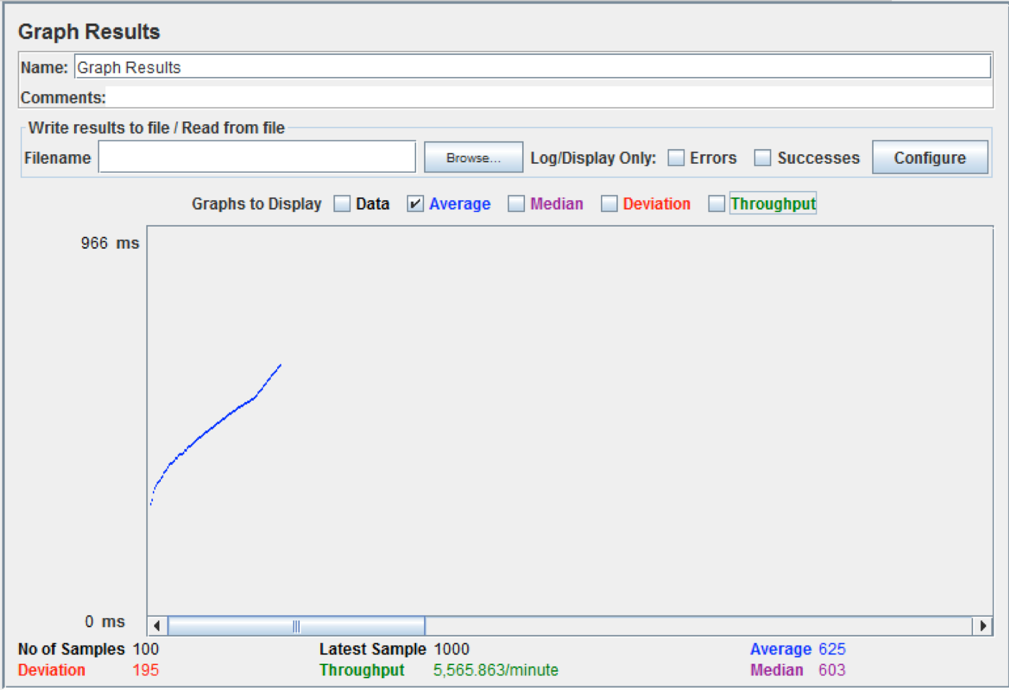
**TESTING**

JMeter

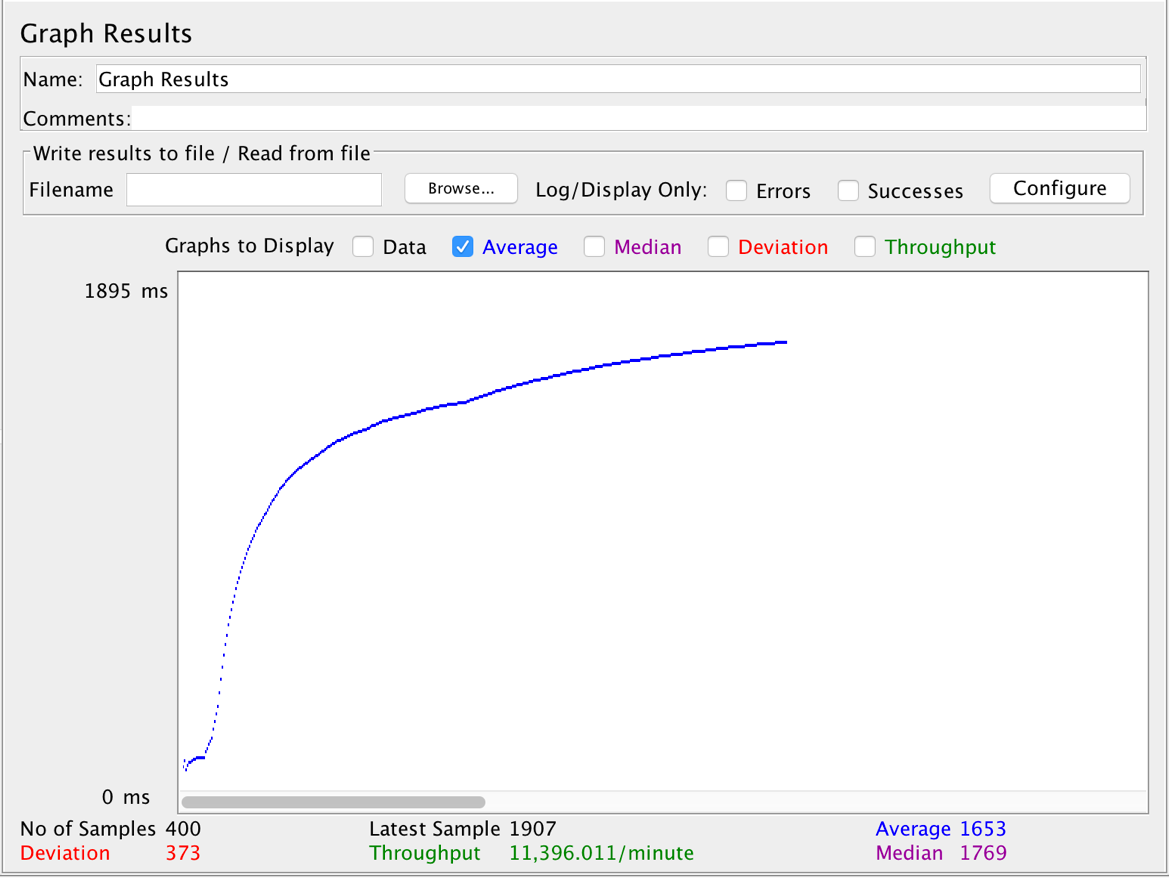
