

# Lab1

GitHub Link:- <https://github.com/ShreyBhatt/EbayLab1>

## Part 1: Calculator

### **Introduction**

### **Goal**

The goal of this Calculator application to demonstrate stateless web services.

### **Purpose & Basic functionalities**

In this calculator application user can do operations like;

1. Addition
2. Subtraction
3. Division (Divide a number by 0 will display result as INFINITY, 0 divide by 0 is UNDEFINED)
4. Multiplication

### **System Design**

The calculator application is created using Html, angular.js,bootstrap,node.js and Express.js.

There are two textboxes in which user can enter values.

There are 4 different buttons for addition, subtraction, division and multiplication.  
When user enters 2 numbers and click on any of the button, the result will be shown in the span element.

The server-side is made using node.js and express.js.

The client side is made using Angular.js, bootstrap and Html.

## Results:

Client of Calculator

ADD function is shown below:

Calculator

Enter Number 1= 32

Enter Number 2= 5

37

ADD SUB MUL DIV

Any of the 2 values can't be NaN:

Calculator

Enter Number 1= 32

Enter Number 2=

Enter both the Number Values

ADD SUB MUL DIV

Divide by 0 will display result as INFINITY:

Calculator

Enter Number 1= 32

Enter Number 2= 0

INFINITY

ADD SUB MUL DIV

Divide 0 by 0 will display result as UNDEFINED:

Calculator

Enter Number 1= 0

Enter Number 2= 0

UNDEFINED

ADD SUB MUL DIV

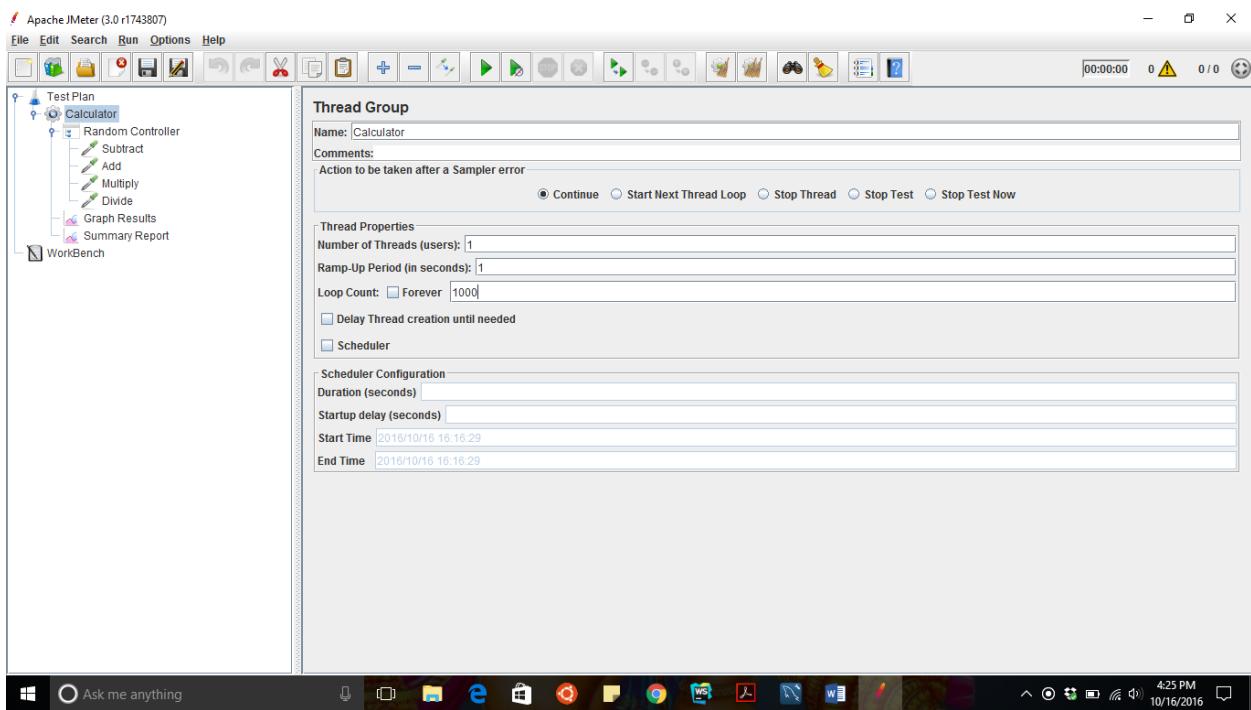
## Server of Calculator



```
"C:\Program Files (x86)\JetBrains\WebStorm 2016.2.2\bin\runnerw.exe" "C:\Program Files\nodejs\node.exe" C:\Users\Shrey\WebstormProjects\CMPE273_Calculator\bin\www
CMPE273 Calculator:server Listening on port 3000 +0ms
GET /calc 200 8.957 ms - -
GET /angularjs/front.js 304 2.749 ms - -
POST /calculate 200 26.228 ms - -
```

## Testing & Performance Graphs

### Jmeter Setup of HTTP Request



## 1) 1000 calls

Calculator.jmx (C:\Users\Shrey\Desktop\Calculator.jmx) - Apache JMeter (3.0 r1743807)

File Edit Search Run Options Help

Test Plan

- Calculator
  - Random Controller
    - Subtract
    - Add
    - Multiply
    - Divide
  - Graph Results
  - Summary Report
- WorkBench

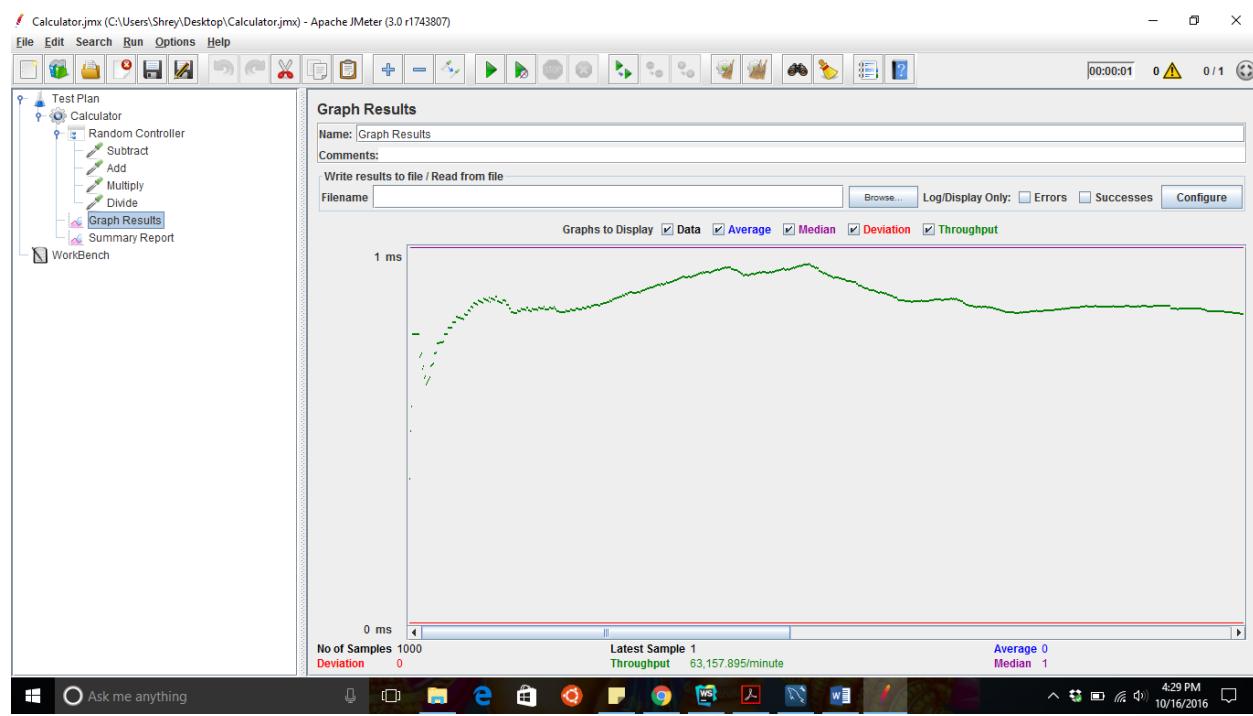
Summary Report

Name: Summary Report  
Comments:  
Write results to file / Read from file  
Filename  Browse... Log/Display Only:  Errors  Successes  Configure

Label	# Samples	Average	Min	Max	Std. Dev	Error %	Throughput	KB/sec	Avg. Bytes
Subtract	229	0	0	8	0.78	0.00%	241.3/sec	39.35	167.0
Multiply	259	0	0	3	0.58	0.00%	273.8/sec	44.92	168.0
Add	250	0	0	4	0.65	0.00%	265.4/sec	43.28	167.0
Divide	262	0	0	3	0.55	0.00%	277.5/sec	45.26	167.0
TOTAL	1000	0	0	8	0.64	0.00%	1052.6/sec	171.94	167.3

Include group name in label?  Save Table Data  Save Table Header

Ask me anything 4:28 PM 10/16/2016



## 2) 5000 calls

Calculator.jmx (C:\Users\Shrey\Desktop\Calculator.jmx) - Apache JMeter (3.0 r1743807)

File Edit Search Run Options Help

Test Plan  
Calculator  
Random Controller  
Subtract  
Add  
Multiply  
Divide  
Graph Results  
Summary Report

WorkBench

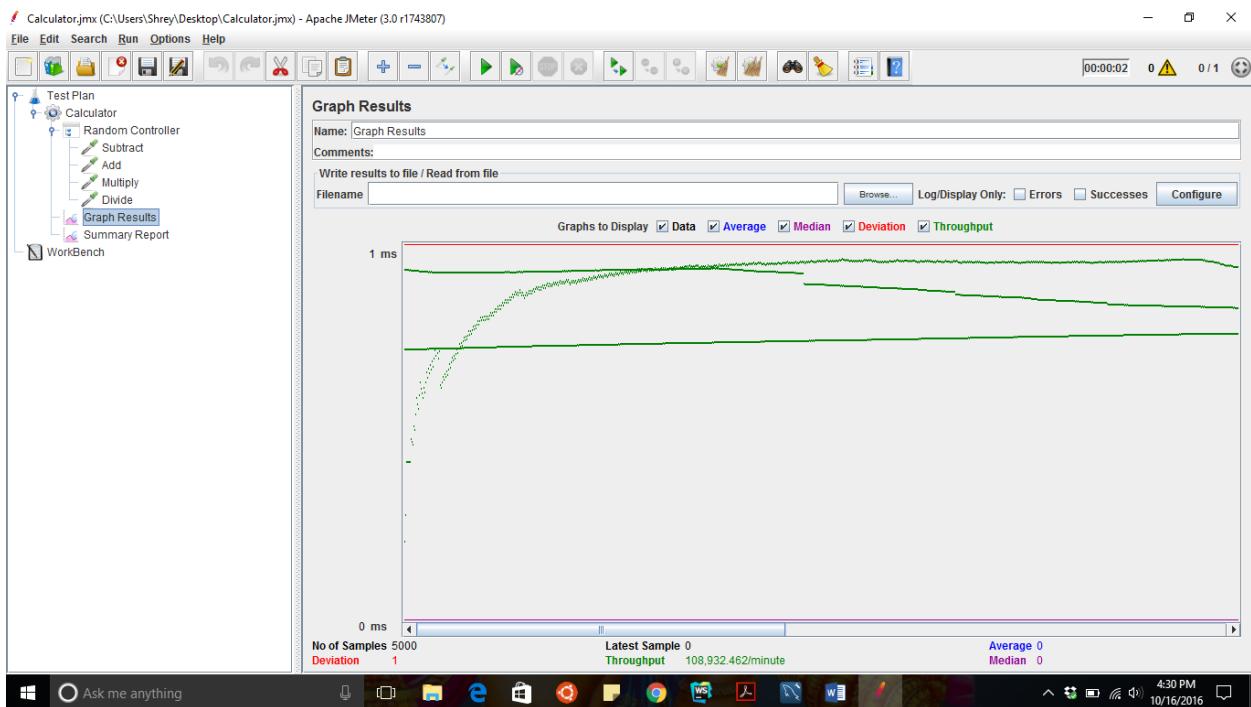
Summary Report

Name: Summary Report  
Comments:  
Write results to file / Read from file  
Filename  Browse... Log/Display Only:  Errors  Successes

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	KB/sec	Avg. Bytes
Subtract	1243	0	0	34	1.09	0.00%	451.5/sec	73.63	167.0
Divide	1257	0	0	30	1.00	0.00%	456.8/sec	74.49	167.0
Multiply	1270	0	0	72	2.13	0.00%	462.0/sec	75.79	168.0
Add	1230	0	0	25	1.04	0.00%	448.4/sec	73.13	167.0
TOTAL	5000	0	0	72	1.41	0.00%	1815.5/sec	296.54	167.3

Include group name in label?   Save Table Header

Ask me anything 4:29 PM 10/16/2016



### 3) 1000 calls on 100 concurrent users

Calculator.jmx (C:\Users\Shrey\Desktop\Calculator.jmx) - Apache JMeter (3.0 r1743807)

File Edit Search Run Options Help

Test Plan  
 Calculator  
 Random Controller  
 Subtract  
 Add  
 Multiply  
 Divide  
 Graph Results  
 Summary Report

WorkBench

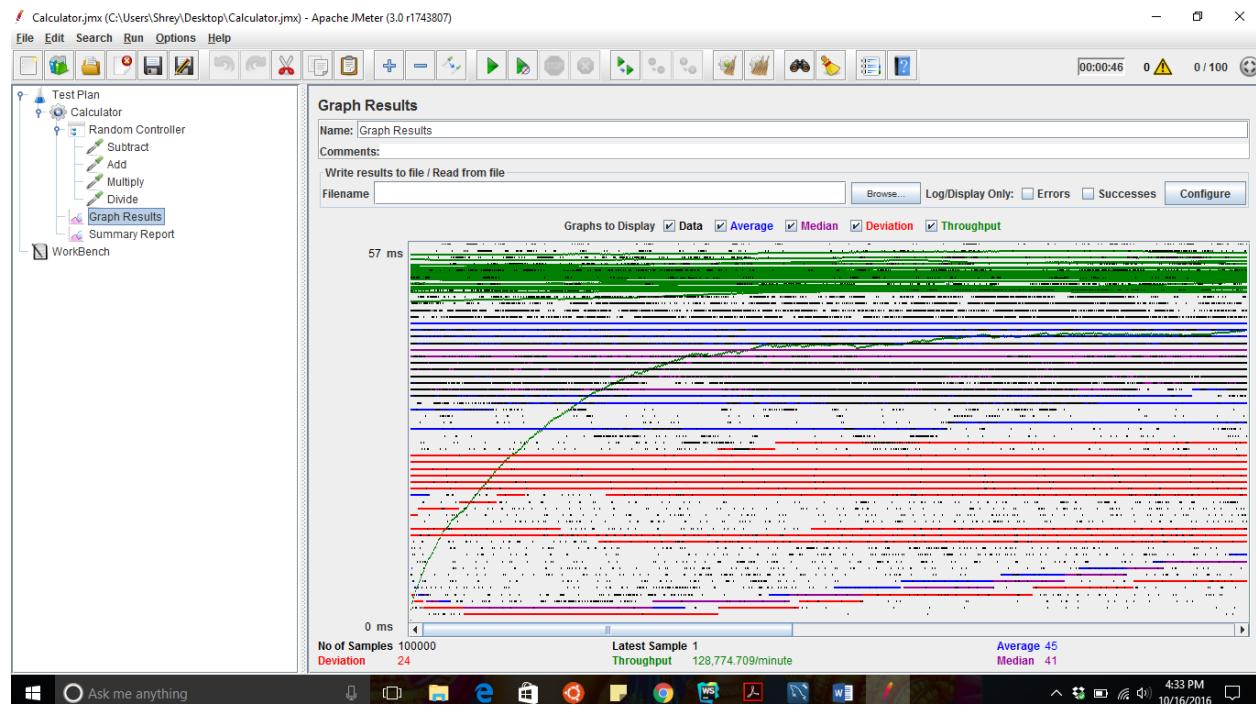
**Summary Report**

Name: Summary Report  
 Comments:  
 Write results to file / Read from file  
 Filename  Browse... Log/Display Only:  Errors  Successes  Configure

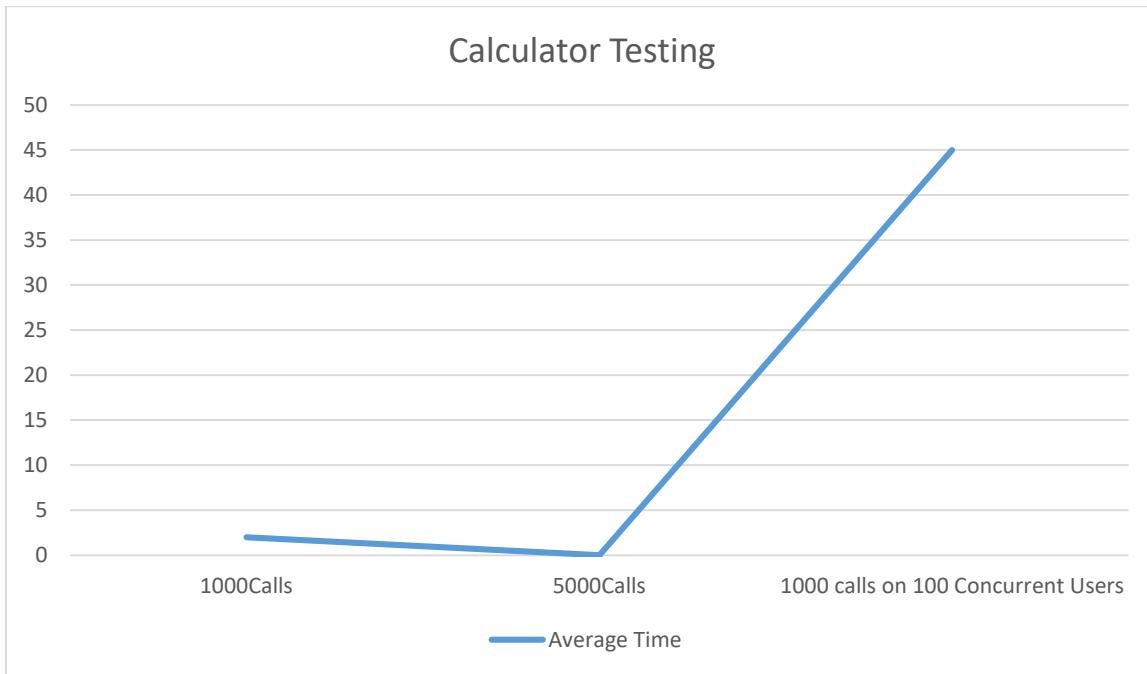
Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	KB/sec	Avg. Bytes
Divide	25011	45	0	315	24.98	0.00%	536.8/sec	87.55	167.0
Add	25082	45	1	341	25.06	0.00%	538.1/sec	87.76	167.0
Multiply	25134	44	1	333	24.23	0.00%	539.6/sec	88.54	168.0
Subtract	24793	45	1	342	24.44	0.00%	532.5/sec	88.84	167.0
TOTAL	100000	45	0	342	24.68	0.00%	2146.2/sec	350.55	167.3

Include group name in label?  Save Table Data  Save Table Header

4:33 PM 10/16/2016



## Analysis Graph



This graph shows that when we test the application with multiple concurrent users the average time increases.

