**Software Definition**: Dropbox prototype

**Goal**:

To provide a solution for storing data and files online with ease of internet and allow to access, share, download data when it needed with proper authorization.

**Purpose:**

The purpose of this application is providing a system which allows user to put their file online and access it from anywhere.

It also allows user to download the file when they needed. System also provide the facility to share the files among other users or someone outside the system. It also allows user to quickly view the stared files.

**System Design:**

We are using **MERN Stack** which include

1) **Mongo DB** as database,

2) **Express Js, Node Js** as backed server

3) **React Js** as frontend.

**Mongo DB**:

Mongo DB is much more better than our traditional RDBMS databases.

Below are some properties which make it better than traditional RDBMS databases.

* **Schema less** − It is a document database. One collection can hold different kind of documents. Number of fields, content and size of the document can differ from one document to another.
* Clear Structure for a single object.
* complex joins not allowed.
* document-based query language: MongoDB supports dynamic queries on documents using a document-based query language that's nearly as powerful as SQL.
* Easy to tune.
* **Easy to scale-out** − MongoDB is easy to scale and its one of its inbuild functionality.
* BSON: Application objects can be directly stored in mongo DB. We do not need to convert them in database object form.
* Fast access: It Uses internal memory for storing the working set, which is allowing fast access of data.

These are not only features which make it better. Below are some more points which make it the best for our purpose.

* **Document Oriented Storage** − Data is stored in the form of JSON style documents.
* It allows index on every attribute
* It allows inbuilt replication and in result high availability
* It includes Auto-sharding
* Fast in-place updates

In our solution we have used mongo DB with only **one server** i.e. without replication.   
We are using our awn **connection pooling** to accommodate high number of database calls without opening and closing connections. Connection pooling is described in later topics.

Also, for ease of work, a **database util** file is created to accommodate all the database operations and a centralize access of connection from one file itself.

**Node JS :** It is used for the server implementation in our solution.

Some of the benefits using node Js is as below.

* **Asynchronous I/O**

The core of node js is to handle asynchronous I/O calls from the ground up and it is a useful solution for most of web- and network-development problems.

In addition to fast JavaScript execution, Node Js has benefit of asynchronous I/O calls which makes the system much faster.

* **Javascript**

Node.js is Javascript.  As we always have javascript on client side, we are making our whole system running on java script and making it single language application.

* **Community Driven**

It’s a community driven language which makes is more open for new enhancements added by the wild java script community available in outer world. One of the most famous is  [Socket.io](http://socket.io/), a module to manage persistent connections between client and server, enabling the server to push real-time updates to clients.

Node mailer is another example of the same and it is also used in our solution.

In our solution, Node js is used for back end business logic implementation. As it is asynchronous, it allows fast execution and high throughput for the application.

Modularity is implemented in the system by putting proper code in the proper directory. This also improves the readability of the system.

Each function is made as small API calls for reusing purpose to reduce the code redundancy.

**Express JS:** Express js is used for web framework.

Express Js has benefits over plain vanilla node js.

* Using express js, the web service implementation is made easy when compares to plain vanilla node js.
* Express js allows access to node modules like CORS for multi domain application.

The CORS technique is used to separate the client and server and still maintains integrity.

We have app.js as controller of the server which catches all the incoming request and redirect them to the handler.

**React JS**: On Client Side, we are using React Js**.**

* The purpose of using react js is to take advantage of unidirectional data flow in UI and partial rendering technique.
* It allows use to render a small part of DOM where the change is needed. Also the changes made are really quick as they are event driven.
* The CORS technique allows the React client to send data to the Node Server.
* Easy support for bootstrap using react-bootstrap module

In reach Js we have a request sender named API.js which is taking care of all the request sending and response gathering from the node server.

Bootstrap and material UI is used to make the system more interactive.

Apart from MERN stack, other components are also used to make the solution more durable and more reliable. Those technologies are as below.

**Kafka Message Queue**: Kafka message queue is used to increase throughput of the application in case of multiple servers available.

Benefits of using **Kafka** is as below.

* **Kafka is Highly Scalable**
* **Kafka is Highly Durable**
* **Kafka is Highly Reliable**
* **Kafka Offers High Performance**

**To get benefits of above mentioned features, Kafka is used in the solution provided.**

**The whole back end is divided into 2 parts.**

1. **Express/Node backend webserver: This part contains Express web application which handles incoming requests and redirect them to respected node handler.  
   Node handler puts the request in the message que and start listening on the response queue. When response received, it revert it back to the client.**
2. **Kafka back end application : This part keep listening on kafka queue for any request message.  
   If any message received, it is transferred to proper api to handle and process the request.  
   The output of API is then appended to the response que.**

**Passport JS:** The main purpose of passport Js is User Management as a Service.

Passport JS allows us to provide our own user authentication, also Everyauth of passport js allows us to authenticate user by using Google, Yahoo, Twitter etc accounts.

Passport Js stores User Session ins database and so the session management is made horizontally scalable.

Passport Js is used with express session and express cookie management.

**Connection Pooling**: a custom connection pool is implemented to make pool of connection in bulk and use them when needed.

The connection pooling technique Pseudo code is as below.

**var** *createConnectionPool*=**function**(count,callback){  
 **for**(**var** i=0;i<count-1;i++){  
 *createConnection*( **function**(err, \_db){  
 free\_pool.push(\_db);  
 });  
 }  
 callback(**false**,**true**);  
}

**var** *getConnection* = **function**(callback){  
  
   
 **while**(free\_pool.**length**==0);// wait till pool has some connection to give  
 **var** db = free\_pool[0]; //taking first connection from pool  
 free\_pool = free\_pool.slice(1,free\_pool.**length**); // removing connection

// from pool  
   
 callback(db);  
   
};

**var** *closeConnection* = **function**(db){  
 free\_pool.push(db);// pushing connection back to the pool  
}

**Centralize DB API**: one centralize API is made to work with database and accommodate all possible DB operations in one file and to reuse the same code everywhere.

Also it allows the system to be open for any enhancement in connection creation and pooling ways as its only one file where we need to put the change and all other files will be automatically changed.

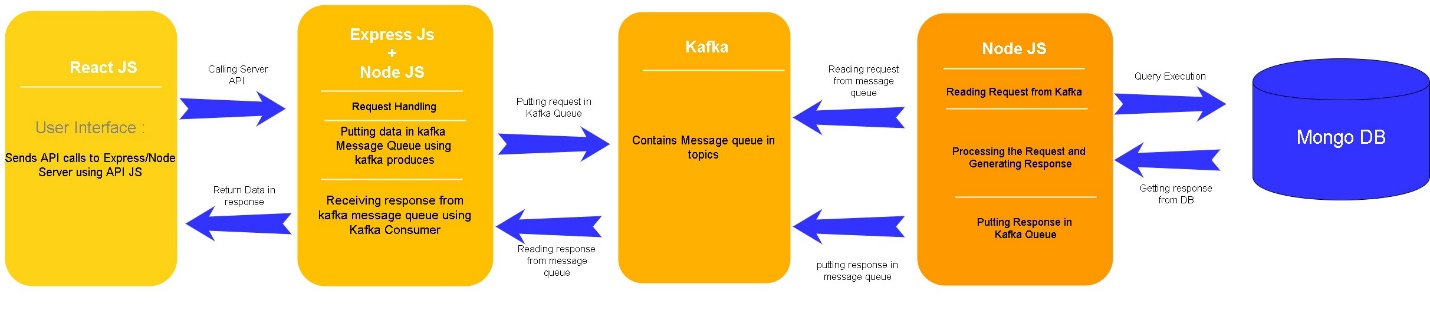
**Authenticity Interceptor:** In Express The system is using one server interceptor to authenticate each and every request. If requests is authenticated, it will be redirected to handler else the unauthorized access status will be send to the client.

**bCrypt**: While storing passwords into the database, **bCrypt** encryption is applied to make passwords unreadable.

The Salt and Hash technique is used to make password more secure.

**NodeMailer** is used to send sharing notification to the users about file sharing.

**System Architecture**



**Front End: React JS** is used as front end of the application. It fetches data from the server using API.js which contains system calls for all APIs.

**Server: Express JS + Node Js** is used as Server-side application of the System. All calls made from front end is intercepted by Express Js in app.js file and redirected to proper Node API to be processed.

The system is divided in terms of system logic and request serving. The Express-Node server is just available to get the request and instead of processing them, it send that to a **kafka queue** in appropriate topic using producer and starts listening on the response queue for response. Once Response received, it sends that back for Front End.

**Message Queue**: Kafka Queue is used to implement message queue for request sending and response receiving.

**Back-end Application: Node JS** application is used to implement business logics. It continuously reading the Kafka queue for incoming messages and after getting a message on any topic, a business logic is run and the response is being put in Kafka Queue.

**Mongo DB:** Mongo DB is used to sore the data.

**Database Design**

**Users:** To store user signup information and user profile information

**Directories**: The Collection is used to keep track of directories in the system.

**Directory\_Logging**: this Collection is used to store the operations performed on the directory by user

**Directory\_permission**: This Collection is used to store directory sharing information

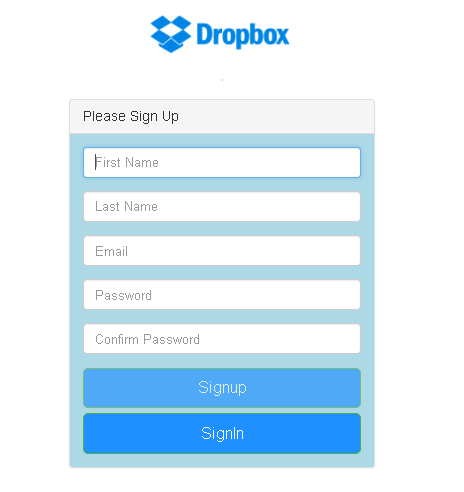
**Staerddir**: This Collection is used to store starring information of directory by user

**Usergroup**: This Collection is used to store data related to user group and associated users in the group.

**System Flow:**

The system starts with the login/Signup pages.