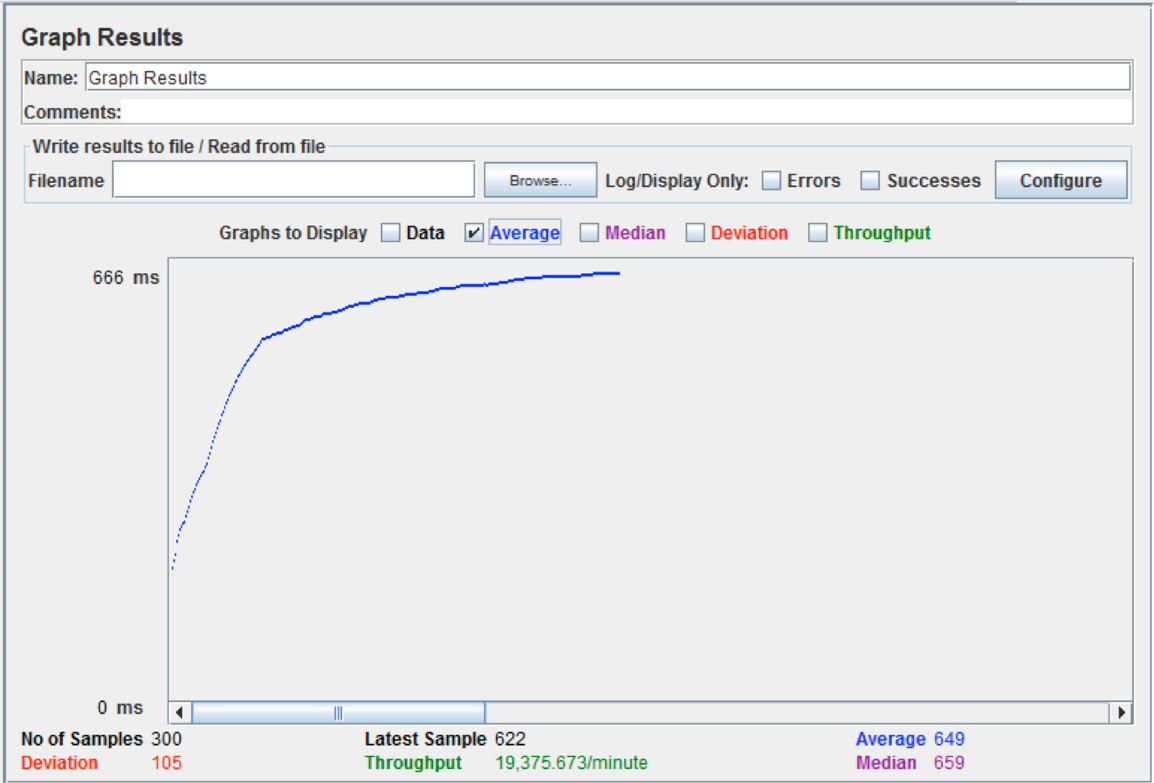
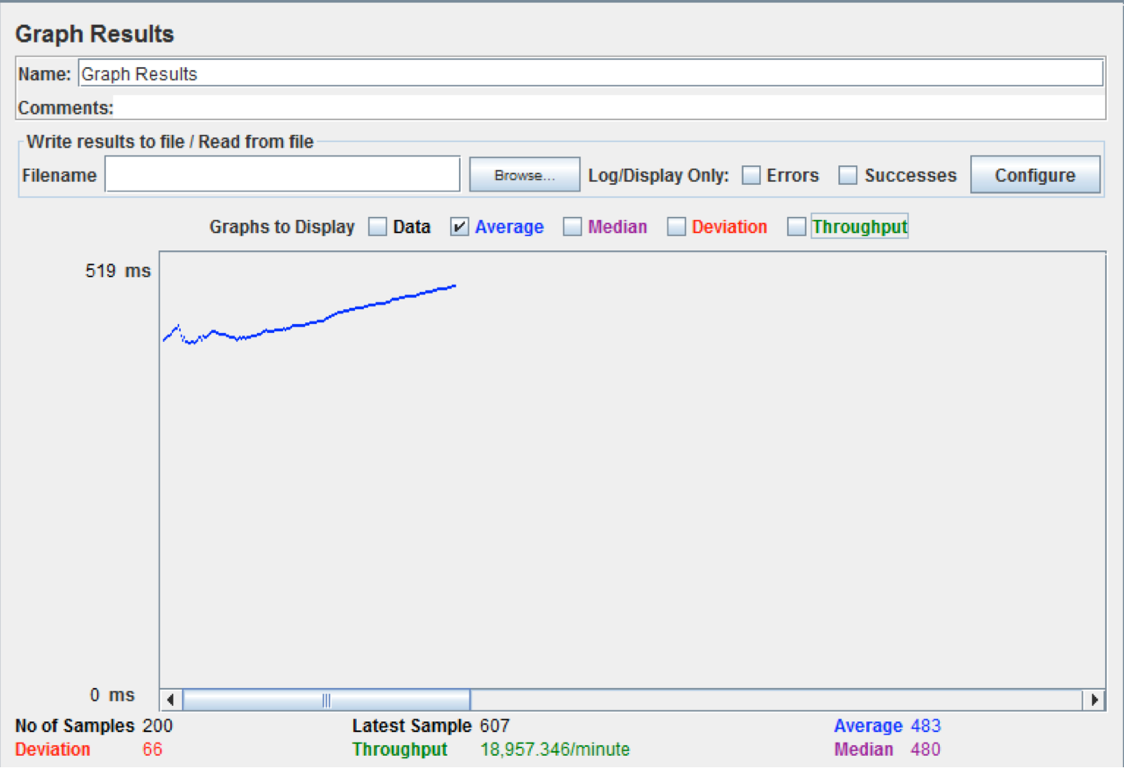