### Reason:-

Average Response time refers to the average amount of time Server takes to return the results of a request to the user. The Average response time is directly proportional to the number of users and number of requests submitted.

Therefore the average response time increases when we test the application with multiple requests by multiple concurrent users.

## **Part 2 : Ebay Marketplace Application**

### **Introduction**

The Ebay Marketplace Application is the replica of Ebay used to demonstrate REST web services

### **Goal &Purpose**

- Goal of this application is to demonstrate REST web services.

### **Basic Functionalities**

1. Basic User functionalities:
  - a. Sign up the new users (First name, Last name, Email, Password)
  - b. Sign in with existing users
  - c. Sign out
2. Profile:  
About: Birthday, Ebay handle, contact information and location

3. User's advertisements for others to read. Which includes the item name, item description, seller information, item price and quantity

4. Selling Functionality

5. Bidding Functionality

6. Shopping Cart functionality

7. User tracking functionality

8. Credit card validations on payment.

9. Order History Functionality

## **System Design**

There are 14 pages in the entire application

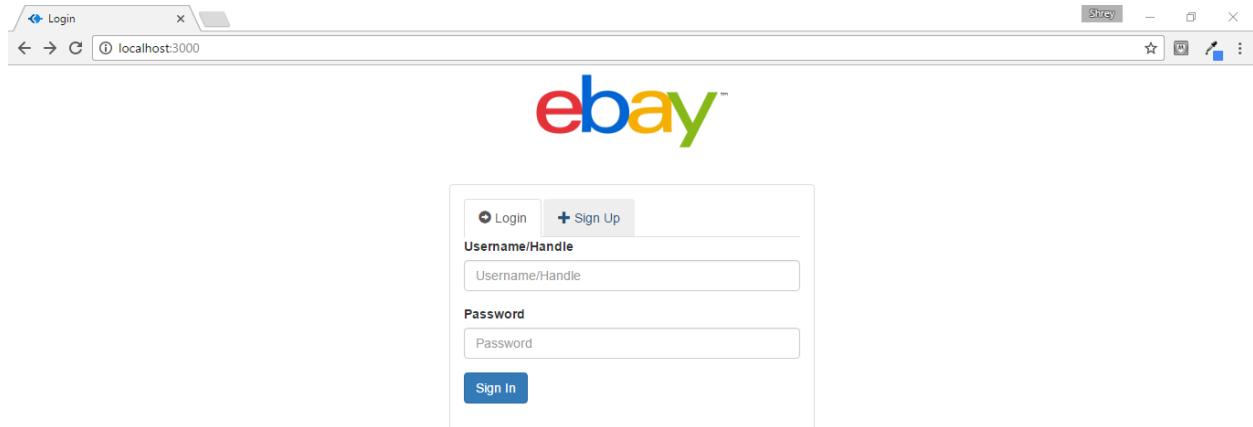
1. Main page which will show the login and signup options.
2. Home page which will be shown only after successful login. It will display All Advertisements which are posted by other users.
3. All bids page
4. Profile page which shows further details of the user.
5. Advertisement posting page
6. Bid posting page
7. Product Detail page
8. Bid Product Detail Page
9. My advertisement page which will show advertisement posted by us
10. My bids page which will show bid posted by us
11. Shopping cart page
12. Last Login page
13. My orders page
14. Credit card Validation Page on checkout

The header contains a navigation bar which has options of shopping cart, create account and signout.

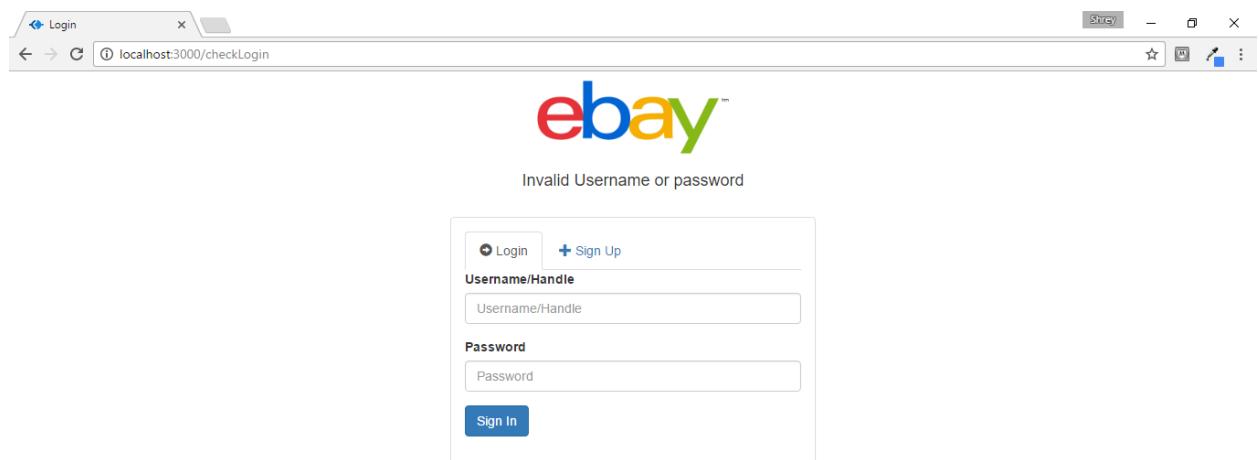
- Front End: Angular.js, Bootstrap, HTML
- Back End: Node.js
- Database: MySQL

## **Results**

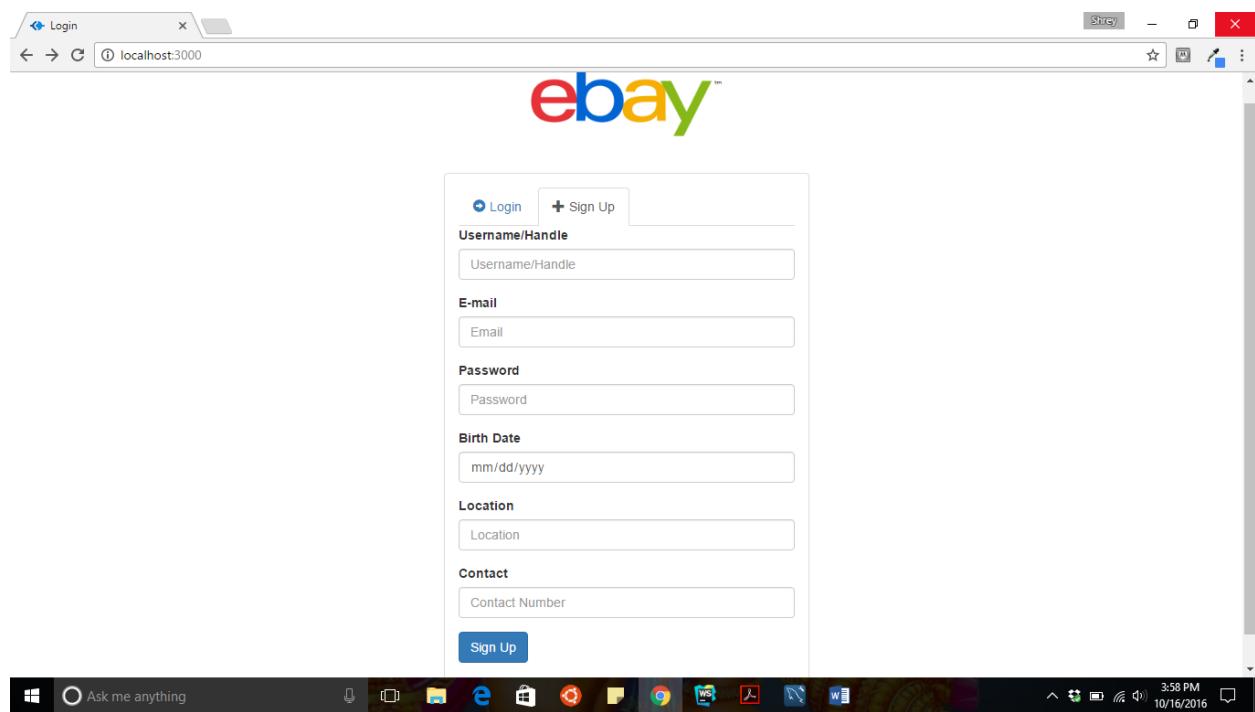
- **Screenshots**
- **Login Page**
- First page to be displayed after running the application. User Can not use the application without Logging in.



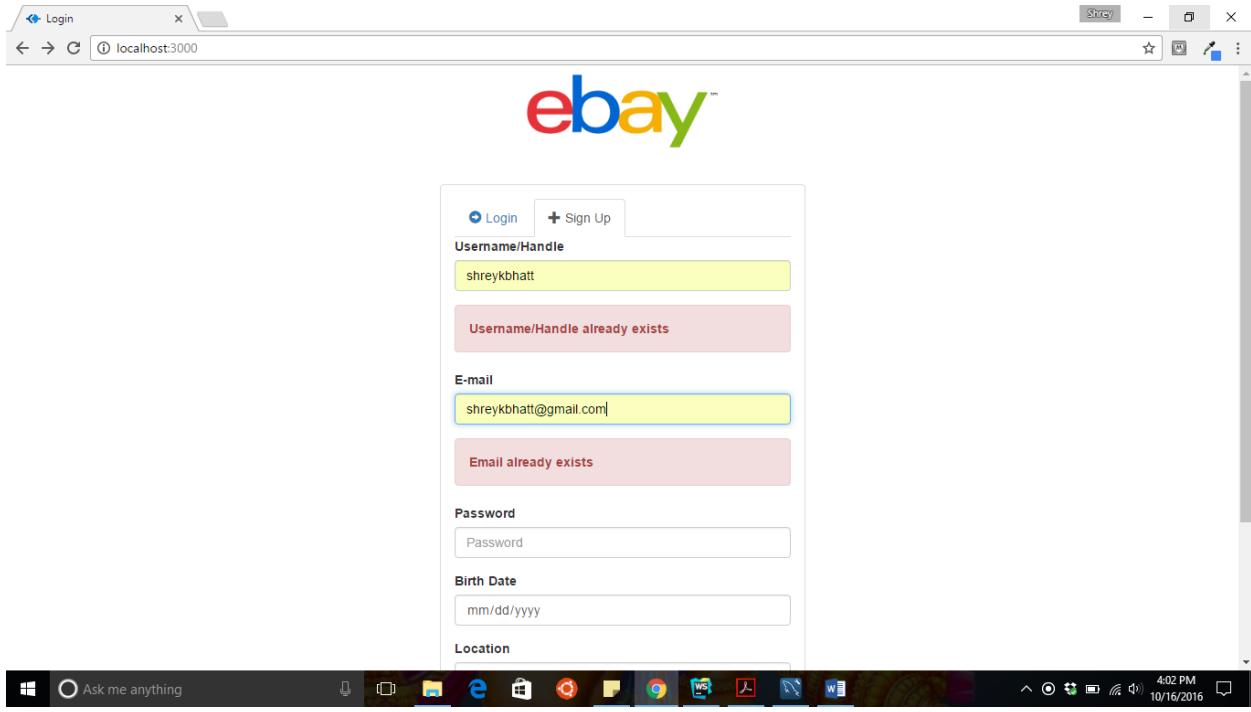
- **Invalid Username/password**
- If the username/password is not correct it will display “invalid username or password” message.



- **SignUp/Register Page**
- Enter Username/Ebay Handle,E-mail,Password,BirthDate,Location and Contact Details.



- If the User Already Exists then It will not allow to add.



- **After Login Screen**
- It will display all the advertisement which are not posted by the user who is currently logged in.

All Advertisements

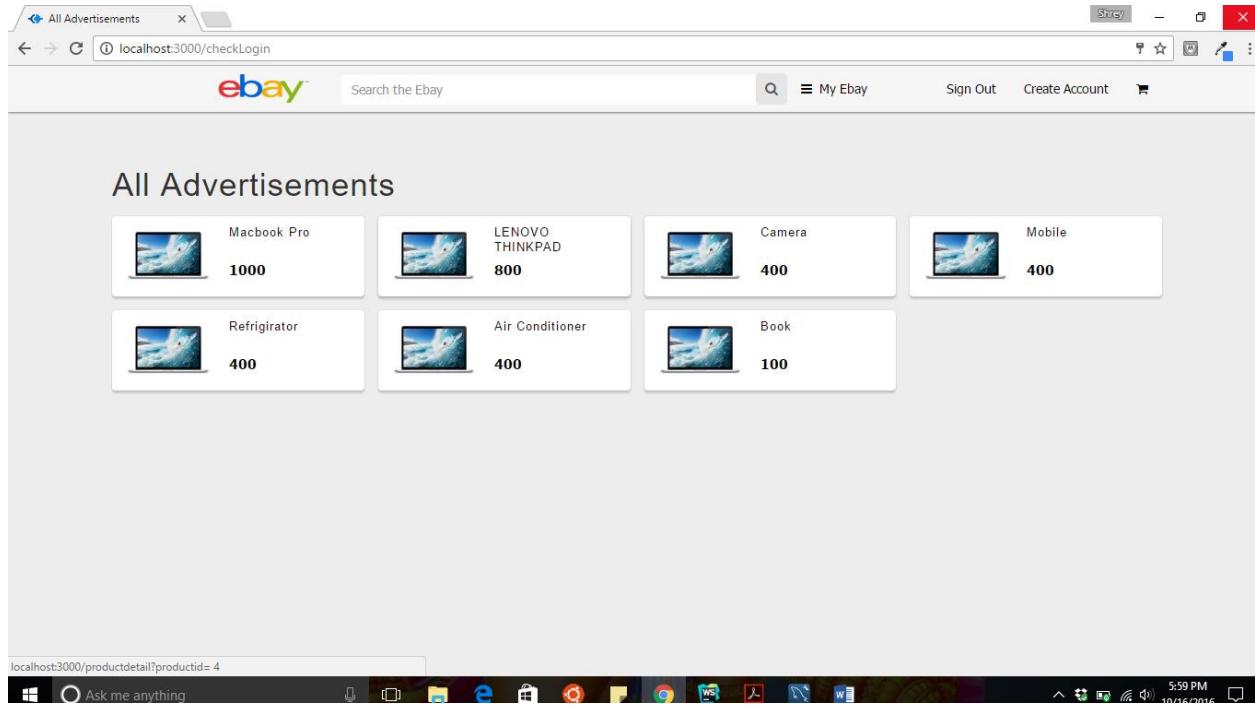
 Dell Laptop 700	 WebCam 500	 Macbook Pro 1000	 Samsung GALAXY S7 800
--	---	---	--

localhost:3000/productdetail?productid= 7

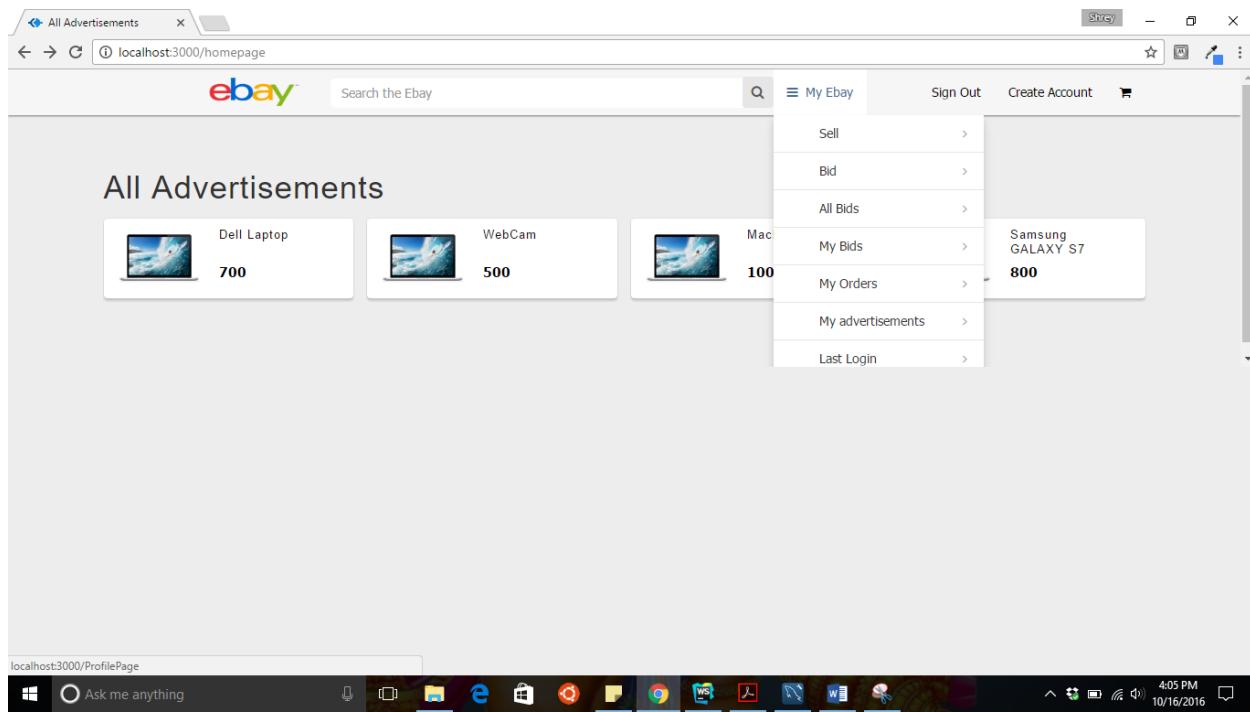
```
Object {data: Array[4], status: 200, config: Object, statusText: "OK"}  
advertisment.js:24
```