**Sign UP**



This is the functionality which a user can use to register themselves with the system to use its functionality.

The users will have a screen where they must insert some basic information such as first name, last name, username, email address, password.

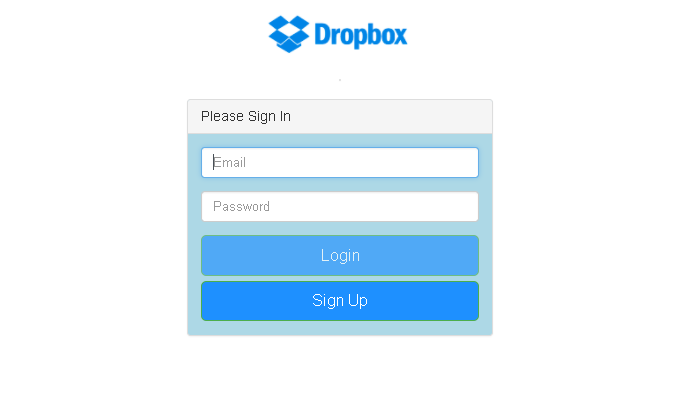
After successfully inserting data, user clicks on the Signup button and the system is doing basic validation on the values entered by the user.

If all data are correct, the system is generating a new account for the user and redirect them on the signin page to use the system.

As the password is the most crucial thing, it’s encrypted before storing in the database.

Collection Used to store information: **users**

**Sign In**



This is the entry page to the system. User must have to login to access the application.

On this page, user must insert username and password for validation.

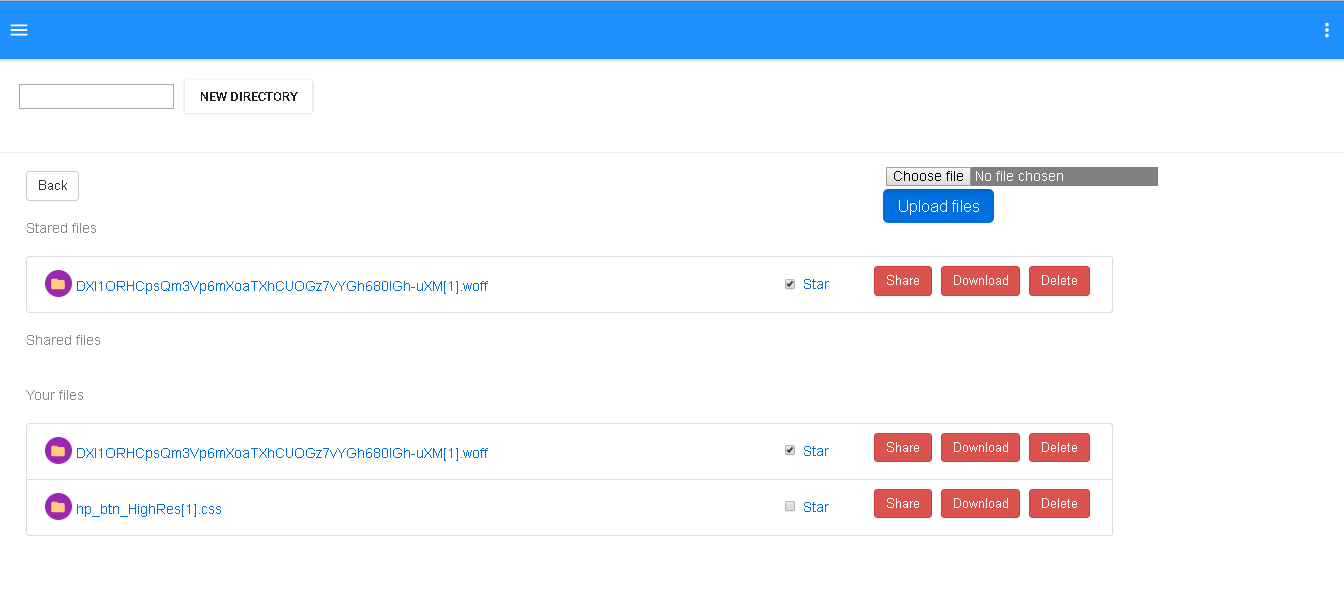
After successfully inserting the information, user clicks on sign in button and the server validates the inputs entered by the user.

Then the system matches the data with the available records in the database. If it matches, it allows the user to get into the system and use the feature.

The system then redirects the user to the welcome/home page.

Collection Used to get information: **users**

**Home Page**

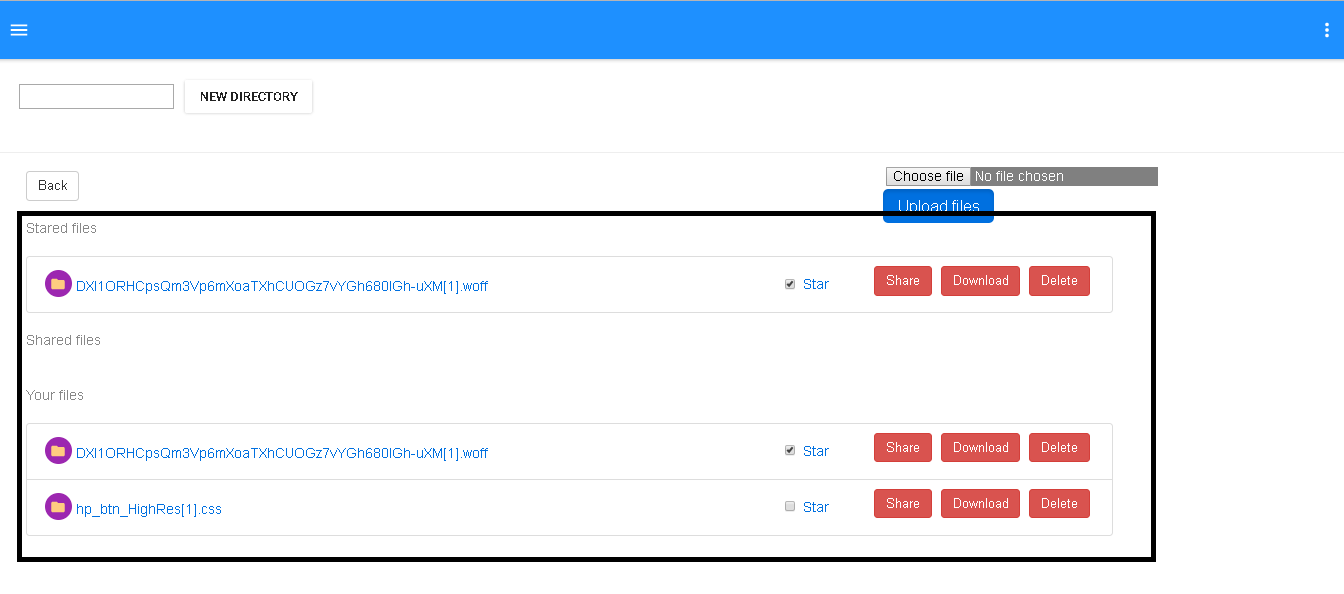


This is the page where most of the functionality of the application resides.

It contains File listing, file sharing, file download, file deletion.

It also includes directory creation and links to the user profile and user activity logs.

**File Listing**

****

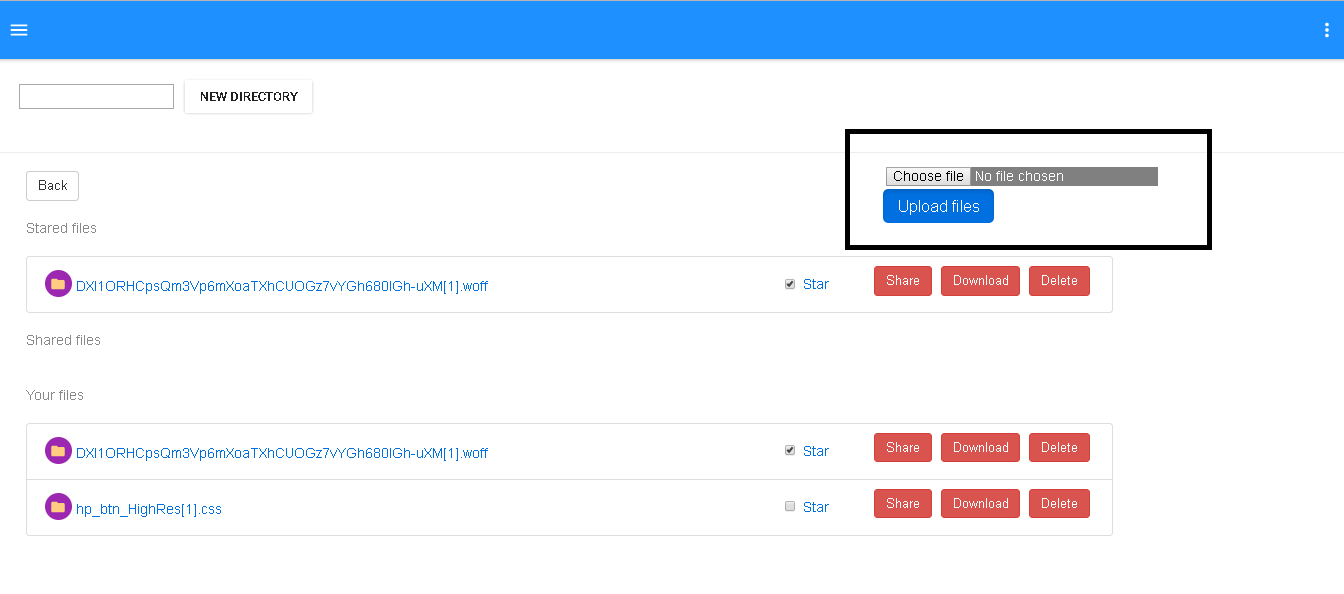
Each user has their own set of directories which they can access.