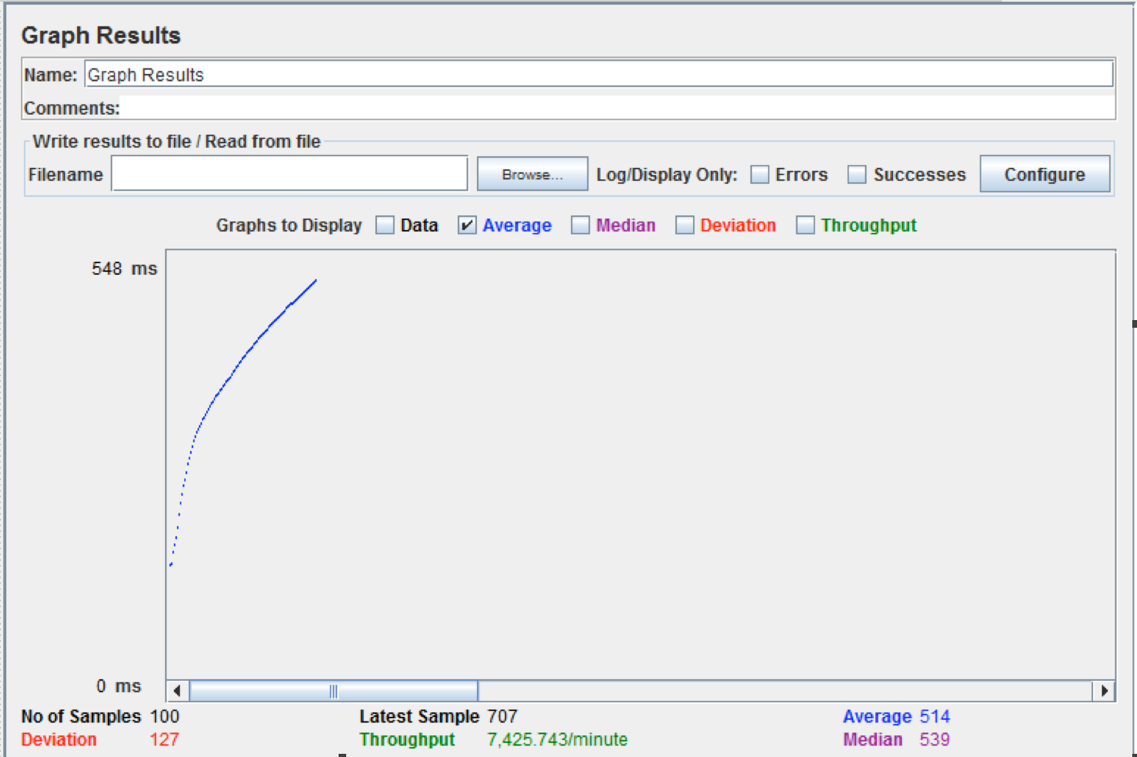
**A . Without Connection Pooling**