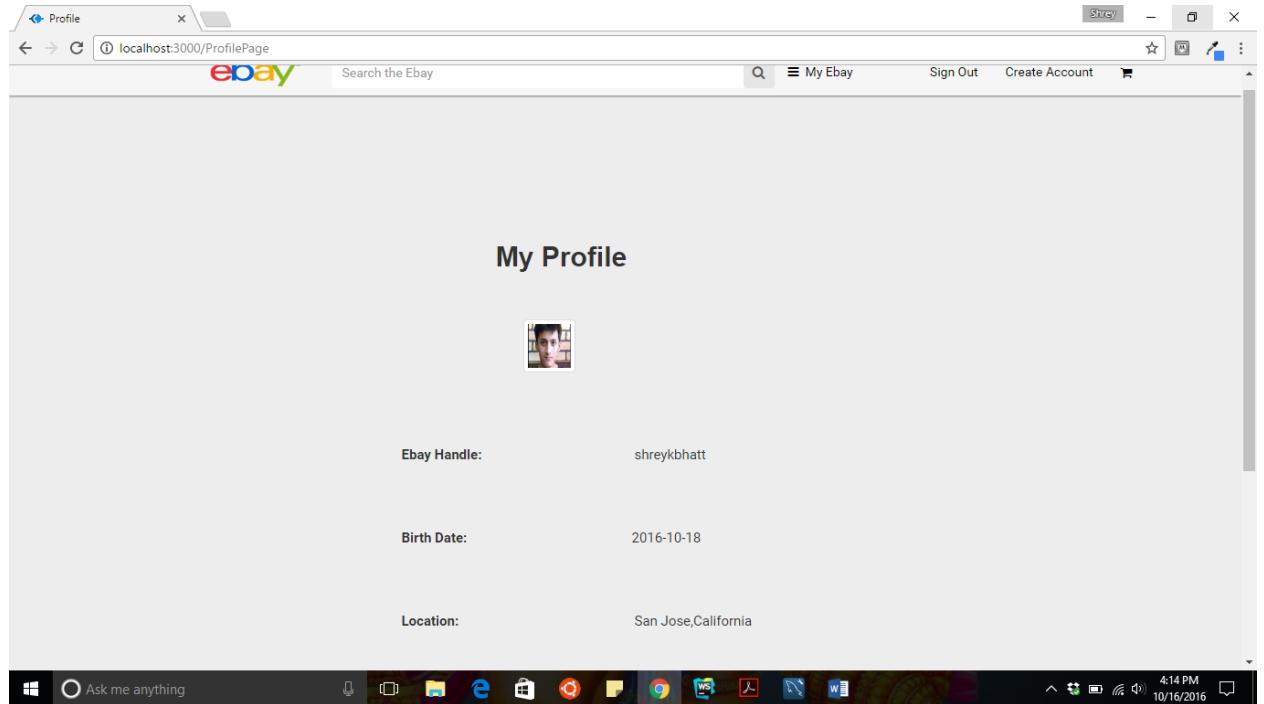
- **HomePage**
- On clicking the Ebay icon the Homepage will be opened.
- It will display All the advertisements which are not posted by the user who is currently logged in.



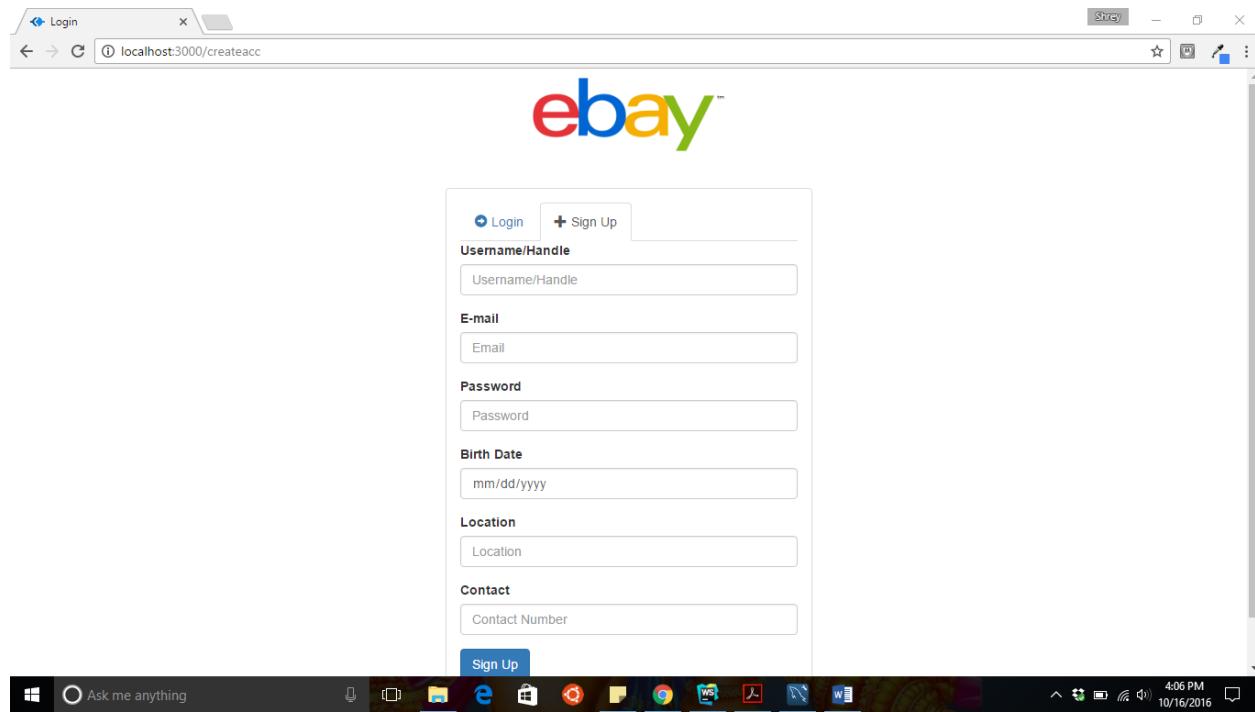
- **Menu Pops Up on clicking the MyEbay**



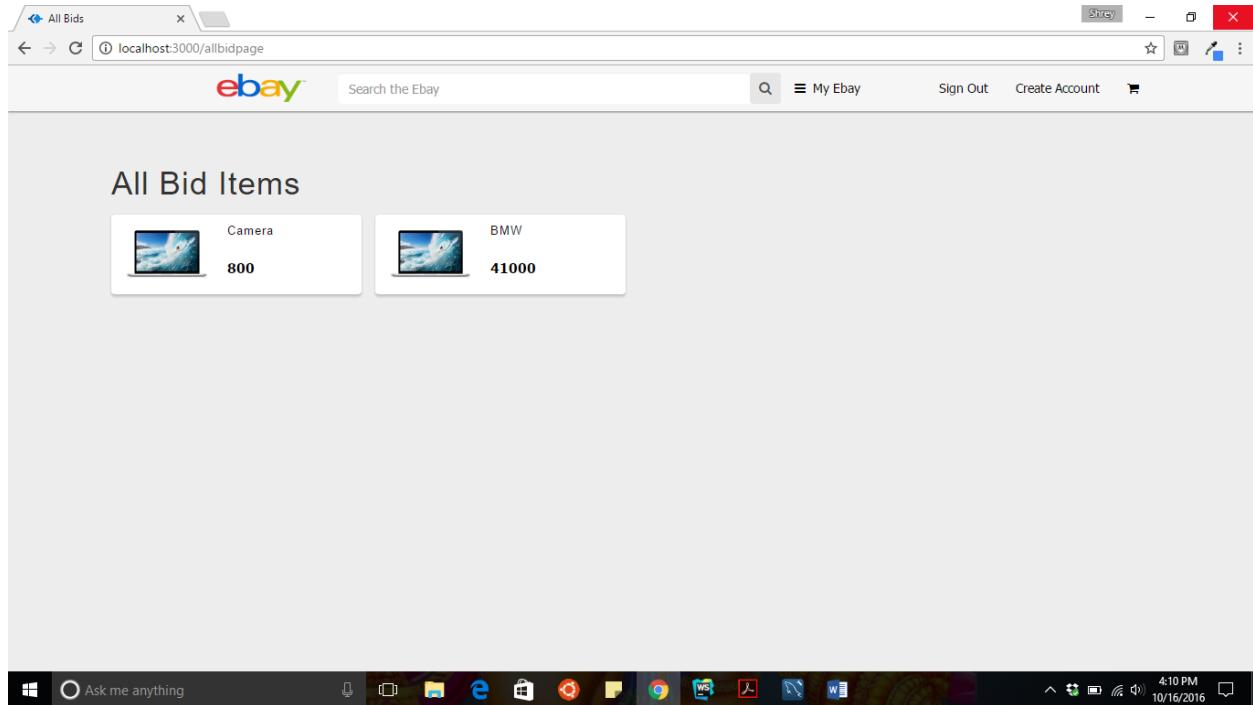
- **ProfilePage**



- **Create Account**
- On clicking the create account page in the title bar the registration/SignUp page appears.



- **All Bids**
- This page displays all the products available for bidding which are not posted by the user who is currently logged in.



- **My Advertisements**
- This page displays all the advertisements which are posted by the user who is currently logged in.

My Advertisements

Macbook Pro 1000	LENOVO THINKPAD 800	Camera 400	Mobile 400
Refrigerator 400	Air Conditioner 400	Book 100	

- **My Bids**
- This page displays all the products available for bidding which are posted by the user who is currently logged in.

My Bids

Macbook 1000	Laptop 800	Mobile 600
-----------------	---------------	---------------

- **Post Advertise**
- To post the advertisement, User will fill up the following form.

The screenshot shows a web browser window with the title bar "Post Advertisement Details". The address bar displays "localhost:3000/advertisement". The page content is styled like eBay, featuring the eBay logo and navigation links for "Search the Ebay", "My Ebay", "Sign Out", and "Create Account". The main area is titled "Enter advertisement details:" and contains five input fields: "Product ID", "Product Name", "Description", "Price", and "Quantity". Below these fields is a "Shipping From" label. At the bottom of the screen, the Windows taskbar is visible, showing various pinned icons and the system tray with the date and time "10/16/2016 4:09 PM".

Product ID  
Product Name  
Description  
Price  
Quantity  
Shipping From

- **Post Bid**
- To post the bid, User will fill up the following form.

Post Bid Details

localhost:3000/bid

ebay Search the Ebay My Ebay Sign Out Create Account

Enter Bid details:

Product ID

Product Name

Description

Base Price

Shipping From

**SUBMIT**

Ask me anything 4:10 PM 10/16/2016

- **Product Details**
- It will display product details like Product name, Price and Quantity(can be edited) and product description.

Product Details

localhost:3000/productdetail?productid=%202

ebay Search the Ebay My Ebay Sign Out Create Account



**Dell Laptop**

**\$700**

- 1 + ADD TO CART

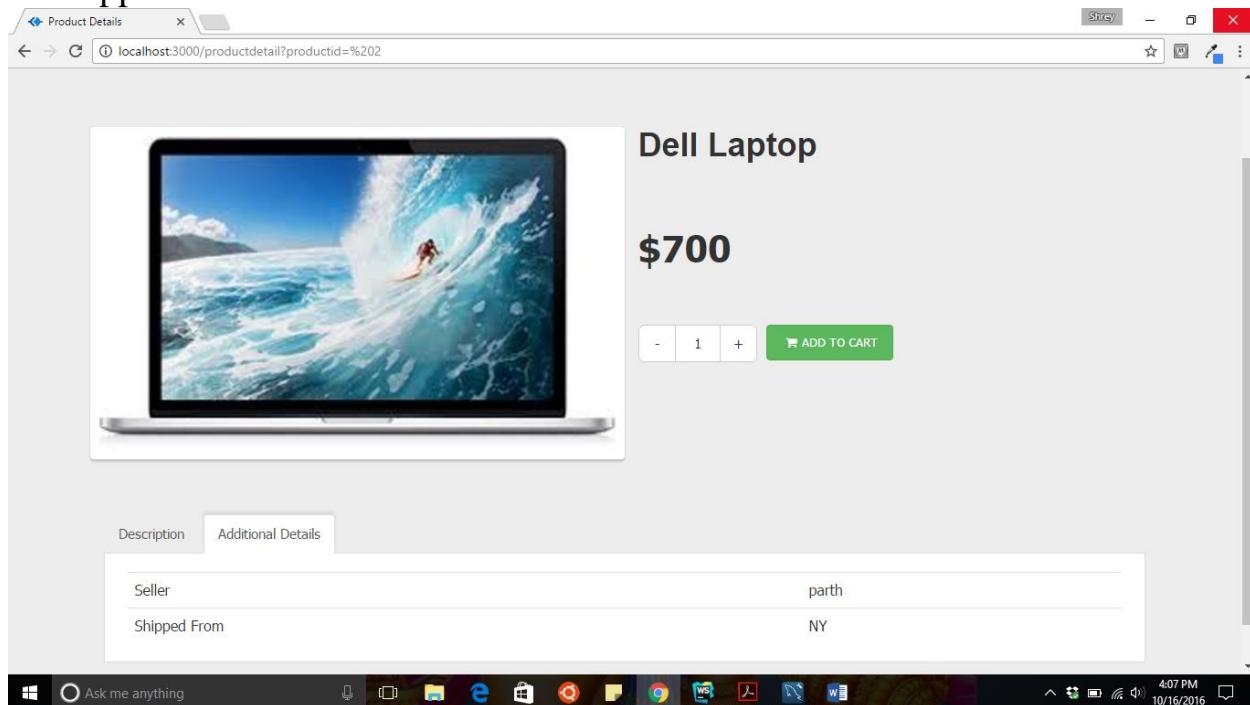
Description Additional Details

1 TB Memory 16 GB RAM 2.5 GHZ

Ask me anything 4:07 PM 10/16/2016

- **Additional Details**

Additional details will contain Seller and the location from where the product is shipped.