On home page, the user by default have the list of already uploaded files. There are also listing of stared files and files shared with the user.

User can also Star the file and it will be shown in the starred tab of the screen.

Collection Used to get information: **Directories, Directory\_permission, stareddir**

**Upload File**



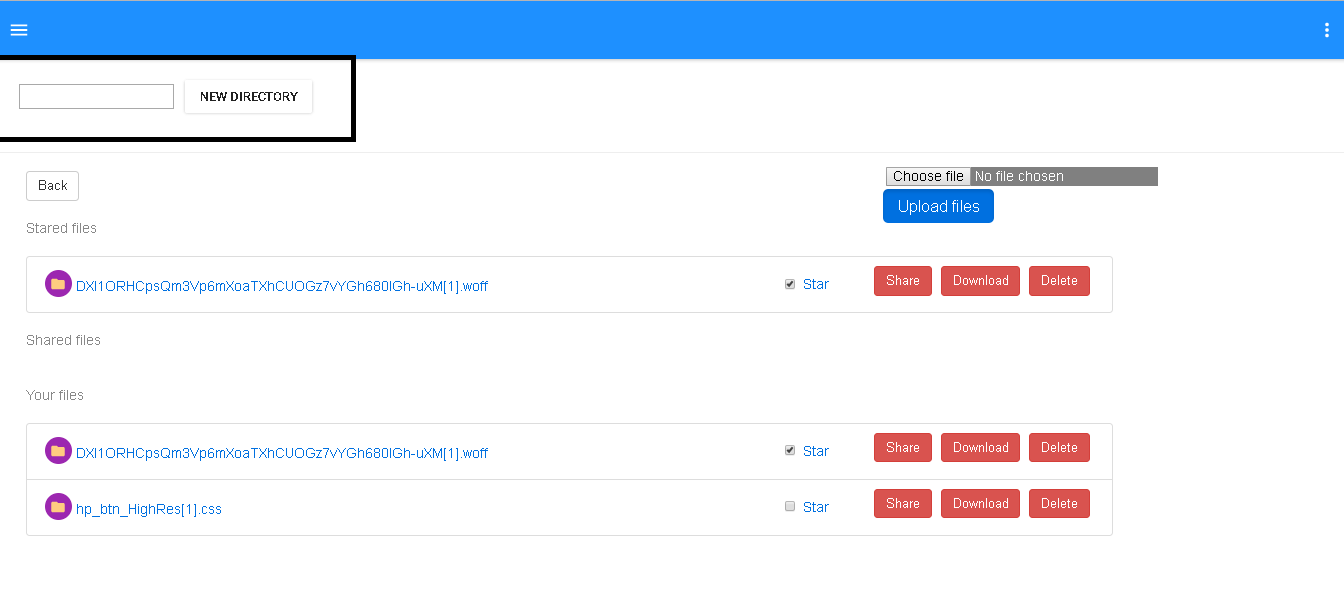
This functionality allows user to upload their own file into the system.

User can select a file from file chooser available in the screen and click on upload.

After that, the system will check the current directory to upload the file. The file will be uploaded and the same will be noted down in the DB.

Collection Used to store information: **Directories, directory\_logging**

**Create Directory**

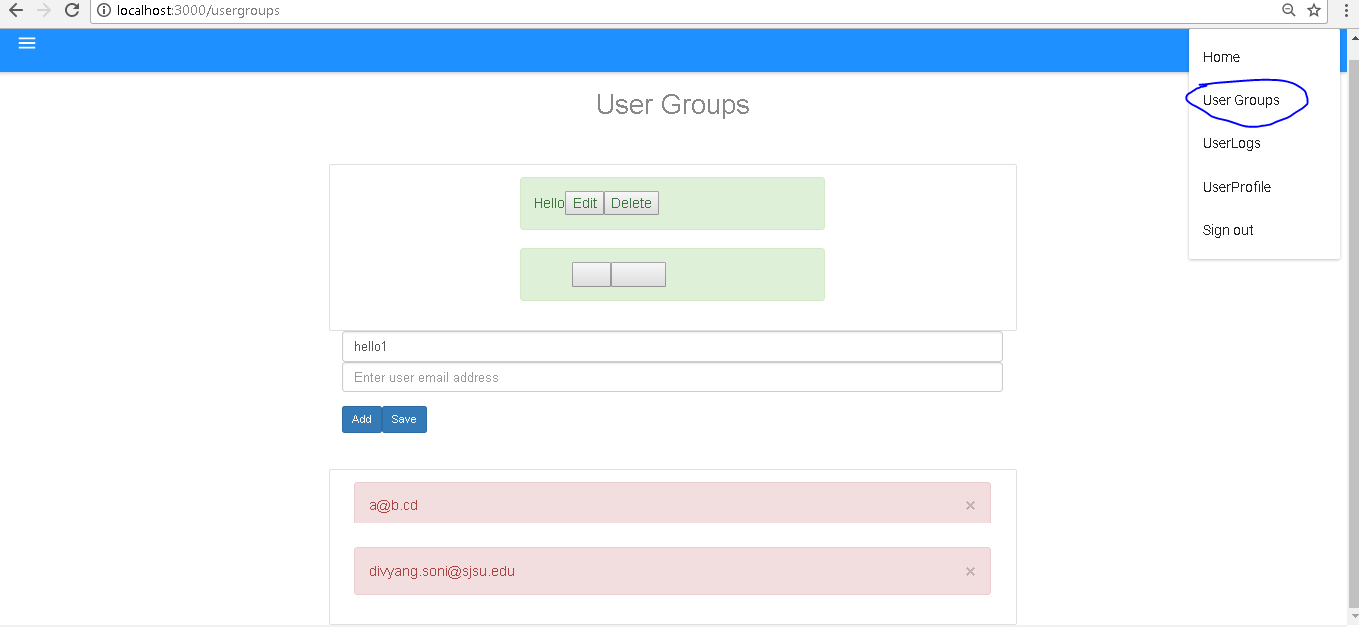


This feature allows user to create new directory.

The user can input name in the text box and clicks on the new directory button. In response, the system creates new directory and logs that in respected collections.

Collections Used to store information: **Directories, directory\_logging**

**User Groups**



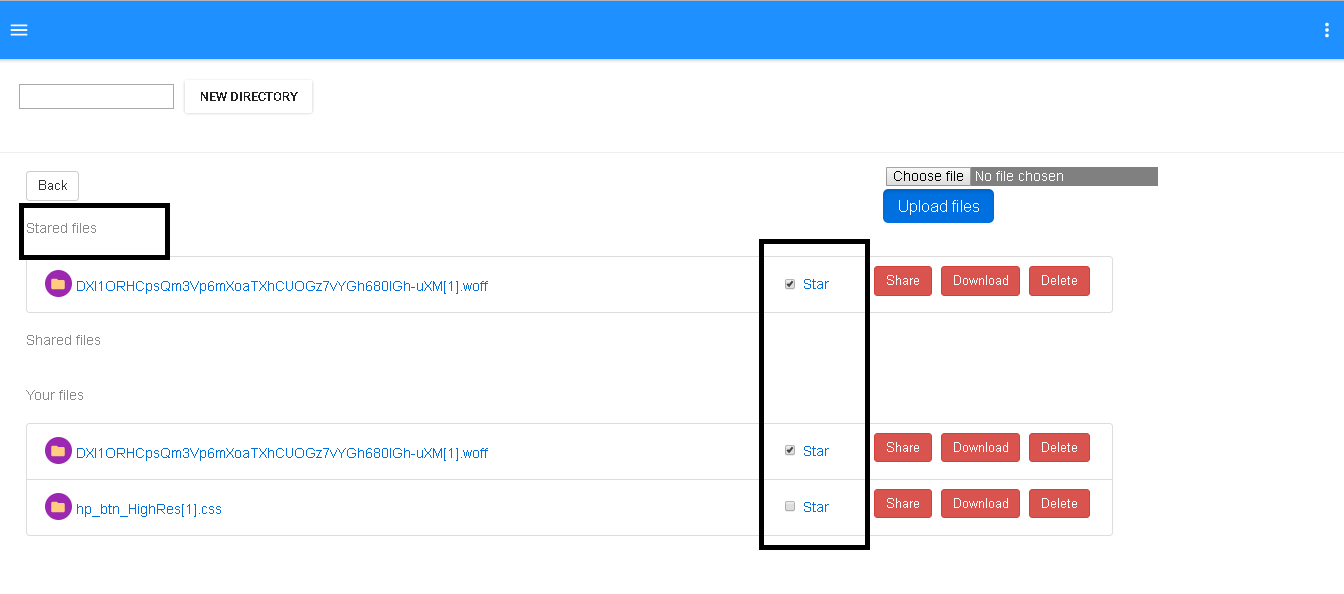
User Can make group and add users using email address.

User can use this user group to share files with other users.

User can add, edit and delete the user groups.