**B. With own implemented Connection Pooling**



**C. With Connection Pooling**



**Conclusion**

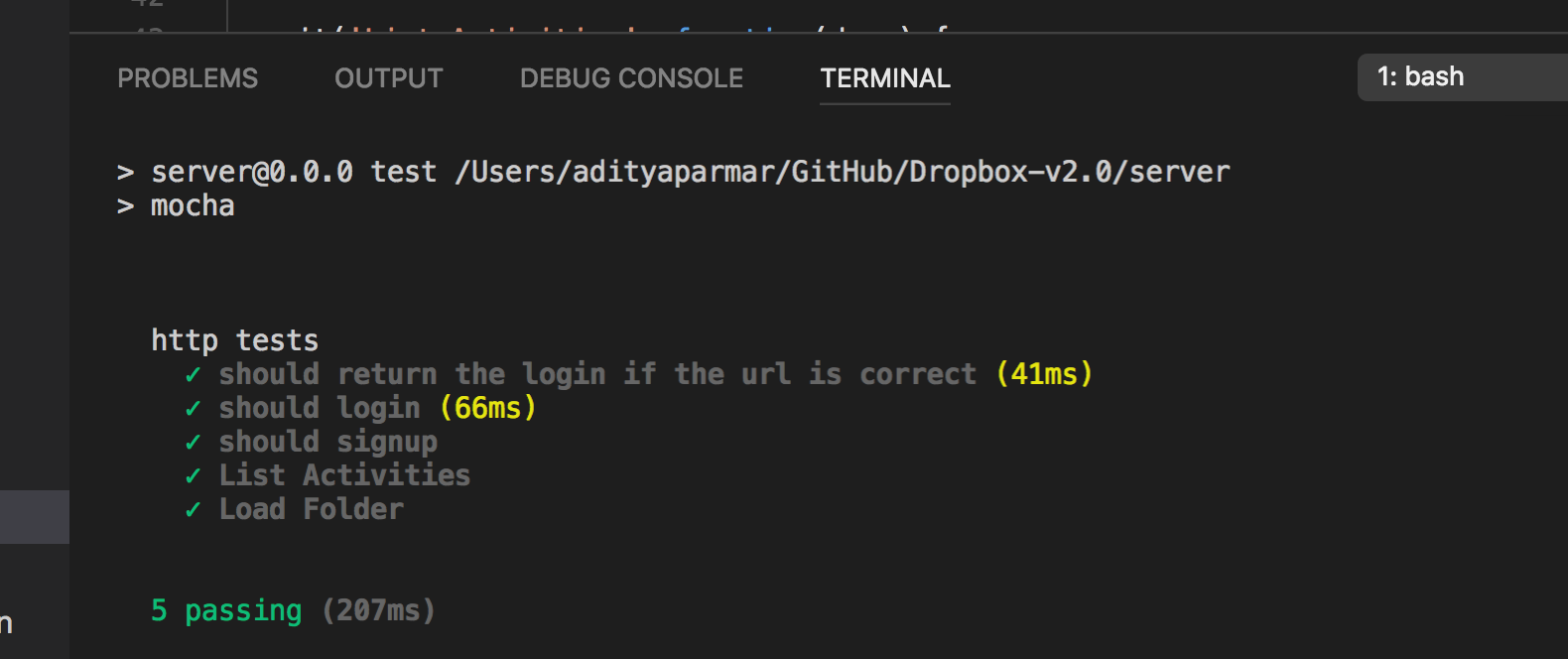
If we compare all three case, we can see that it is beneficial to use connection pooling. If we compare the Average time of 500 concurrent users, one with in-built connection pooling has **1.6x times** betteraverage response time