- **Bid Product Details**

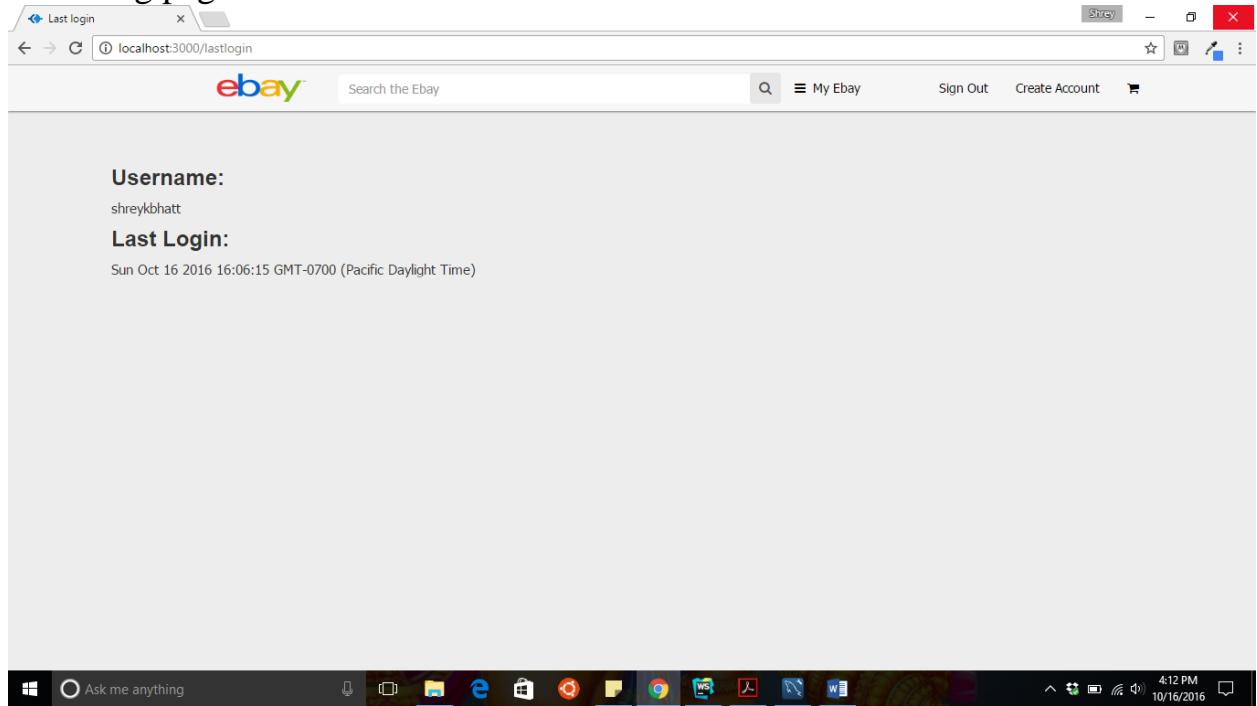
- It will display product details like Product name, Base Price and product description.
- User can Bid on the product by entering the Bid price. After bidding the entry for that product and that particular user will be their in the bid and biddinghistory table. So when User will go to “all bids page” or “bid product detail page” then the price will be updated to latest maximum bid price.

The screenshot shows a web browser window titled "Bid Product Details" with the URL "localhost:3000/bidproductdetail?bidproductId=%203". The page is styled like eBay, featuring the eBay logo and search bar. A large image of a laptop screen displaying a surfer riding a wave is the central focus. To the right, the word "BMW" is displayed in bold capital letters. Below it, the text "Latest Price:\$41000" is shown in a large font. A bidding form with a placeholder "Enter your Bid Price" and a red "BID NOW" button follows. At the bottom, tabs for "Description" and "Additional Details" are visible, along with a note "15 MPH Average". The Windows taskbar at the bottom shows various pinned icons and the date/time "10/16/2016 4:11 PM".

- **Shopping-Cart**
- User can add or remove items from the shopping card.

The screenshot shows a web browser window titled "Shopping Cart" with the URL "localhost:3000/cart". The eBay header and search bar are present. The main content is titled "My Shopping Cart". A table lists three items: a Dell Laptop (700 price, 7 quantity, total 4900), a Macbook Pro (1000 price, 3 quantity, total 3000), and a Webcam (500 price, 2 quantity, total 1000). Each item has a "Remove" link next to its row. To the right of the table, a summary shows "Subtotal 8900", "Shipping Free", and "Total 8900", with a blue "CHECKOUT" button below. At the bottom left is a "CONTINUE SHOPPING" link. The Windows taskbar at the bottom shows the date/time "10/16/2016 4:09 PM".

- **Last Login Page**
- Last Login time is stored in the user database. It will be updated everytime user logs out. So username and lastlogin time will be displayed in the following page.



- **My Orders**
- Orders of the current user will be displayed on the following page.

My Orders

Order Date	Product Name	Quantity	Price	Total
Wed Oct 12 2016	WebCam	2	500	1000
Thu Oct 13 2016	WebCam	2	500	1000
Thu Oct 13 2016	Dell Laptop	7	700	4900
Sat Oct 15 2016	Macbook Pro	3	1000	3000

- Credit Card Validation
- On clicking the checkout, the credit card payment page will be rendered.

Credit Card Payment

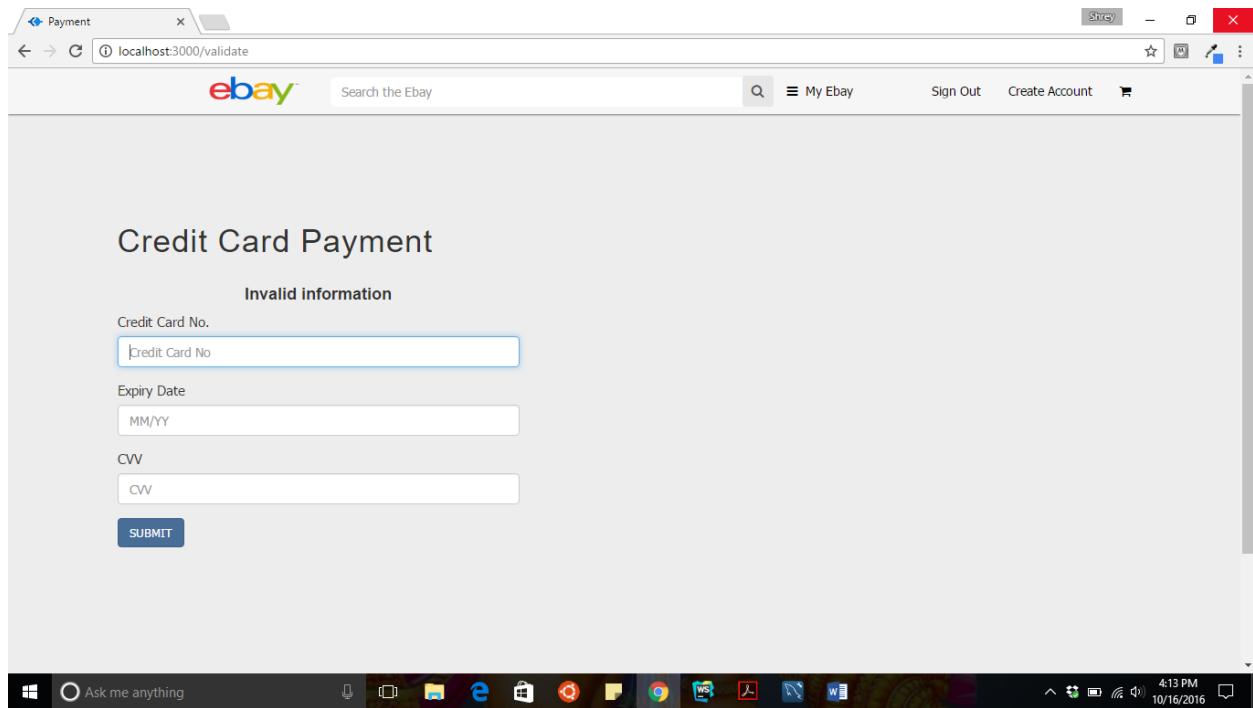
Credit Card No.

Expiry Date

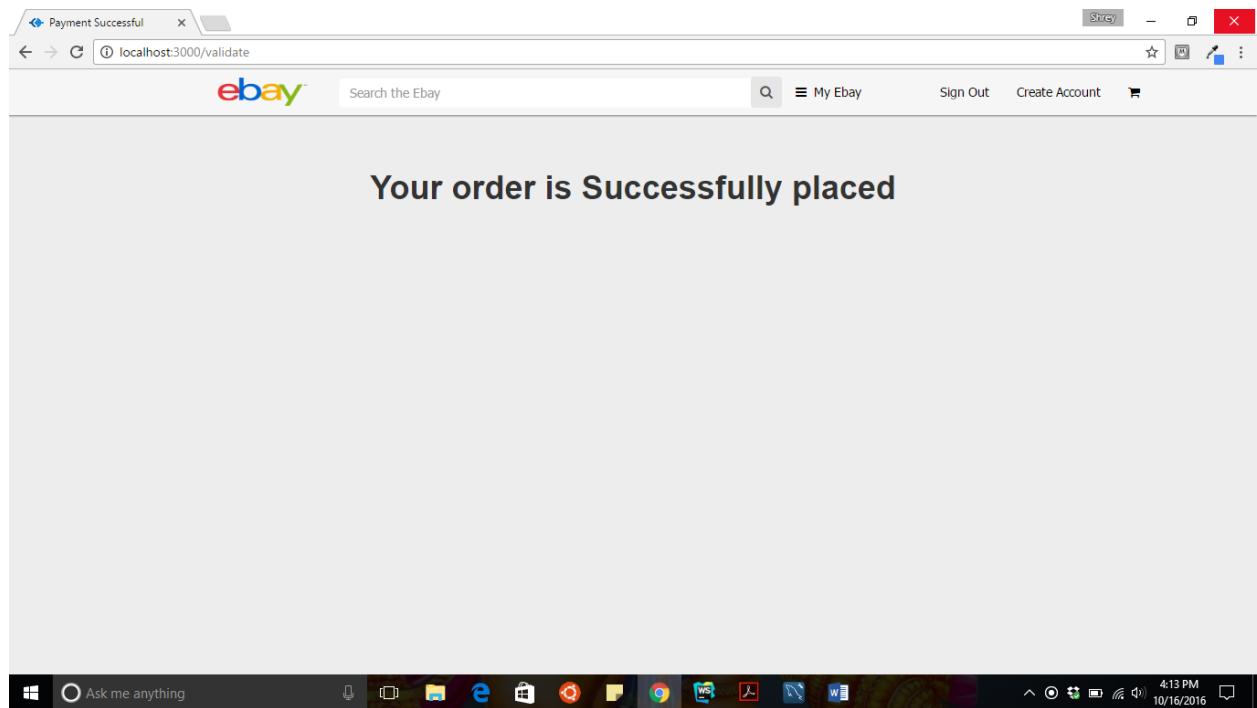
CVV

SUBMIT

- **Invalid Card**
- If the card info is invalid then it will show the message of “Invalid Information”.



- **Payment Successful**
- If the payment will be successful the the “Payment Successful” page will be rendered. The cart will be empty, the order table will have entry for all the products for current user.



- Database: Ebay

- User Table

The screenshot shows the MySQL Workbench interface with the following details:

- Navigator:** Shows the database structure under the 'ebay' schema, including tables like bid, biddinghistory, cart, order, product, user, and views.
- SQL Editor:** Displays the query `describe user;` and its results in a Result Grid. The table structure is as follows:

Field	Type	Null	Key	Default	Extra
useid	int(11)	NO	PRI	NULL	auto_increment
username	varchar(45)	NO		NULL	
email	varchar(45)	NO		NULL	
password	varchar(100)	NO		NULL	
birthdate	varchar(45)	NO		NULL	
location	varchar(45)	NO		NULL	
contact	varchar(45)	NO		NULL	
lastlogin	datetime	YES		NULL	

- Table: bid**: Shows columns including bidproductid, bidproductname, biddescription, bidprice, bidstarttime, bidderid, and biddername.
- Output:** Displays the results of the 'describe' queries with the following details:

#	Time	Action	Message	Duration / Fetch
50	17:13:15	describe bid	10 row(s) returned	0.062 sec / 0.000 sec
51	17:14:02	describe user	8 row(s) returned	0.000 sec / 0.000 sec

## • Product Table

The screenshot shows the MySQL Workbench interface with the database set to 'Local instance MySQL57'. In the Navigator pane, under the 'ebay' schema, the 'Tables' section is expanded, showing the 'bid' table. The 'order' tab is selected in the main query editor. The SQL query entered is:

```
1 describe product;
```

The Result Grid displays the structure of the 'product' table:

Field	Type	Null	Key	Default	Extra
productid	int(11)	NO	PRI	NULL	auto_increment
productname	varchar(45)	NO		NULL	
description	varchar(45)	NO		NULL	
price	int(11)	NO		NULL	
quantity	int(11)	NO		NULL	
shippingfrom	varchar(45)	NO		NULL	
sellerid	varchar(45)	NO		NULL	

Below the table structure, the 'Object Info' pane shows the detailed column definitions for 'product'.

## • Cart Table

The screenshot shows the MySQL Workbench interface with the database set to 'Local instance MySQL57'. In the Navigator pane, under the 'ebay' schema, the 'Tables' section is expanded, showing the 'cart' table. The 'order' tab is selected in the main query editor. The SQL query entered is:

```
1 describe cart;
```

The Result Grid displays the structure of the 'cart' table:

Field	Type	Null	Key	Default	Extra
cartid	int(11)	NO	PRI	NULL	auto_increment
productname	varchar(45)	NO		NULL	
price	varchar(45)	NO		NULL	
quantity	varchar(45)	NO		NULL	
userid	varchar(45)	NO		NULL	
productid	varchar(45)	NO		NULL	
total	int(11)	NO		NULL	

Below the table structure, the 'Object Info' pane shows the detailed column definitions for 'cart'.

## • Bid Table

The screenshot shows the MySQL Workbench interface with the 'bid' table selected in the Navigator. The 'Tables' section under the 'ebay' schema lists the 'bid' table. The 'Result Grid' tab displays the table structure with columns: bidproductid, bidproductname, biddescription, bidprice, bidshippingfrom, bidderid, bidstartime, bidendtime, bidderid, and isBidEnded. The 'Field' column lists data types such as int(11), varchar(45), and boolean. The 'Type' column includes PRI, NO, and YES. The 'Default' column contains NULL or auto\_increment. The 'Extra' column includes various MySQL-specific options like B, UN, ZF, AI, and G.

## • Order Table

The screenshot shows the MySQL Workbench interface with the 'order' table selected in the Navigator. The 'Tables' section under the 'ebay' schema lists the 'order' table. The 'Table - Table' tab is active, showing the table name 'order', schema 'ebay', collation 'utf8 - default collation', and engine 'InnoDB'. The 'Comments' field is empty. The 'Columns' tab displays the table structure with columns: orderid, productname, quantity, price, total, and date. The 'Data Type' column shows various MySQL data types like INT(11) and VARCHAR(45). The 'P' column indicates primary key status. The 'NN' column indicates not null. The 'B' column indicates binary. The 'UN' column indicates unique. The 'ZF' column indicates zero fill. The 'AI' column indicates auto increment. The 'G' column indicates generated. The 'Default/Expression' column is empty. The 'Column Name' column lists the column names. The 'Collation' dropdown is set to 'utf8 - default collation'. The 'Comments' field is empty. The 'Data Type' field is empty. The 'Default' field is empty. The 'Storage' section includes options for Virtual, Stored, Primary Key, Not Null, Unique, Binary, Unsigned, Zero Fill, Auto Increment, and Generated. The 'Apply' and 'Revert' buttons are at the bottom right of the column editor.

- **BiddingHistory Table**

The screenshot shows the MySQL Workbench interface with the following details:

- MySQL Workbench Window:** Local instance MySQL57.
- Navigator:** Shows the schema structure under the 'ebay' database, including tables like bid, biddinghistory, cart, order, product, user, and views like sakila, sys, and test.
- Central Panel:** A query editor window titled 'order' with the command `describe biddinghistory;`. The results are displayed in a table format:

Field	Type	Null	Key	Default	Extra
<b>id</b>	int(11)	NO	PRI	NULL	auto_increment
userid	varchar(45)	NO		NULL	
productid	varchar(45)	NO		NULL	
productname	varchar(45)	NO		NULL	
time	bigint(20)	NO		NULL	
price	varchar(45)	NO		NULL	
isBidEnded	varchar(45)	NO		NULL	

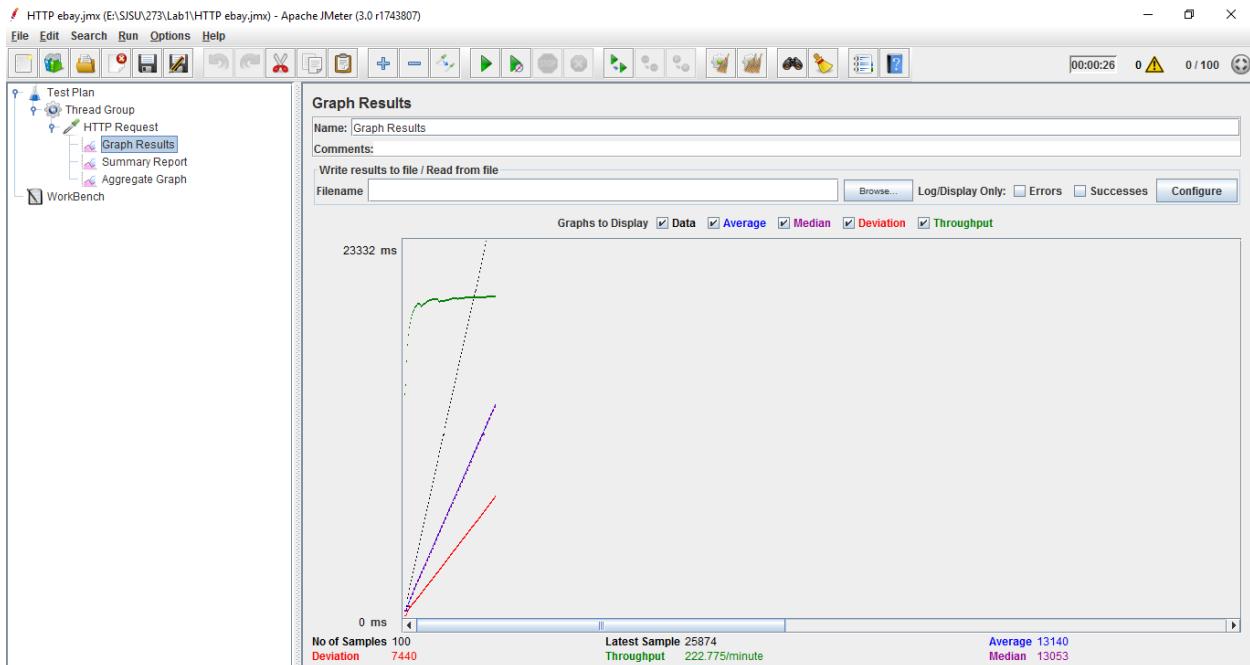
- Result Grid:** A sidebar panel containing various tools and snippets.
- Taskbar:** Shows the Windows taskbar with the date and time (5:16 PM, 10/16/2016).