Document to store this data is : **Usergroup**

**Star Directory**

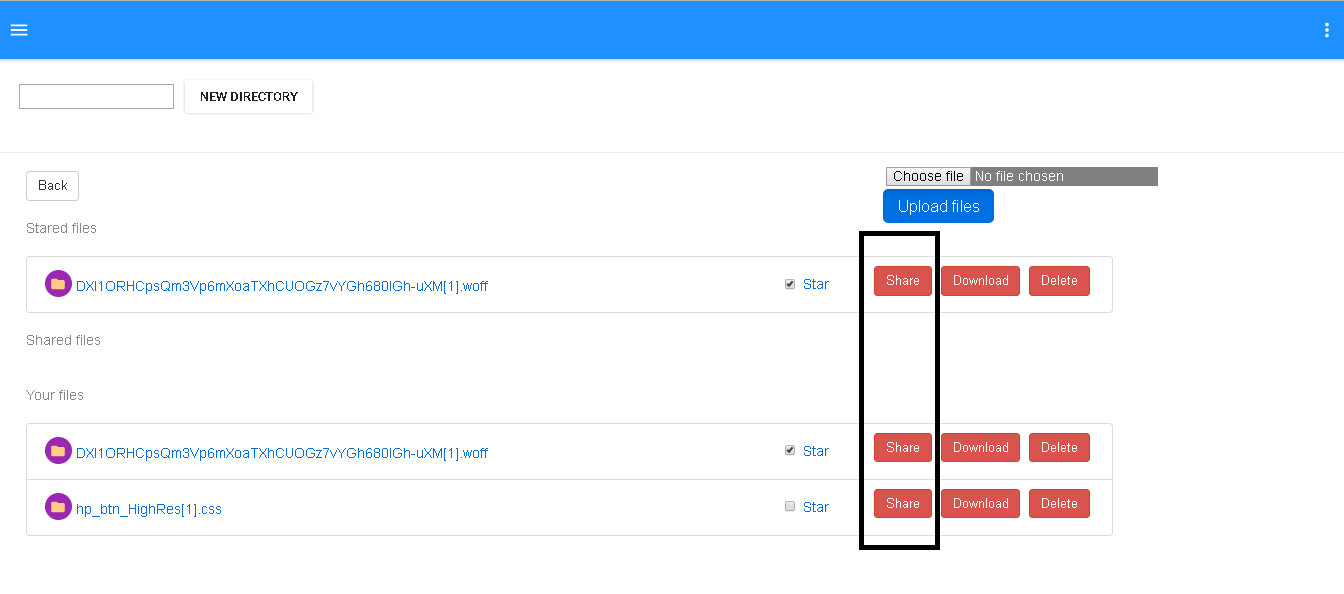


User can start a directory or file to show them always up in the stack.

The user will click on the star checkbox and the system will toggle the star status of the directory/file.

Tables Used to store information: **stardir**

**Share Directory/File**

****

The user can select the share button and a text box will pop up which allows user to enter comma separated email address. By clicking the share button, the user submits the request.

The system then decides that given email address are contains all existing user or any user outside the system.

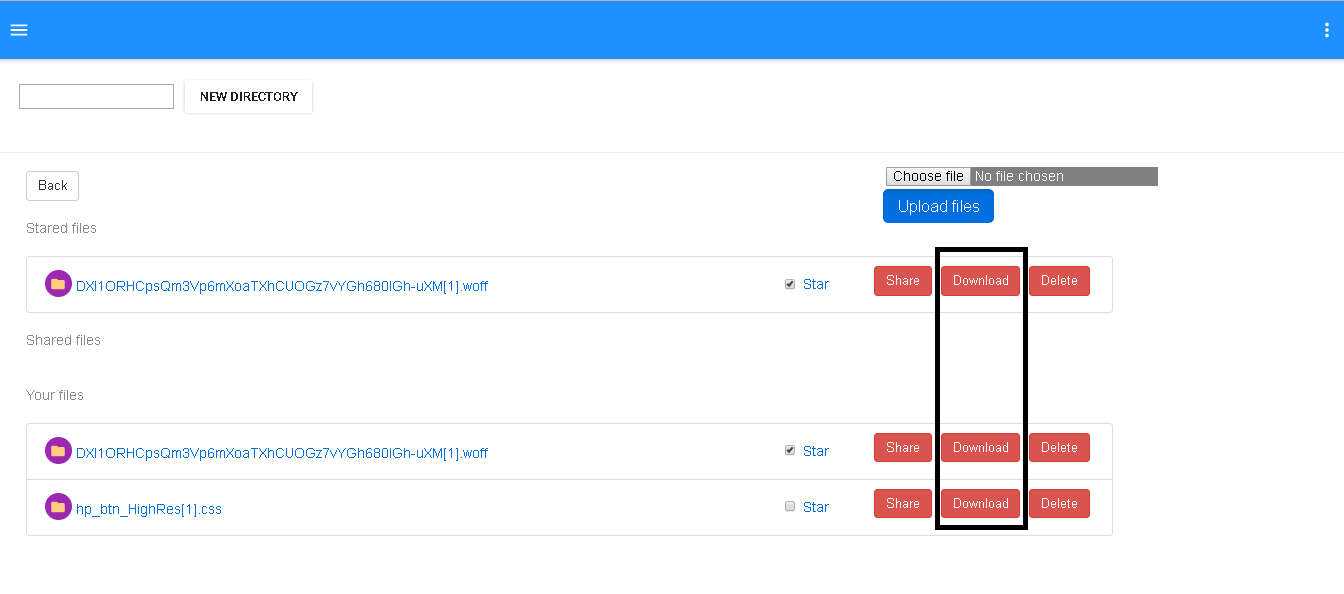
If any user is outside the system, system asks user that the sharing is going to happen by link and anyone having the link can accept it.

After having user’s consent, system shares file as link and send a message to notified users.

If all the emails are of existing customers only, the system shares file as user permission and shared users can see the file in their file list.

Tables Used to store information: **directory\_permissions**

**Download File**

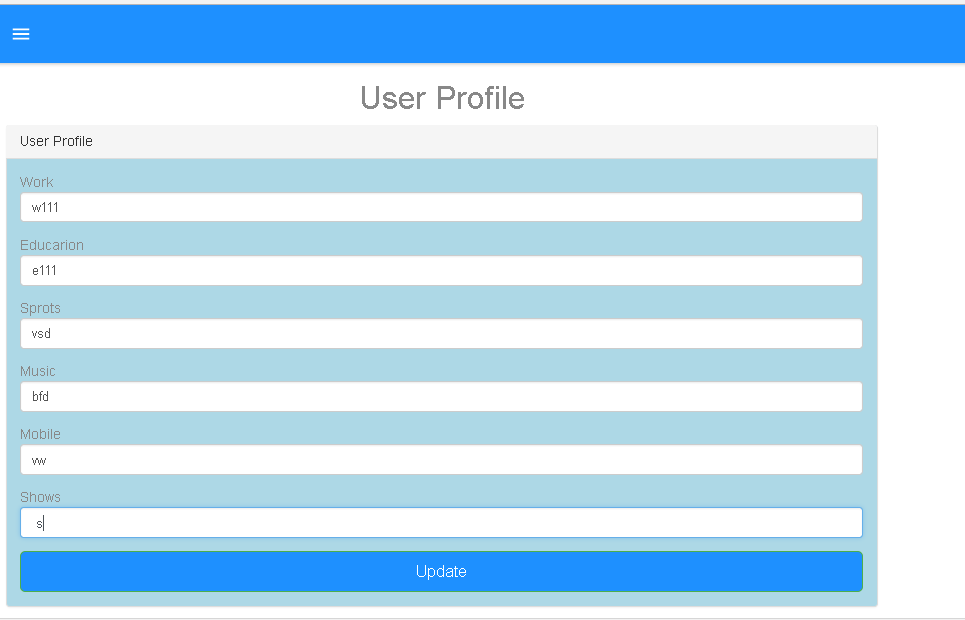
****

This functionality allows user to download file.

User will just click on the download button and the system will download the file on the browser.

The user needs to allow the browser to save the data and it will be automatically downloaded.

**User Profile**



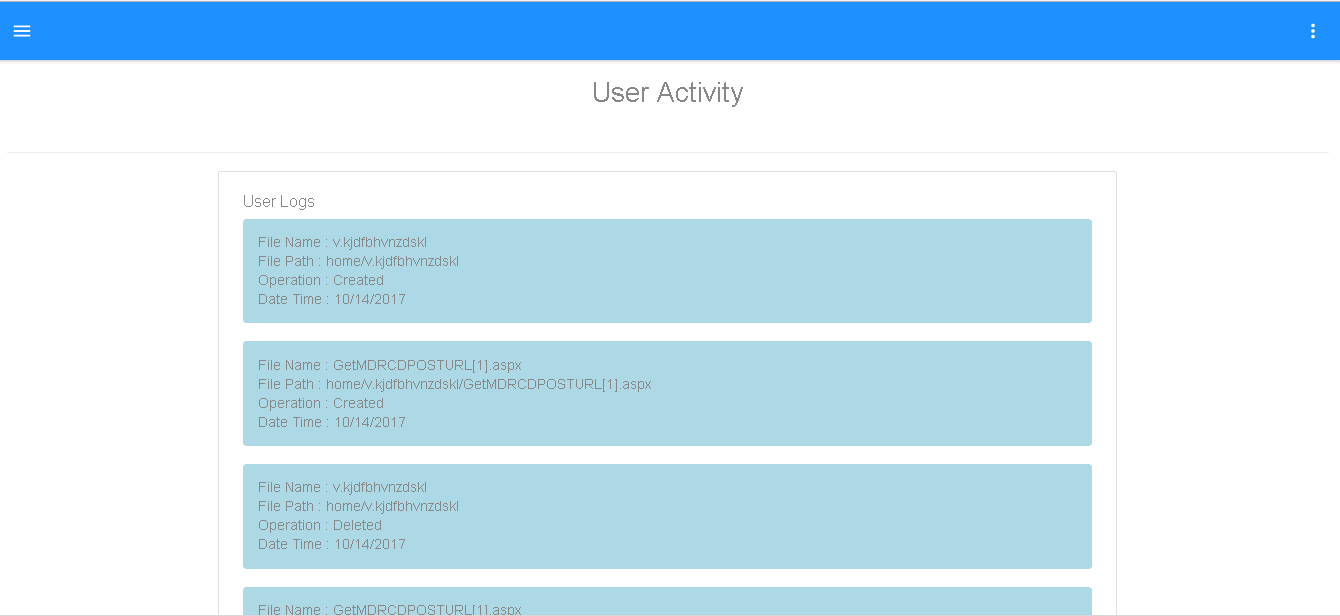
This feature allows users to add/update their user profile which contains user’s extra information.