and my own implemented connection pooling has **1.2x times** betteraverage response time. We must use connection pooling for better performance and availability.

**TESTING**

Mocha



**OUTPUT**



**5 randomly selected API.**

Questions

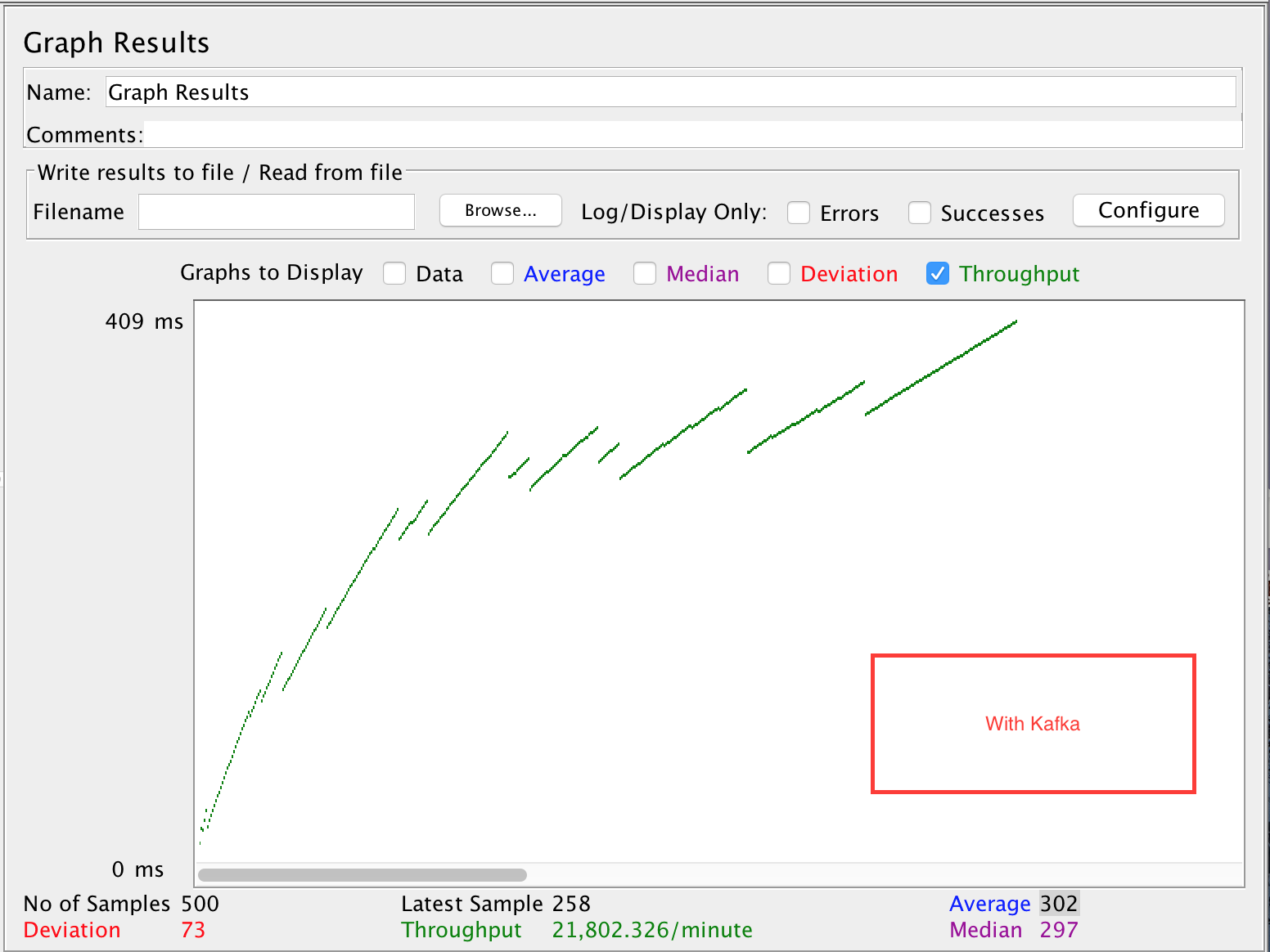
1. **Compare passport authentication process with the authentication process used in Lab1.**