# Testing

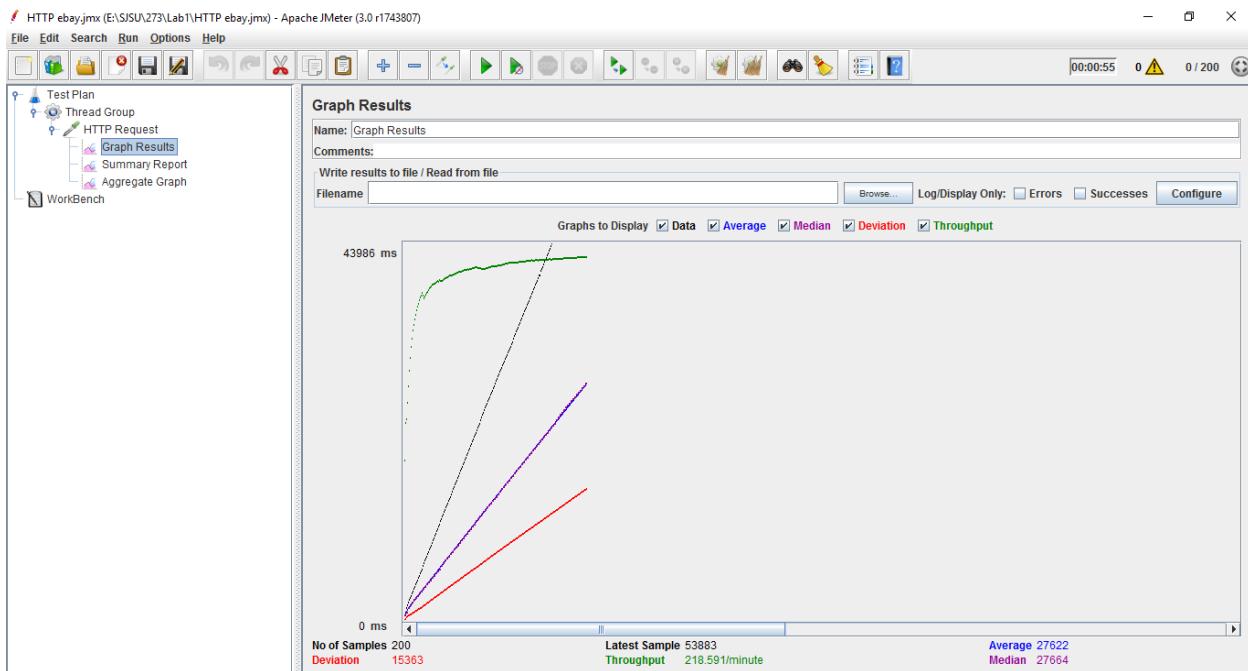
- **JMeter Screenshots**

## 1. Without Connection Pooling

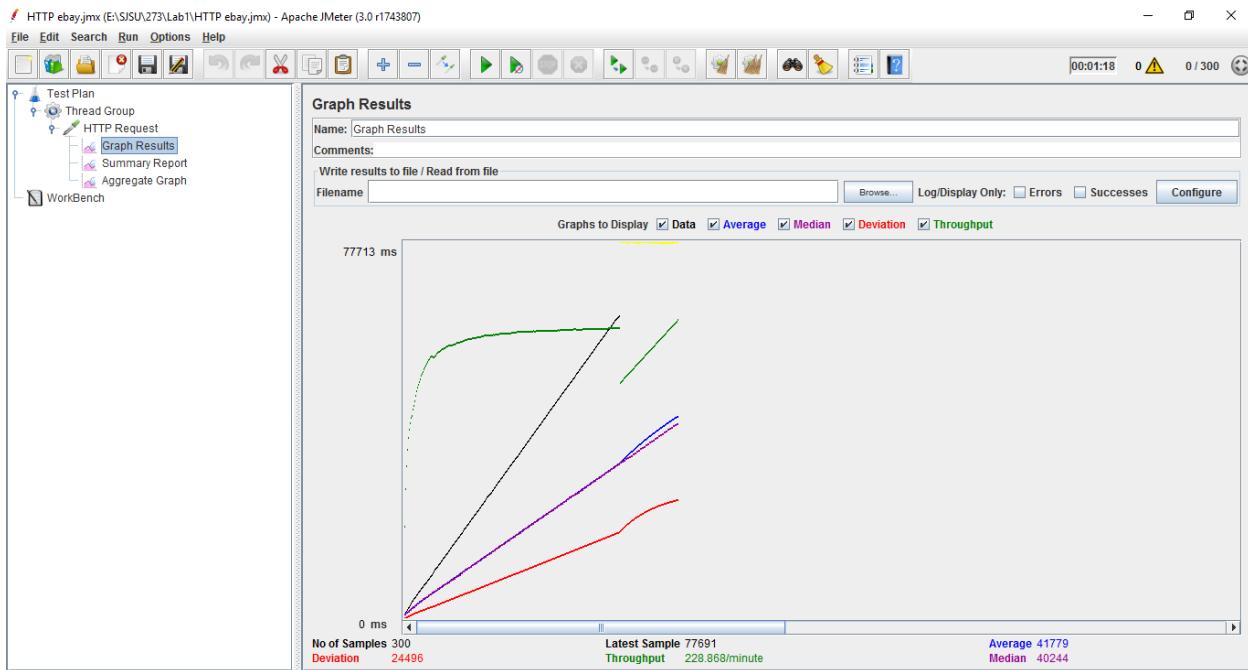
### 1) 100 concurrent users



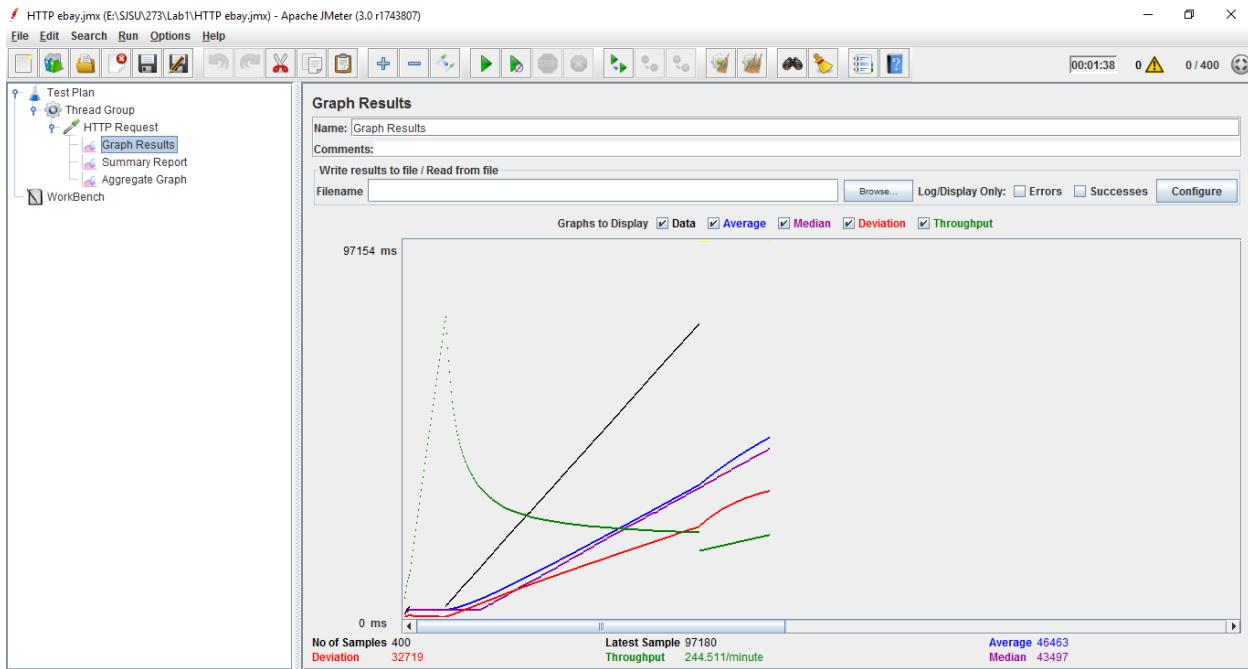
## 2)200 concurrent users



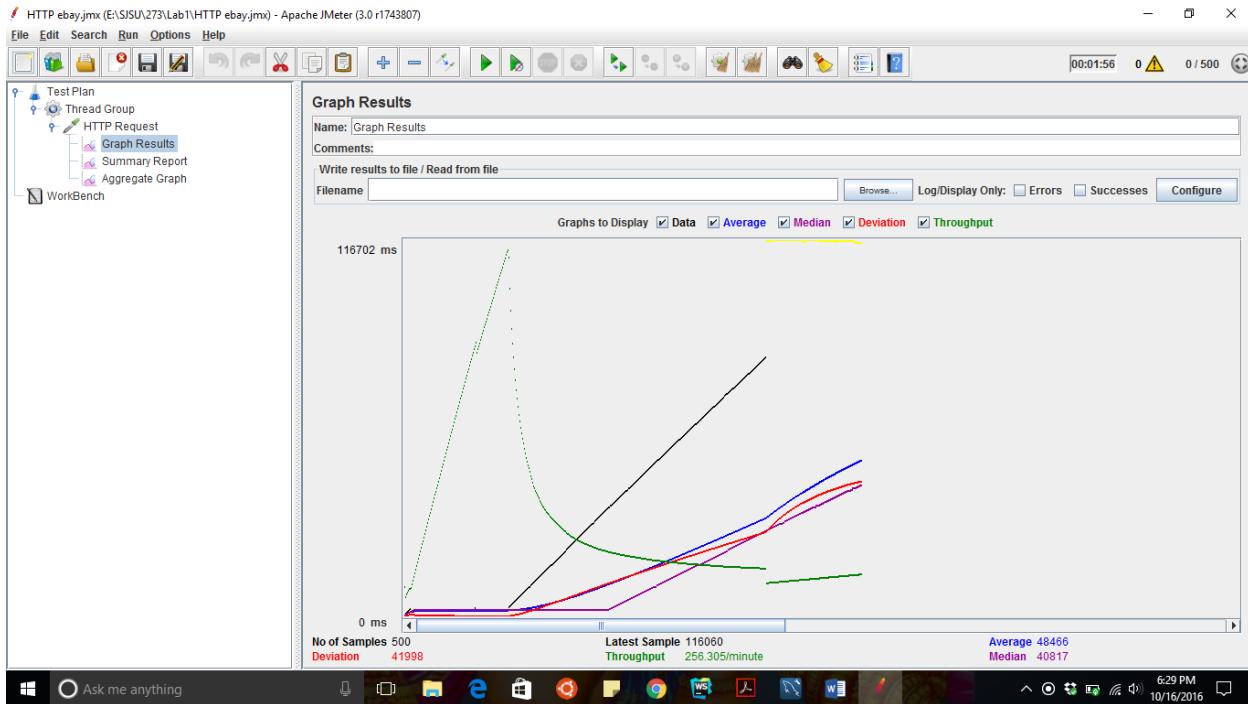
## 3)300 concurrent users



#### 4)400 concurrent users

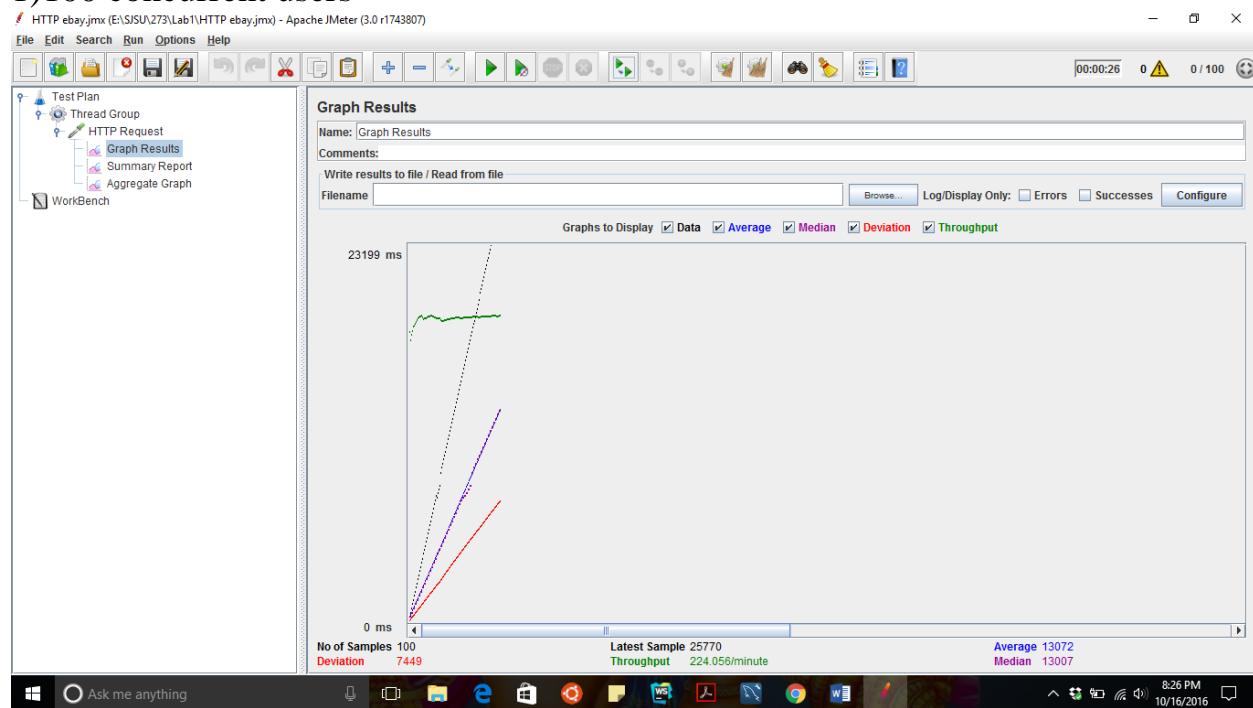


#### 5)500 concurrent Users

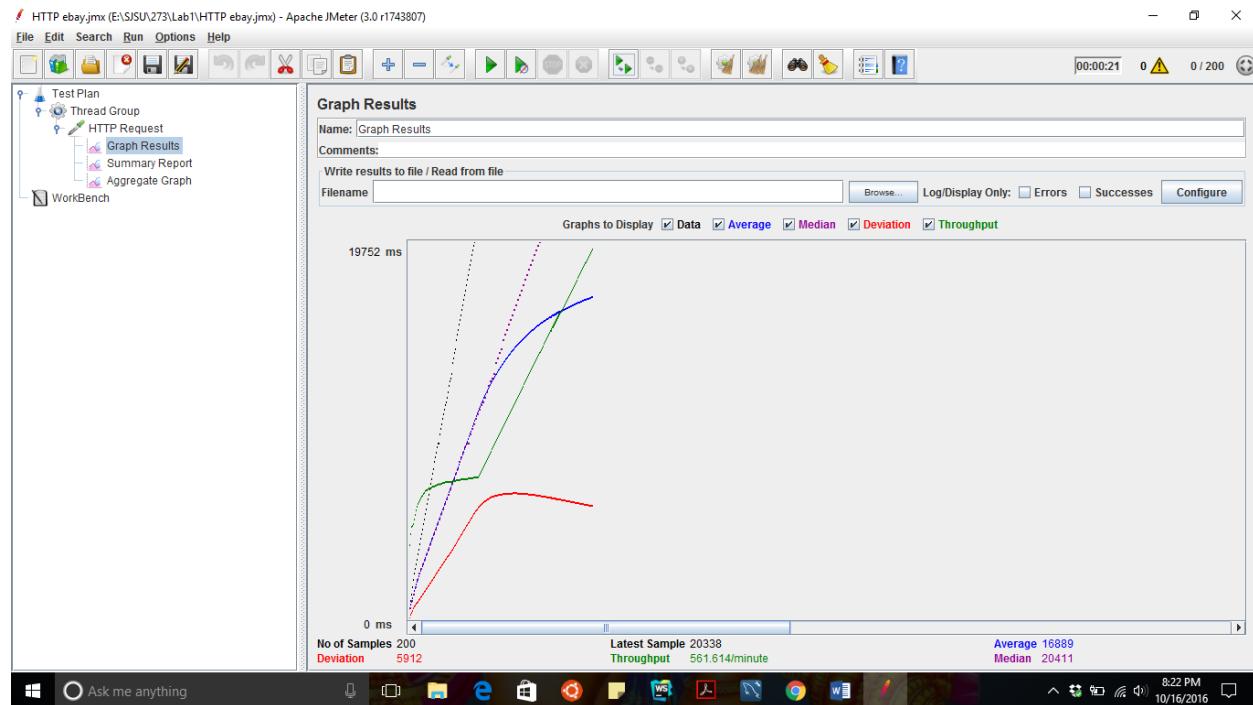


#### 2. With Connection Pooling

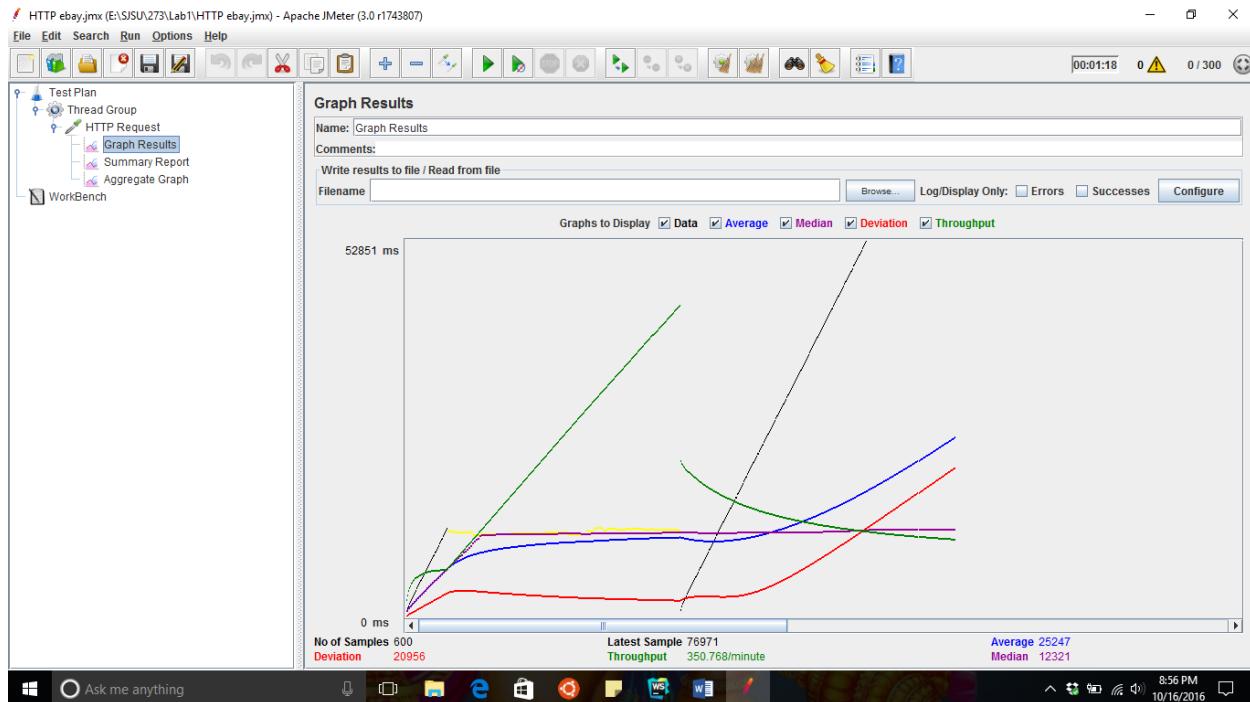
## 1)100 concurrent users



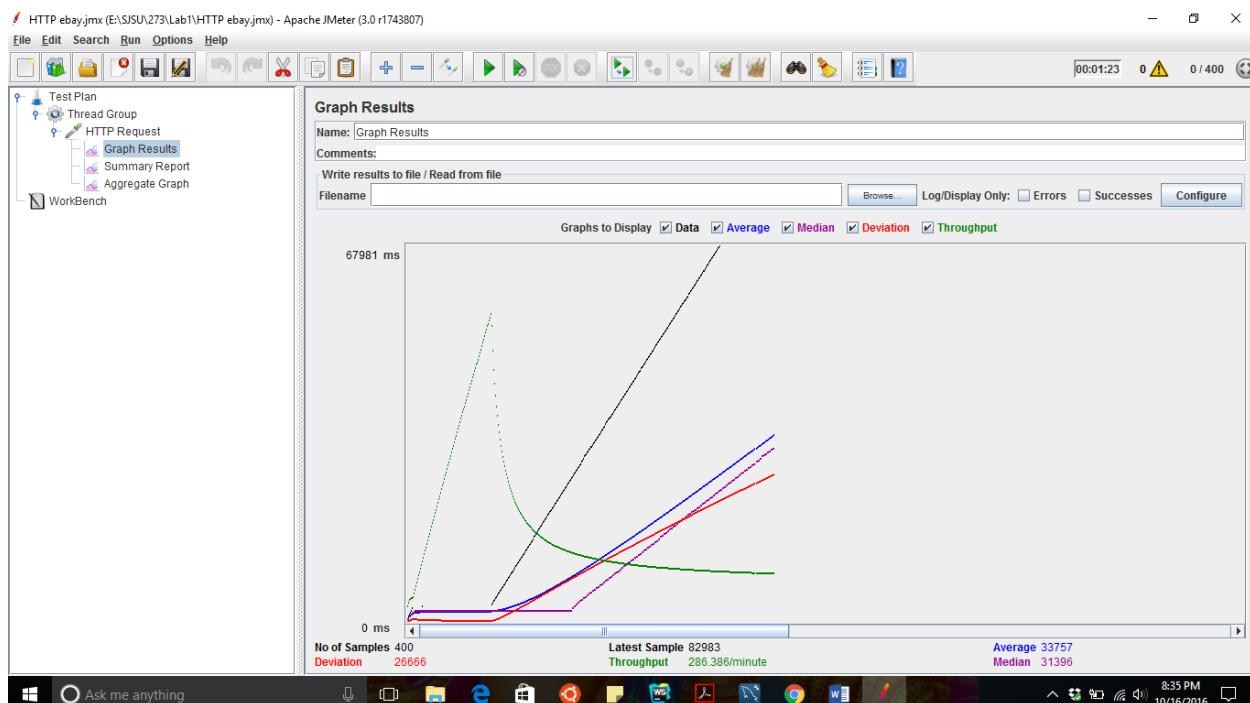
## 2)200 concurrent users



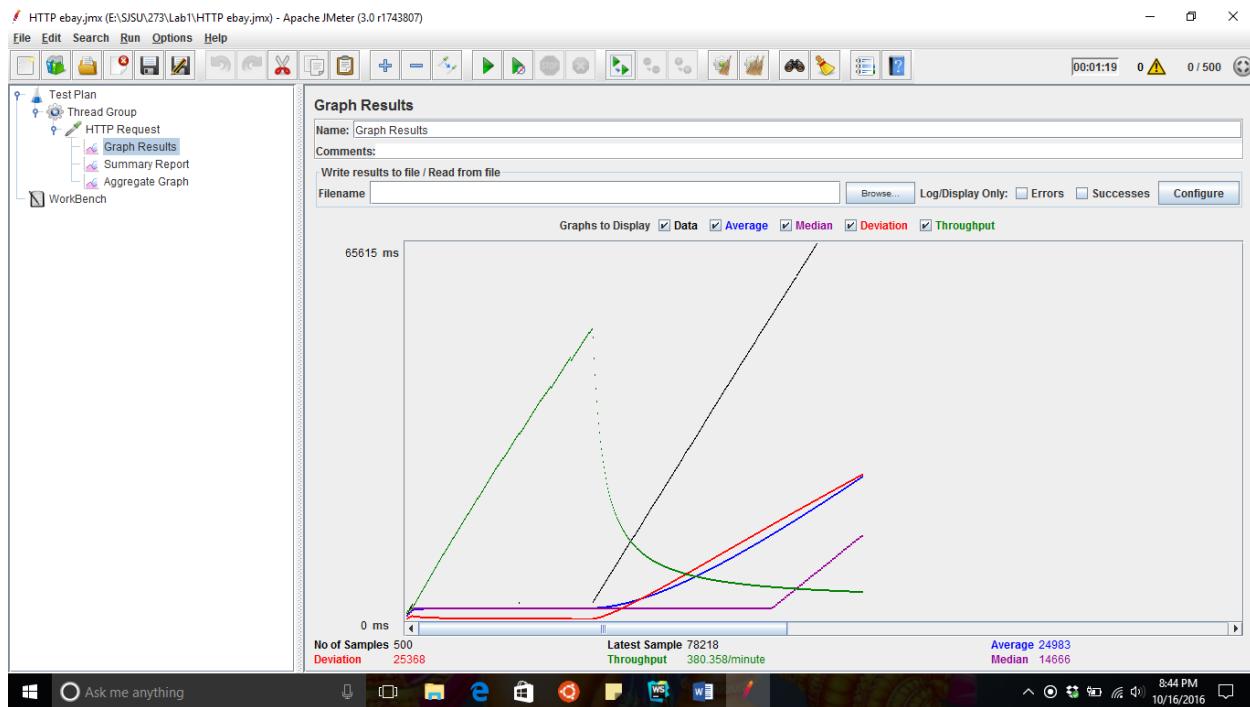
## 3)300 concurrent users



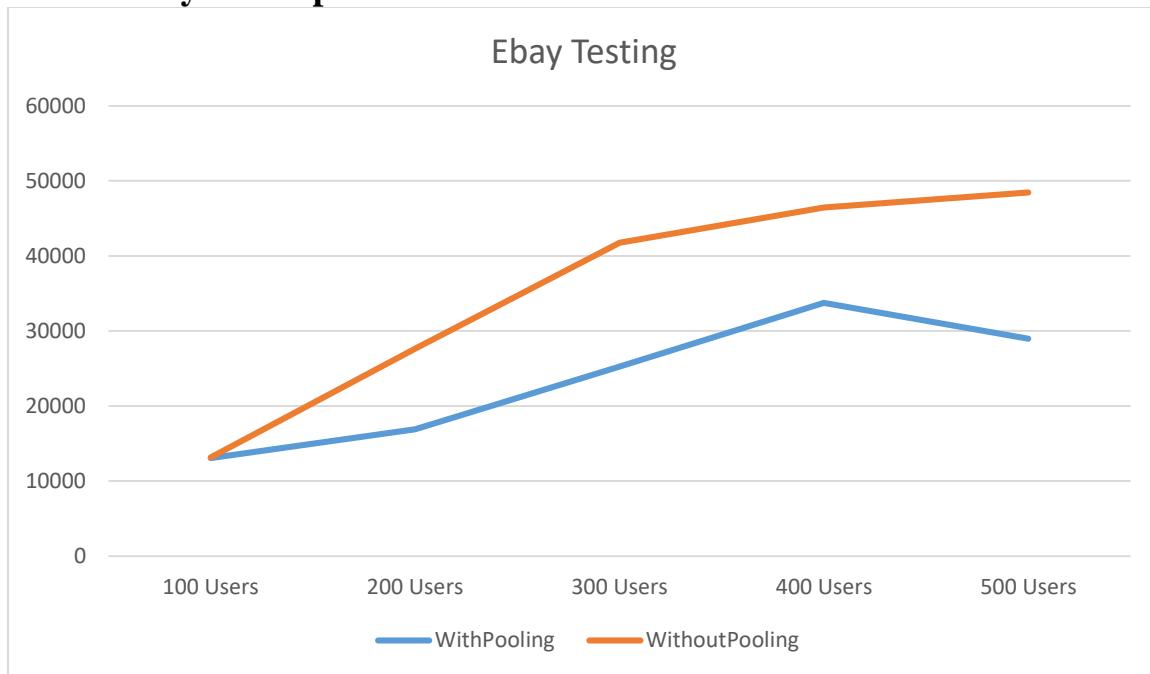
#### 4) 400 concurrent users



## 5) 500 concurrent users



### • Analysis Graph



- **Mocha Testing:- test.js File**

```
var express = require('express');
var request = require('request');
var assert = require('assert');
var http = require('http');
var mocha = require('mocha');