Users will enter the details in the text boxes and clicks on the update button.

The system then stores data into the system and repopulate it on the screen.

Tables Used to store information: **userprofile**

**User Activity Logs**

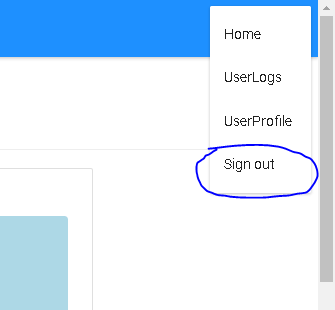


This functionality is used to show user’s their activity log on the files.

It shows creating directory, delete directory, upload file in directory, delete file

Tables Used to store information: **directory\_logging**

**Sign Out**



This functionality allows users to securely letting user to logout from the system.

**Performance**

After running the system, performance being measured is as follow.

1. The system is responding every request within 1 seconds.
2. The System can fetch data from database in efficient and faster way.
3. The System can handle large number of requests without getting crashed.
4. System is not allowing users to get data without logging in
5. UI input/listings are instantly reflected in UI without delay.
6. System is changing necessary part of DOM instead of refreshing whole page
7. Crucial data are safe and not shown on the UI
8. UI is always free for the user to perform any task as all the activities are asynchronized
9. No connection leakage found while running the application.

Why This Performance?

**Apache Kafka** is used to maintain the message queue which allow the system to accept thousands of requests together as server is just accepting the request and putting it into kafka message queue. So actual business logic is written in the kafka backend server which can be replicated according to our use.

**React JS** is used to **prevent unnecessary refreshing of pages**. Instead of that it only renders the necessary part of the DOM.

The **UI never freeze** and allows user to interact with other functionalities due to **unidirectional** flow of React JS.

**Connection Pooling** is used to decrease time of connection creation and closing, Which improves the response time of the system.

**Database Util** is written to make all database related activities in one file. No other files is allowed to create its own connection. Due to that connection management is made really easy and all the connection related enhancement can be done in one file itself instead of in multiple files.

**API calls** are **combined** to minimize the server calls and make code more readable.

As node JS in available in backend, the **callback functionality** is used to make the calls **asynchronous** when needed.

**Small methods** are implemented to perform small tasks to **remove code redundancy** **and increase code reusability**.

**Q1)** **Compare passport authentication process with the authentication process used in Lab1.**

Ans: In lab 1, we have used client session for the login authentication and for which the session was stored only in the server. Due to that the session was not horizontally scalable as each server have to have its own copy of session which makes session storing a heavy process.

While in passport js, it automatically sessions in the database, for us its mongo db. So, every server doesn’t need to store server explicitly. Servers can read the session from the database and use them.

Passport employs a strategy design pattern to define a clear separation of concerns between the core module and various authentication mechanisms. This has many benefits, including smaller overall code size and well defined and testable interfaces.

When it comes to security, passport js in more secure than authentication without any middleware as it provides encryption to your data.

Passport JS is providing inbuilt unit tests to validate the responses which provides us more assurance of reliability.

Also, we can use the passport js to validate any request coming in the application.

**Q2) Compare performance with and without Kafka. Explain in detail the reason for difference in performance.**

As Kafka message queue is implemented in our project, it is not capable to handle huge requests.   
Unfortunately, we didn’t have multiple servers to test our application and so we cannot see much difference in with and without kafka application but yes what we observed is that if we implement kafka in our application, the throughput is increases for large amount of request.

For without kafka application, the throughput with 500 calls was nearly 355 and if we test it with kafka it shows 275 as throughput.

If we can implement more servers to handle kafka messages, we can see a drastic change in our through. A study shows that if we just add 1 more server in will increase our throughput nearly by double.

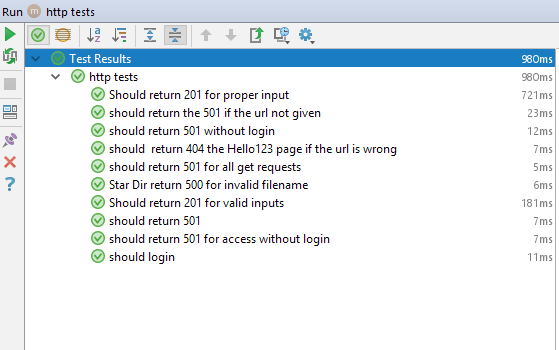
Also, we can change our throughput by implementing different kind of message behavior. I.e. We have implement multiple topics for multiple functionalities or we can define one single topic and use keyed messages, or we can implement a mixed approach and also we can have different servers which listens on single topic.

**Q3) 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.**

If I have been given an option to use both mysql and mongo db in my project, I would like

* Store my user related data in Mysql which includes my login information and user profile. As user data are more crucial, we can choose reliability over throughput.
* Store my file/directory related core in mysql which will includes directory/file name, path and parent file id
* I will store my user sharing related data in mongo db as it is going to be a huge data. User can share file with n number of users and also it can be unshared. So data is going to be huge for share related files.
* I will also store my user logs in mongo db as multiple user can do multiple activities on file and we just need that data to show on the UI and no other physical relation is needed in those data.
* I will store staring related data also in mongo db as it is again going to be multiple staring for one file if its shared with multiple users.