* Passport is an authentication middleware for Node-Express framework by Jared Henson. Passport adds an additional layer of indirection allowing the separation of authentication logic from the rest of the application keeping the code clean and also offering inbuilt support for various other authentication strategies like Facebook Login, Google Login, etc. In this application, I have used Passport's 'Local Strategy' to implement user authentication mechanism. Passport itself doesn't encrypt the password being stored in the databases but allows to developer to implement any encryption algorithm or policy to implement encryption have use bcrpyt to salt and hash the user's password before saving them to mongo dB.
* Passport does encrypt its sessions and save to database which provide persistent session.
* In Lab1 I had used JWT token, But the problem is it affect me with persistent session. Every time I must decrypt it to establish session. It is very complex logic. But, in passport it is all handle by framework as I discuss above. It provides better performance, User Experience and security than my lab 1 authentication.

Blowfish for encryption. Both are good encryption algorithms but Blowfish does have the

added advantage of showing more resilience to rainbow and dictionary attacks.

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





* You can clearly see this without Kafka Throughput is 8,415/minute and with Kafka it is 21802/minute. You can see that it clearly more than twice better performance.
* Kafka is able to provide such availability due to being able to delegate the requests to different consumers via topic whereas in case of normal database access, increased accesses result in bottleneck situation when number of such accesses increase.
* Kafka has better throughput, built-in partitioning, replication, and fault-tolerance which makes it a good solution for large scale message processing applications. These is why Kafka is a good solution for large scale message processing applications

1. **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**

* For applications that require higher availability and tolerance, I would go for MongoDB and for the ones that need more consistency; I would use a traditional relational database like MySQL. I will use MySQL database for storing sensitive data and transaction data as it is more secure than MongoDB. I will use MongoDB for high amount of data which can be reused again in the application.
* For example, I choose mysql for authentication data which should be secure. For payment module, I use mysql.
* I used Mongodb for saving file data, sharing data and activity data which I need more available for high performance.
* I would use MongoDB for parts that need more reads as with MongoDB I can de-normalize several tables into just one or two collections which may potentially minimize my queries to just one as hard drive space is cheaper than CPU/servers.