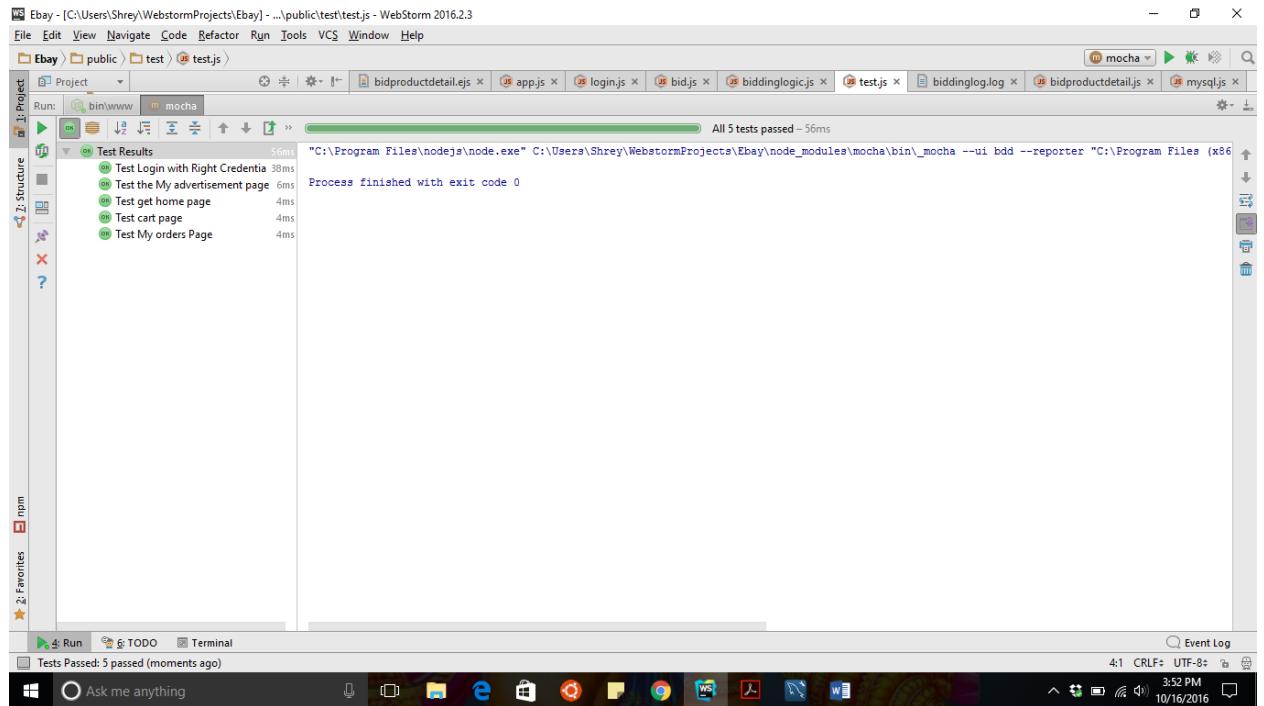
it('Test Login with Right Credentials', function(done) {
  request.post(
    'http://localhost:3000/checkLogin',
    { form: { username: 'shreykbhatt', password: 'test' } },
    function (error, response, body) {
      assert.equal(200, response.statusCode);
      done();
    }
  );
});

it('Test the My advertisement page', function (done) {
  http.get('http://localhost:3000/myadvertisement', function (res) {
    assert.equal(200, res.statusCode);
    done();
  });
});

it('Test get home page', function (done) {
  http.get('http://localhost:3000/homepage', function (res) {
    assert.equal(200, res.statusCode);
    done();
  });
});

it('Test cart page', function (done) {
  http.get('http://localhost:3000/cart', function (res) {
    assert.equal(200, res.statusCode);
    done();
  });
});

it('Test My orders Page', function (done) {
  http.get('http://localhost:3000/myorders', function (res) {
    assert.equal(200, res.statusCode);
    done();
  });
});
```