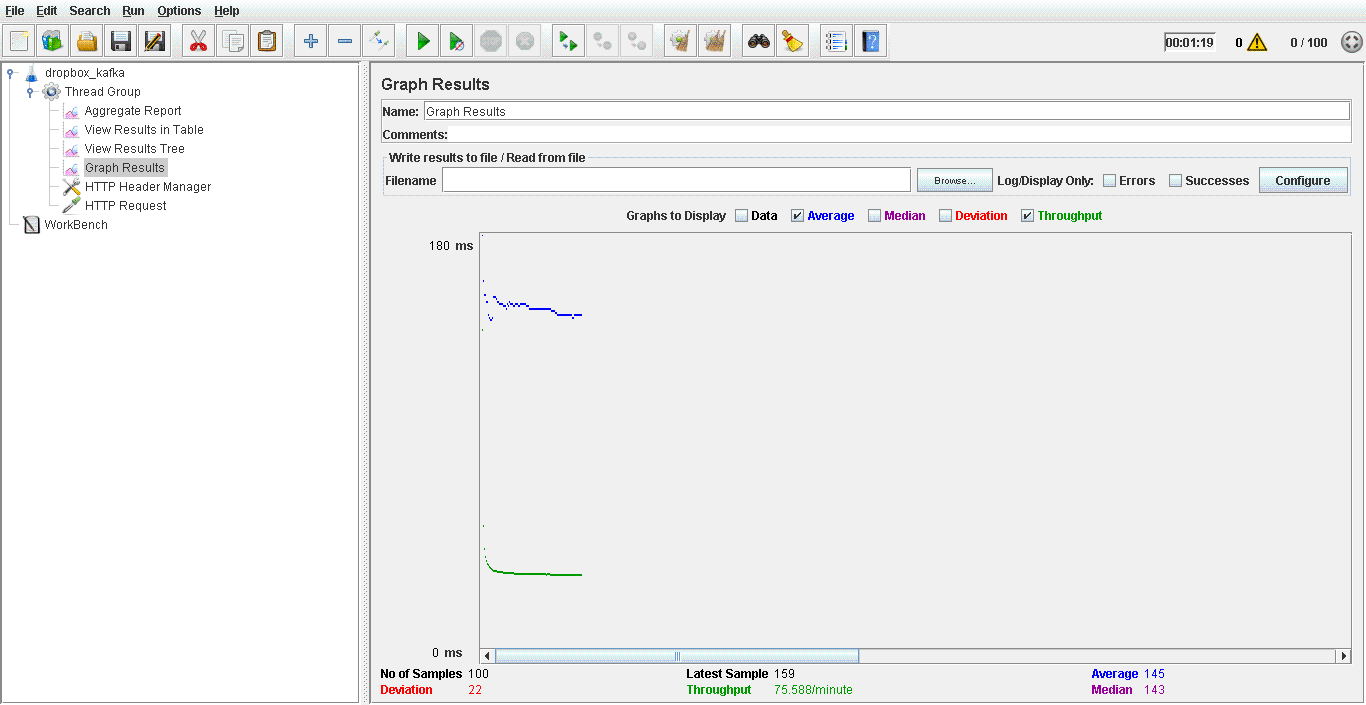
**Mocha Testing**



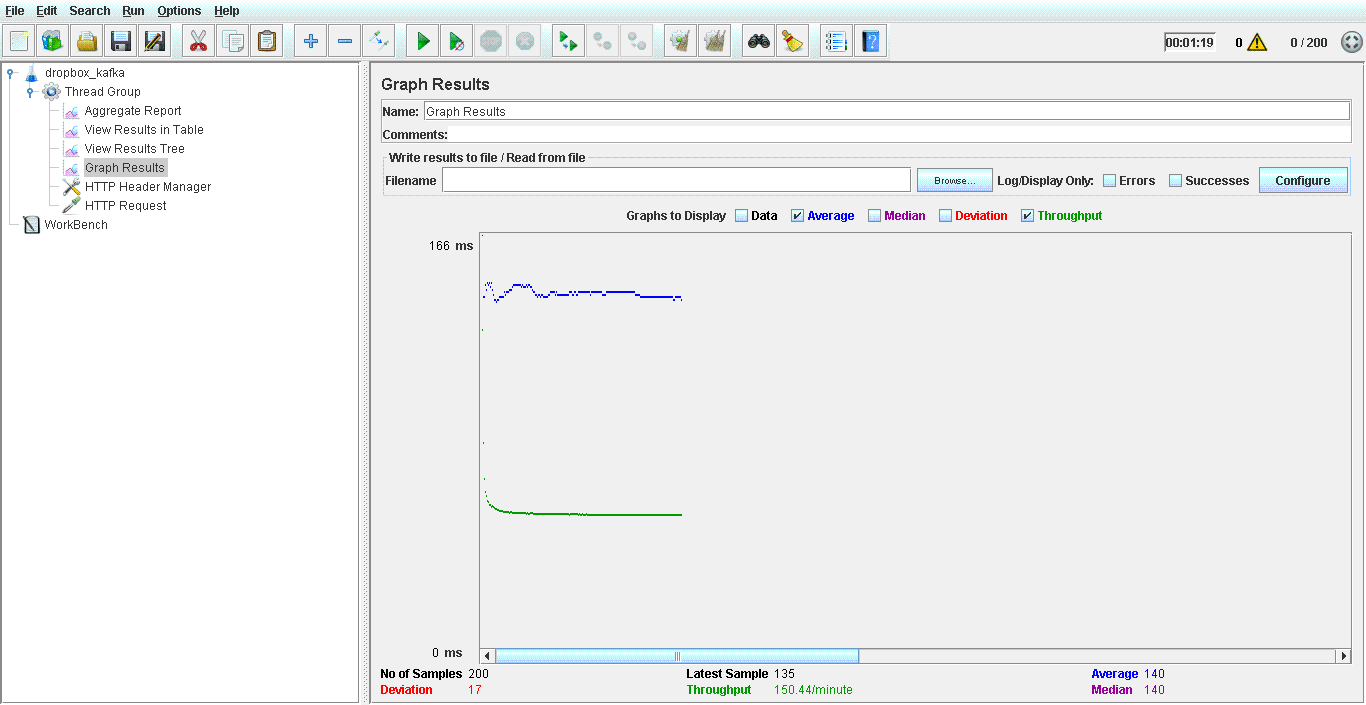
**Jmeter Testing for DropBox (Restfull API)**