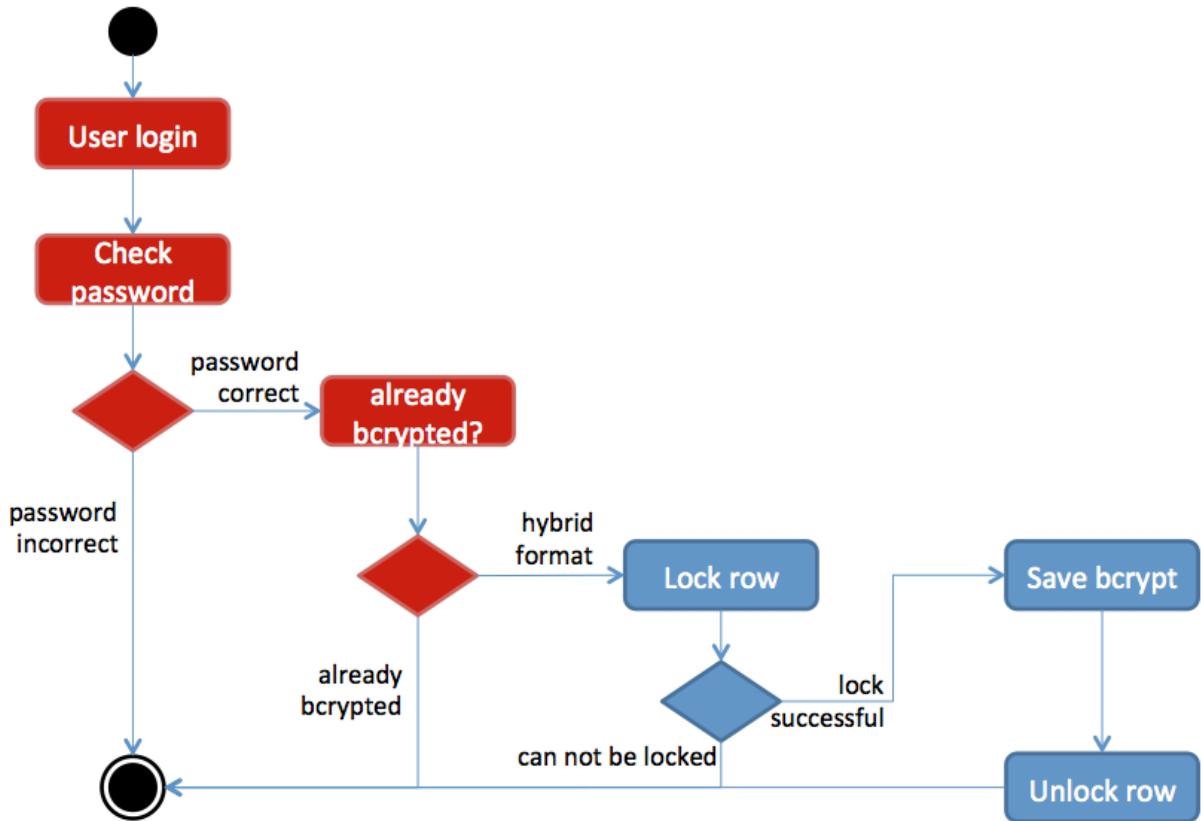
## **Part 3 : Answers**

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

- The algorithm I used for encrypting the password is Bcrypt.
- I used the bcrypt-nodejs module available in node for encrypting and later validating this Password with the encrypted password from the MySQL User database.
- Basically the Bcrypt executes an internal encryption/hash function many times in a loop
- There are many algorithms available like SHA (1, 224, 256, 384, 512), MD(2,4,5,6),Crypto, AES 256 etc.
- The reason I choose Bcrypt is because I think it is secure and easier to implement in node.js application using npm (Node Packet Manager).
- This is how I have used it:

```
var encrypt=bcrypt.hashSync(password1);
:
:
:
if(bcrypt.compareSync(password1,results[0].password)) {
:
:
```

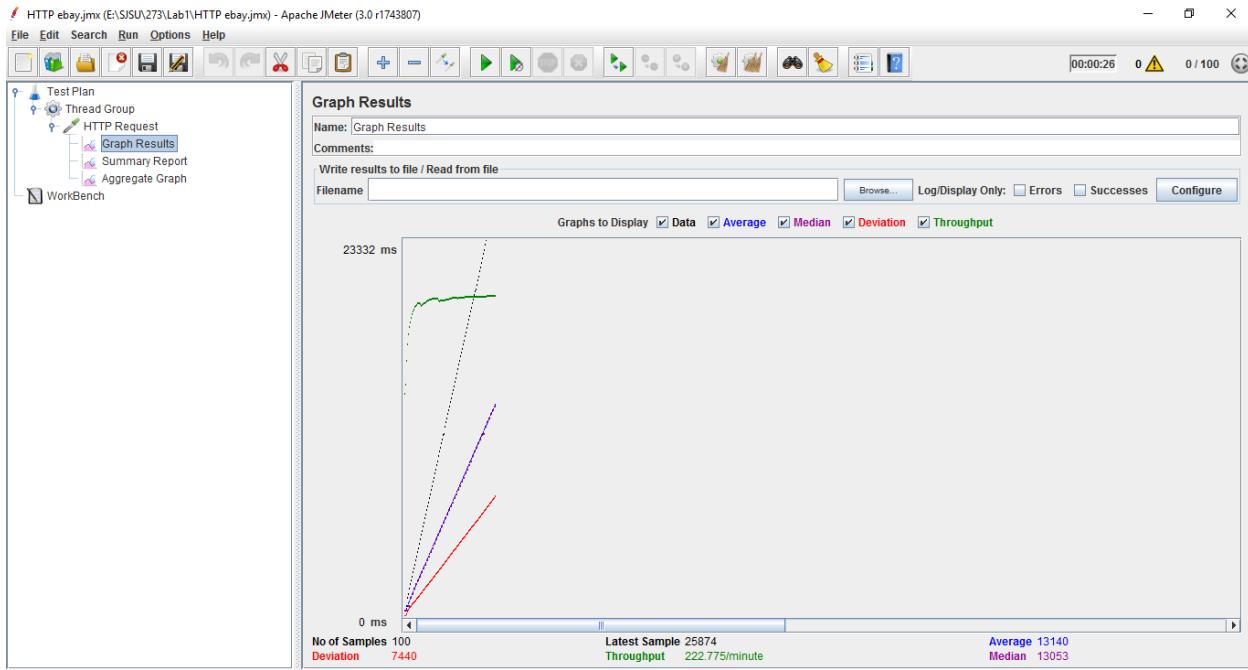
- **Working Of Bcrypt algorithm:-**



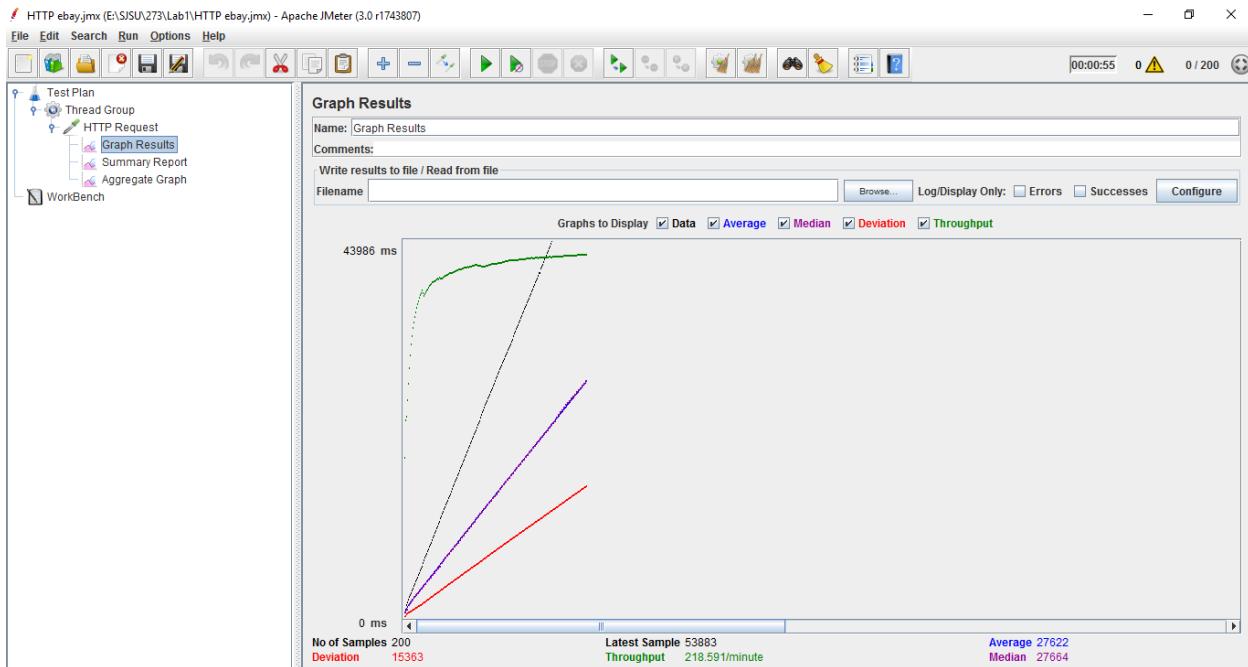
2. Compare the Results of the graphs with and without connection pooling. Explain the results in detail. Describe the algorithm of connection pooling used in your application