1. **Without Connection Pooling**

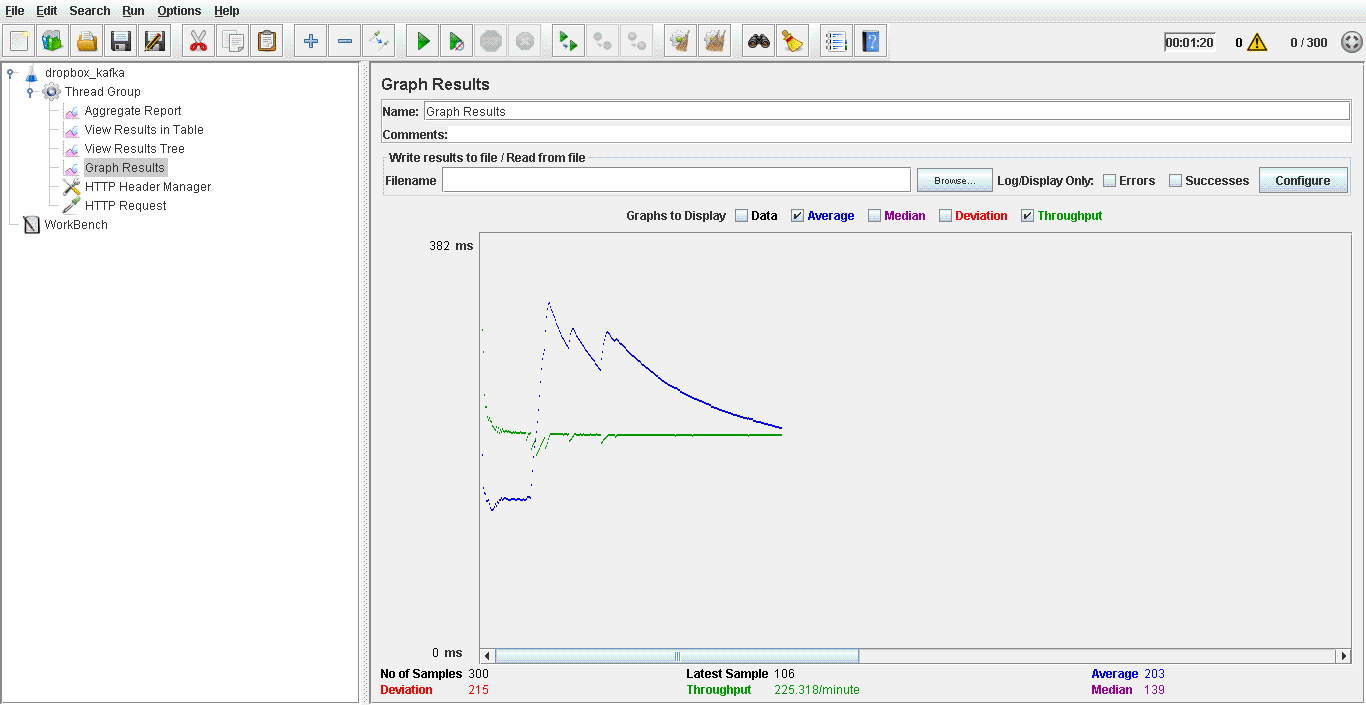
**100 users**



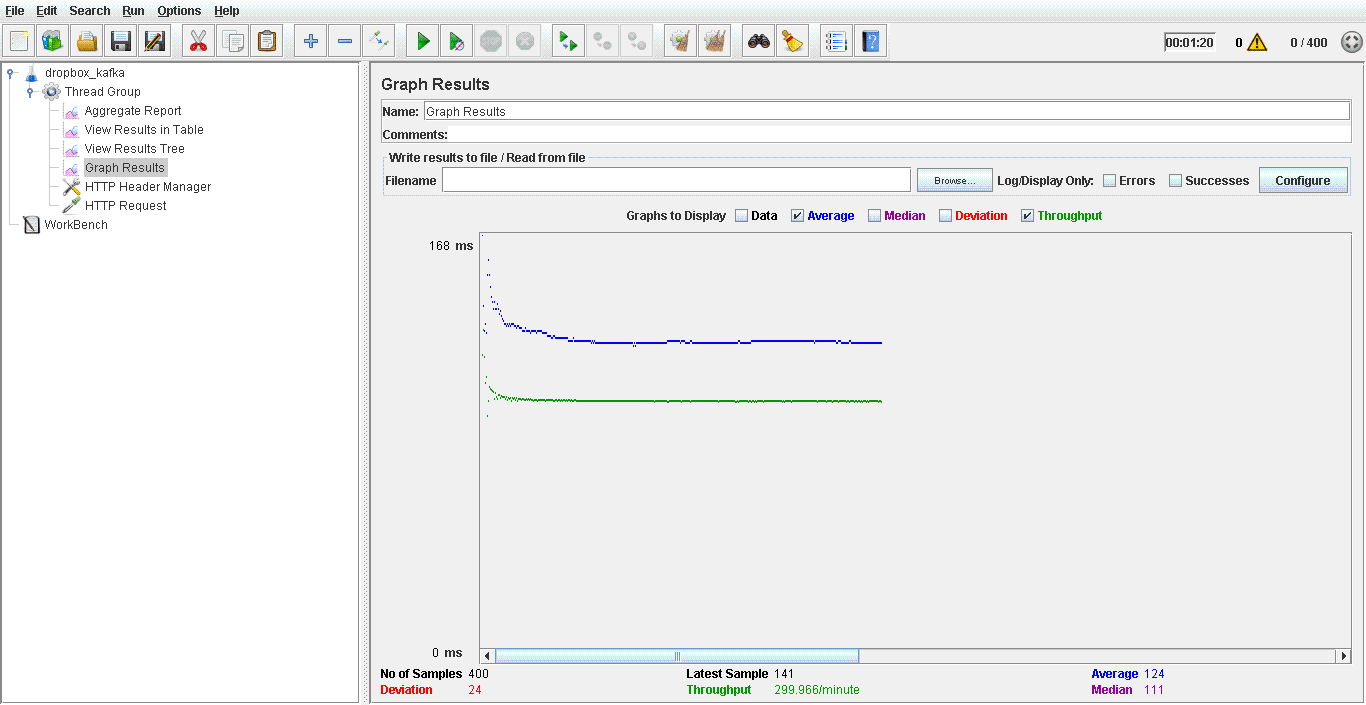
**200 users**



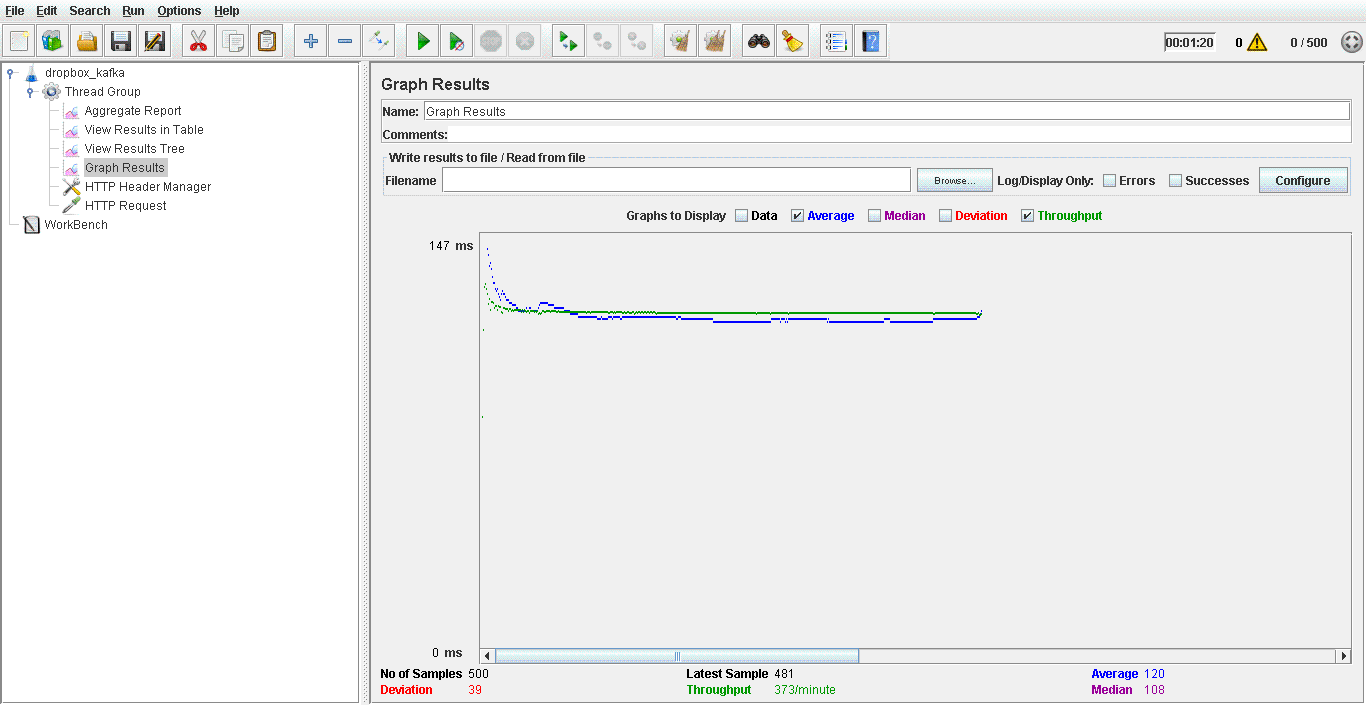
**300 users**



**400 users**

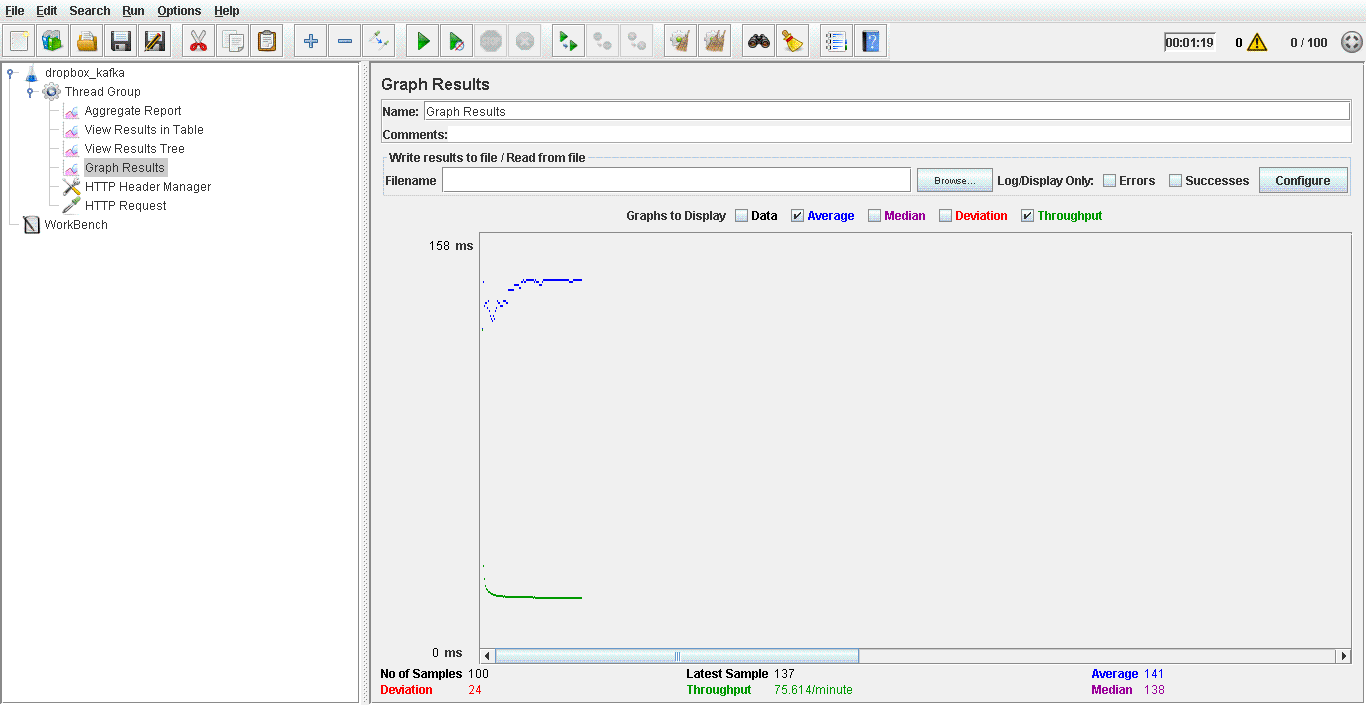


**500 users**

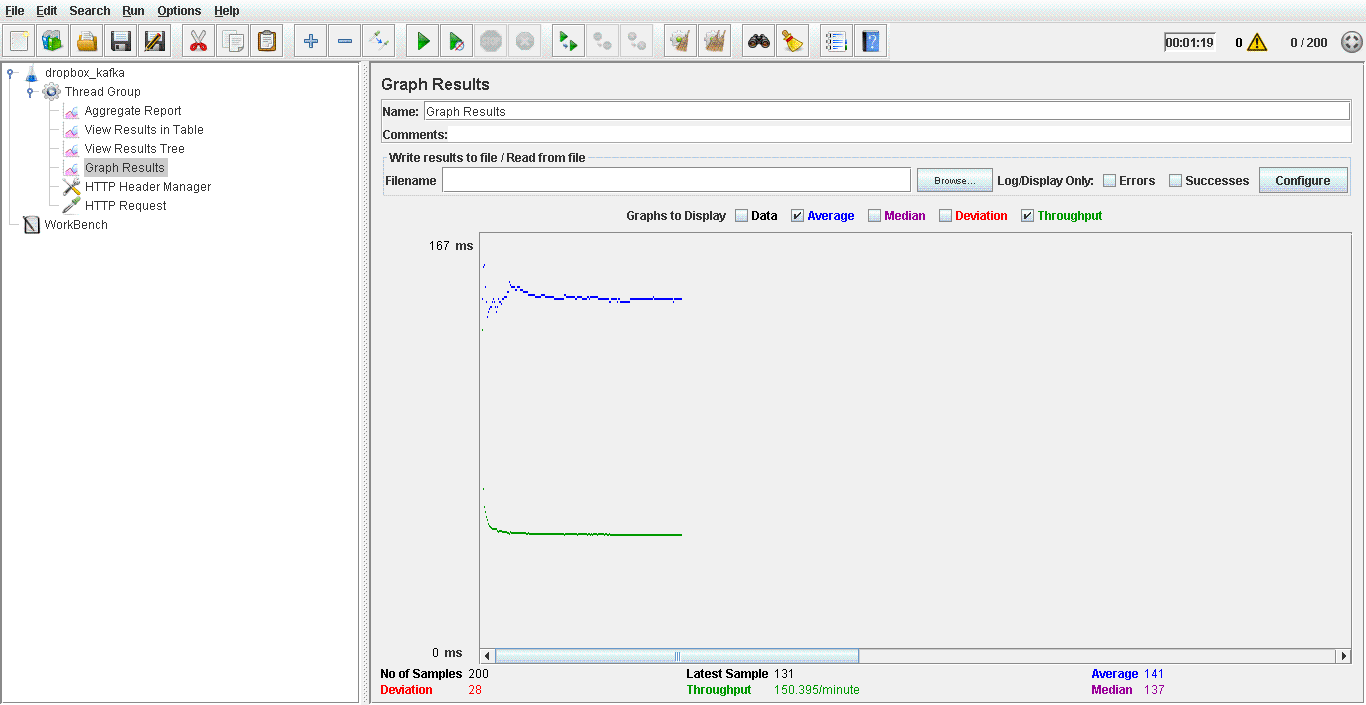


1. **With Database Provided Connection Pooling**

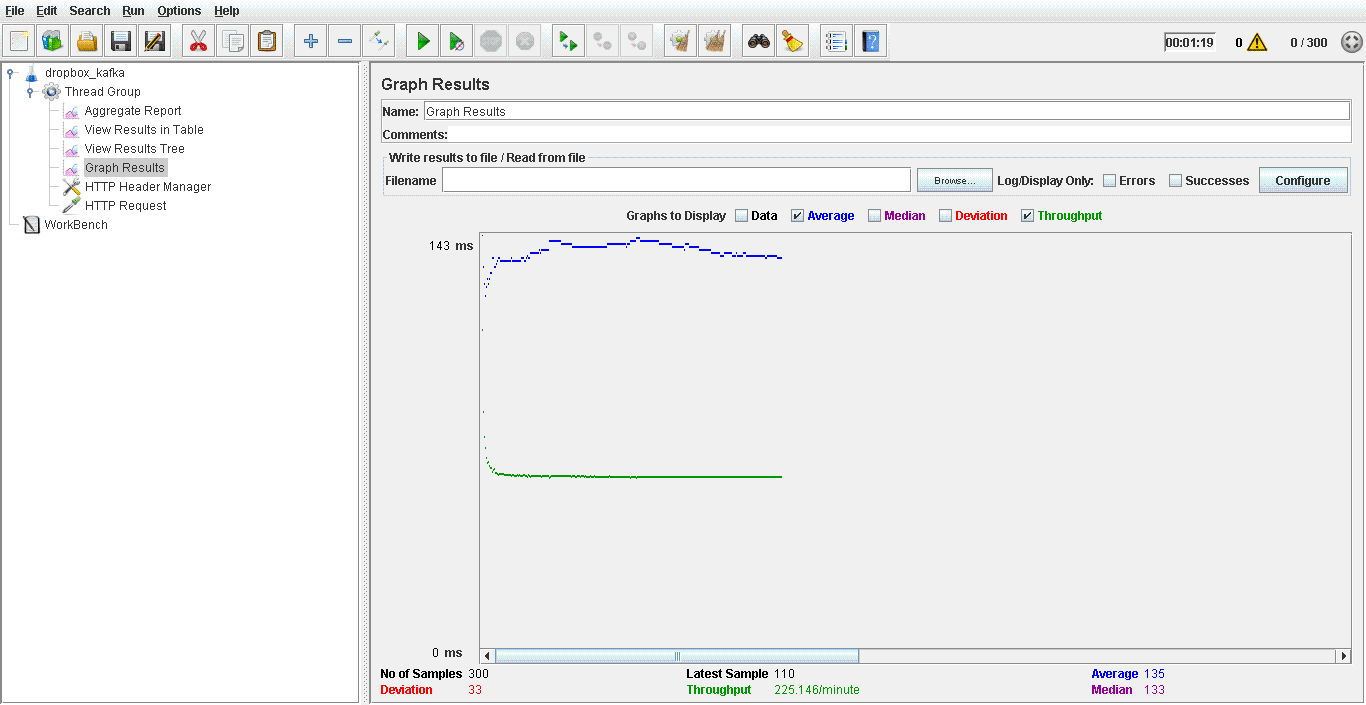
**100 users**



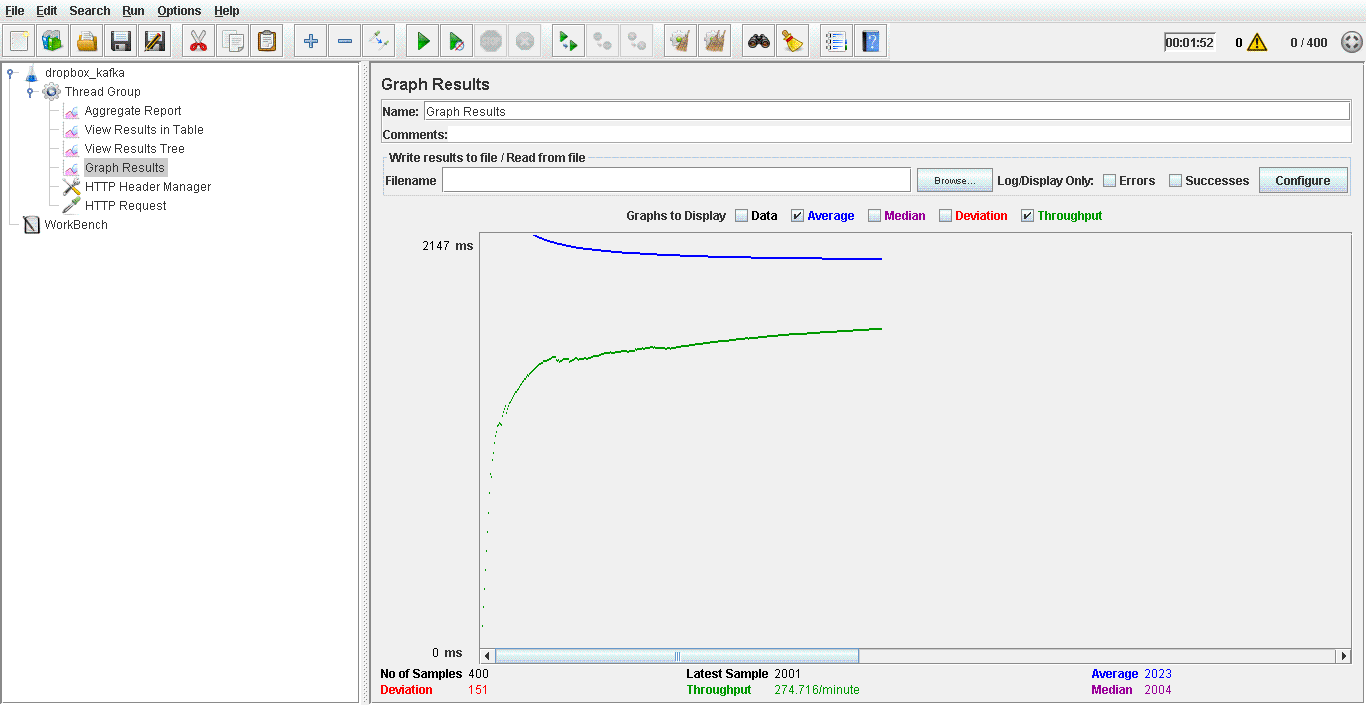
**200users**



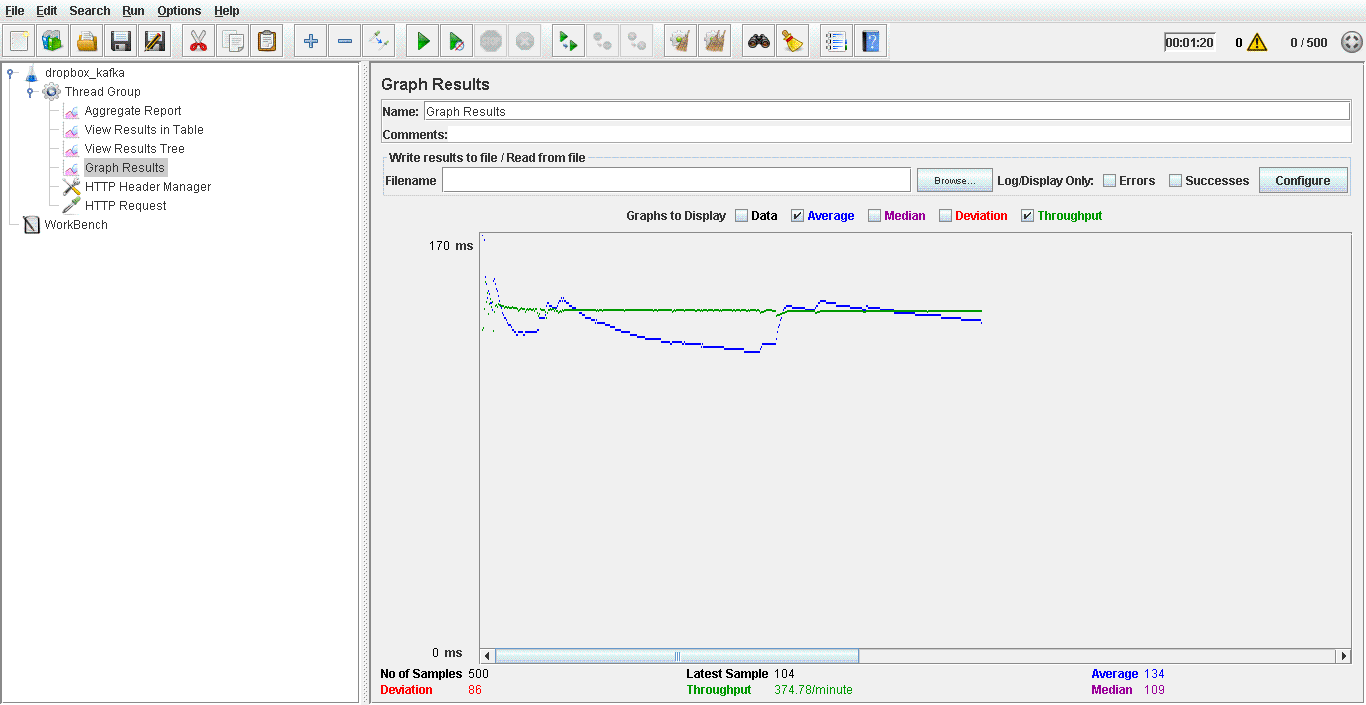
**300 users**



**400 users**



**500 users**



**With Custom Connection Pulling:**