### 1. Without Connection Pooling

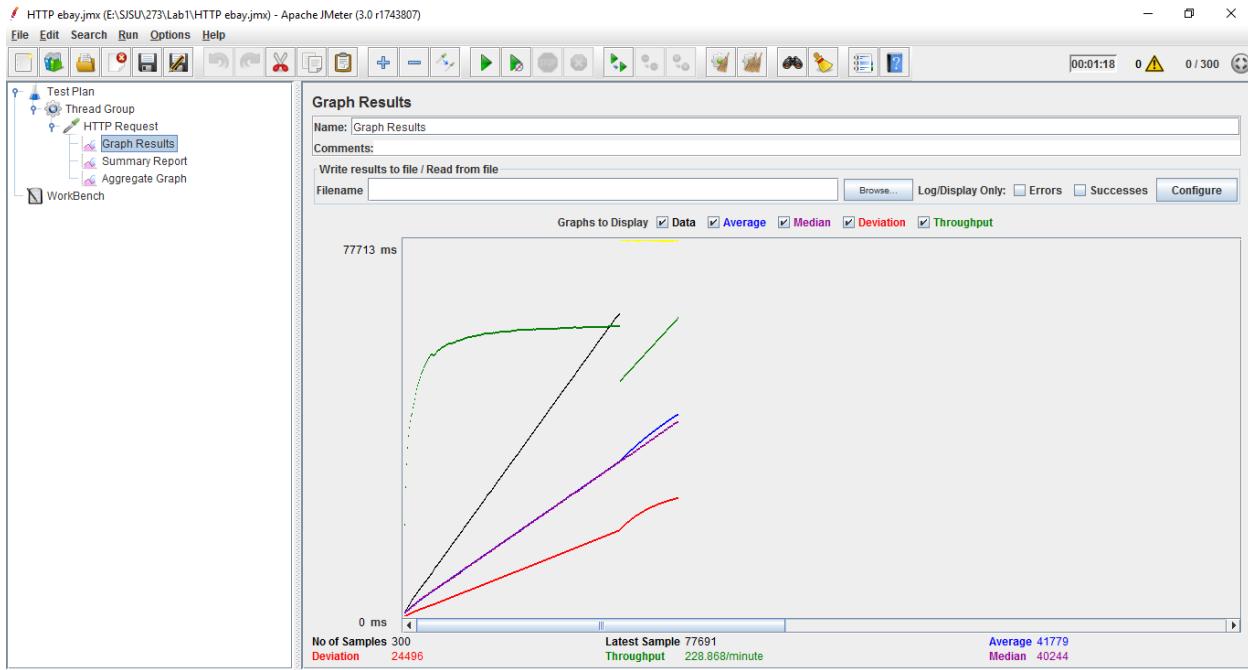
1) 100 concurrent users



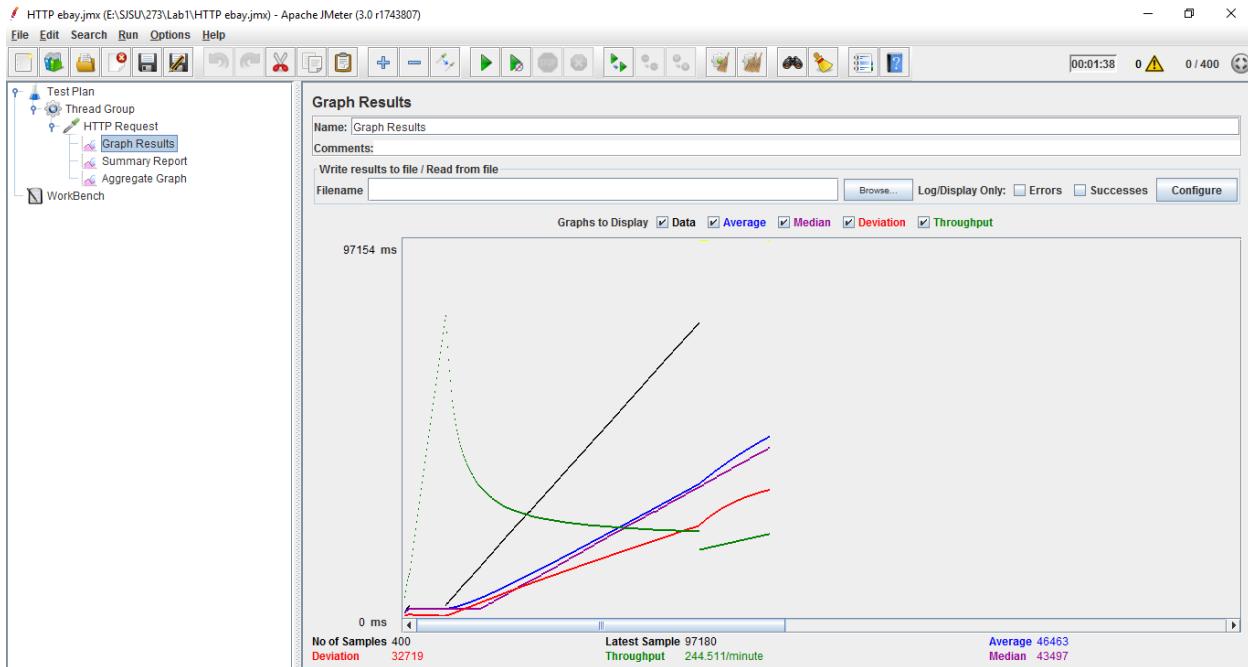
## 2)200 concurrent users



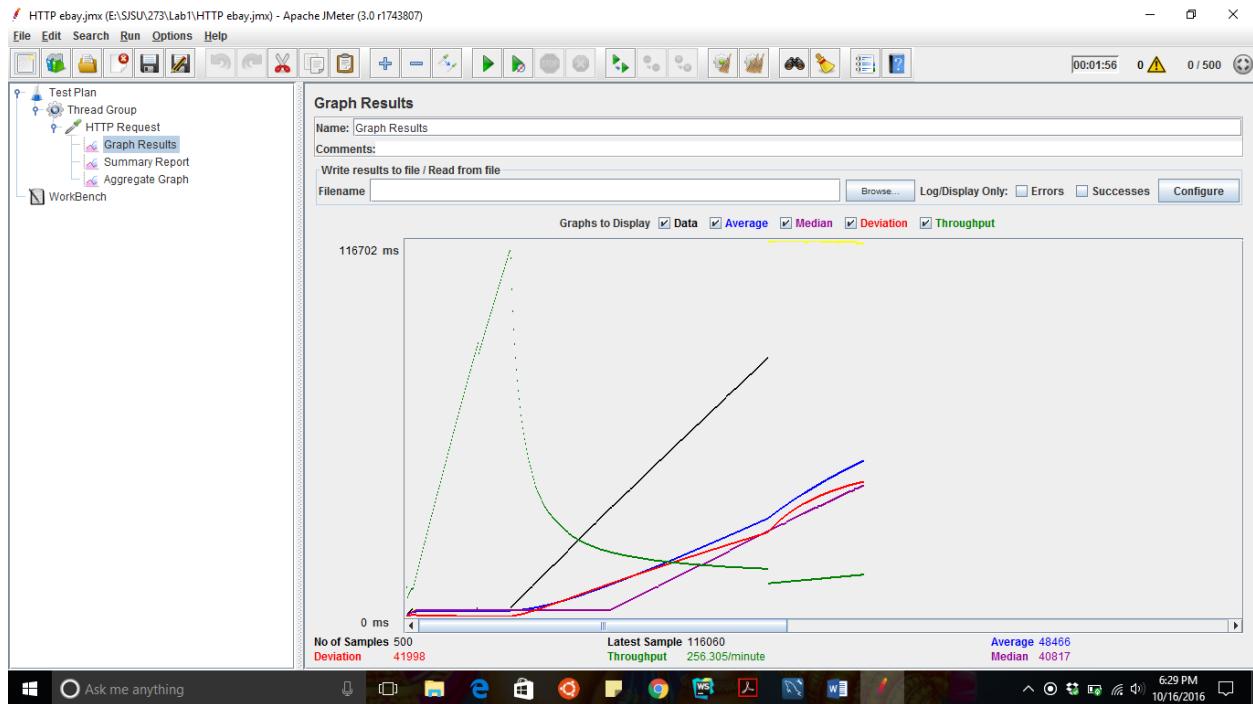
## 3)300 concurrent users



#### 4)400 concurrent users

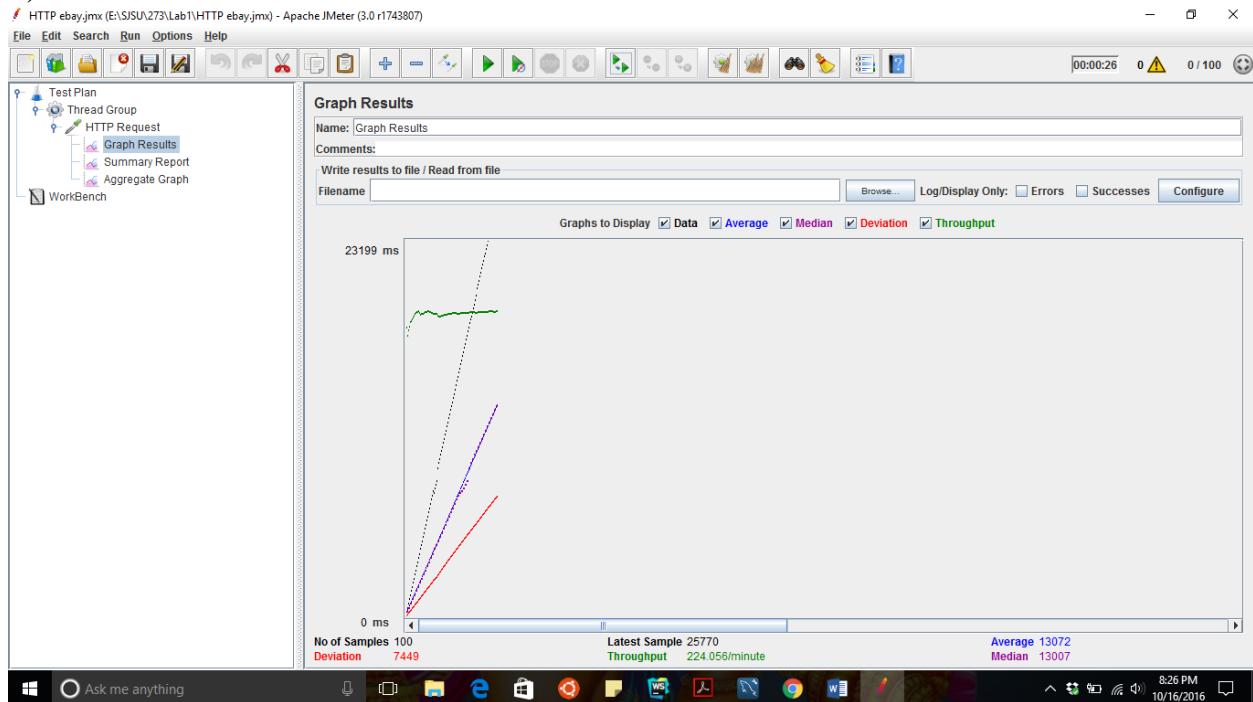


#### 5)500 concurrent users

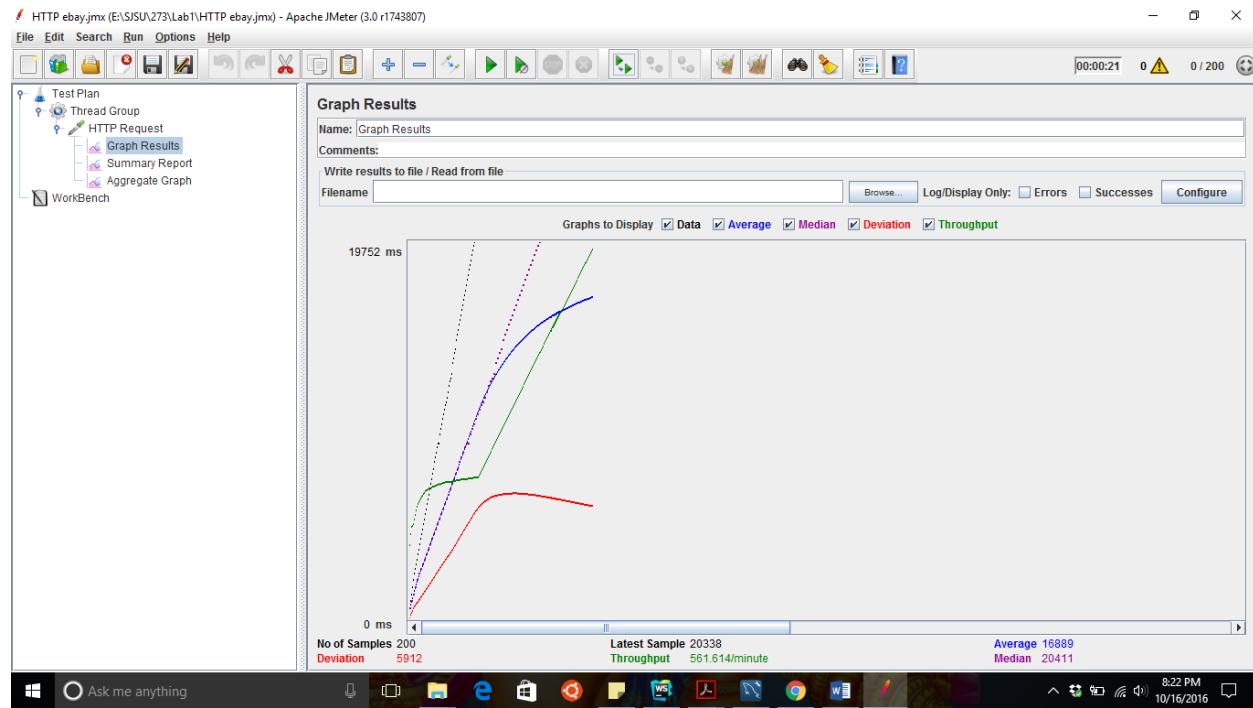


## 2. With Connection Pooling

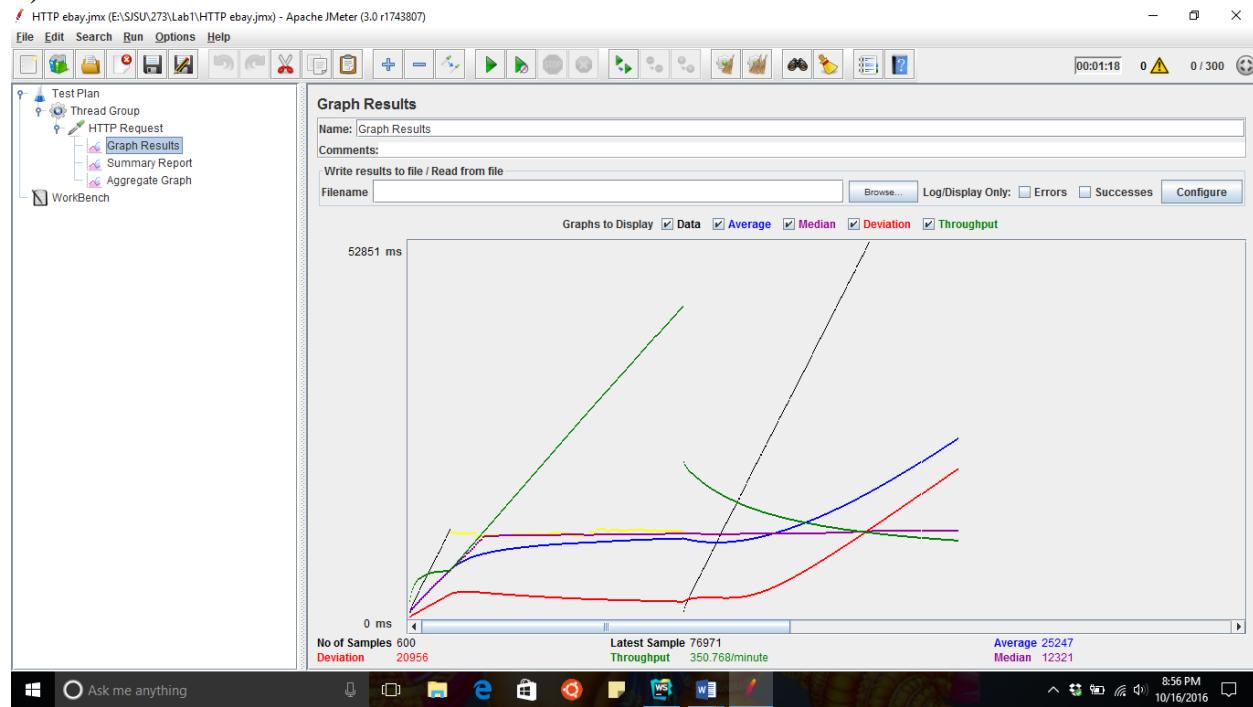
### 1) 100 concurrent users



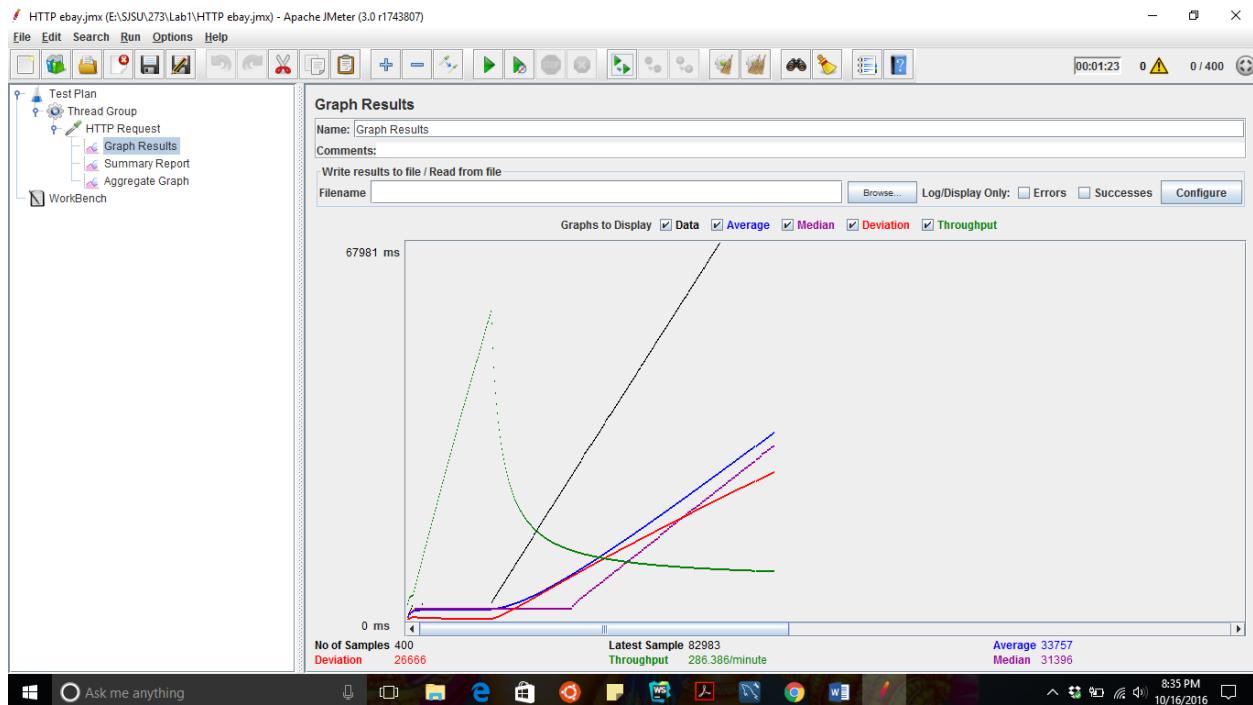
## 2)200 concurrent users



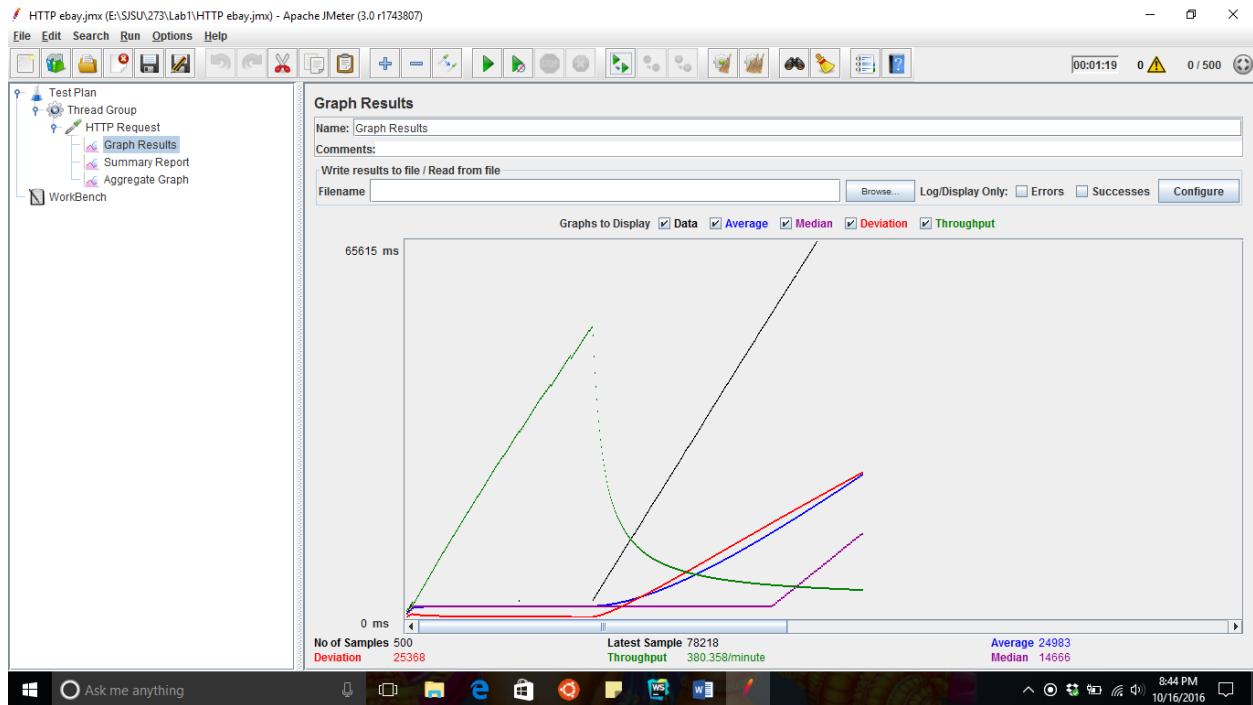
## 3)300 concurrent users



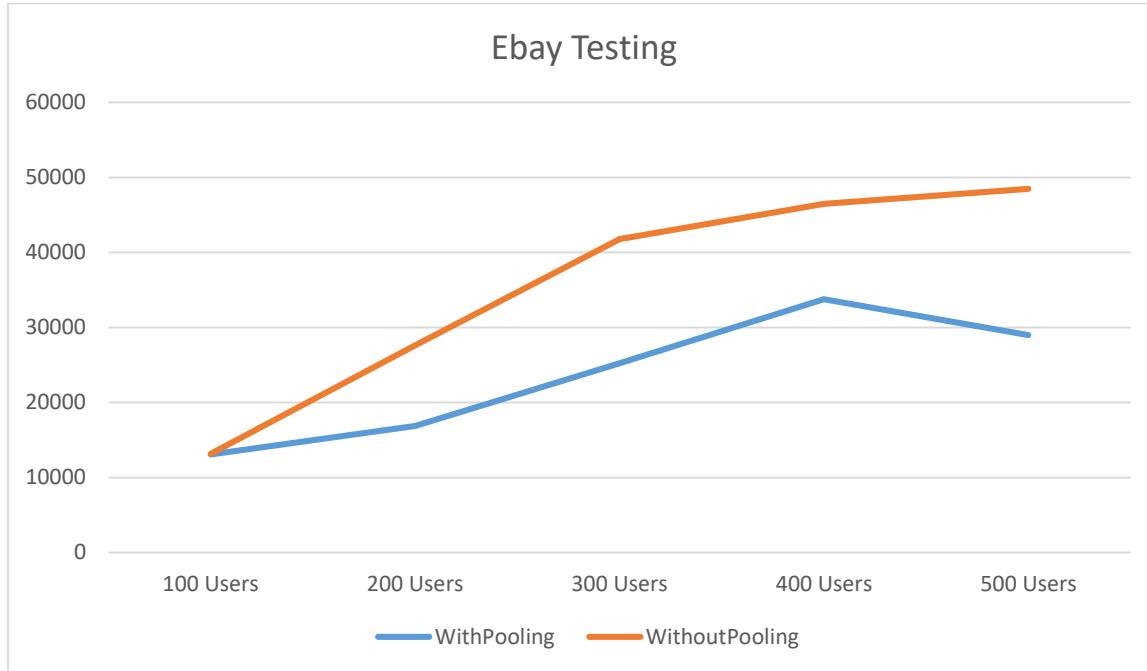
## 4)400 concurrent users



## 5) 500 concurrent Users



- **Analysis Graph**



## Explanation

It can be observed from the graphs shown above that if the connection pooling is implemented in the application then the average time for request and response decreases to a considerable amount. When multiple concurrent users are requesting for resource; then connection pooling will be very useful because it reduces the number of times new connections are created. Therefore, the process of getting a connection also becomes faster.

In my application I have made a pool which allocates connections to database. At present, I have kept the pool size 500 and queue size 100. When a new connection is needed it will simply borrow the connection from the pool and release as soon as the work gets completed.

```

var pool = [];
var count = 0;
var queue = [];

var queueasMap = new Map();

var pSize = 500;
var qSize = 100;

function CreateConnectionPool() {
    for (var i = 0; i < pSize; i++) {
        var connection = mysql.createConnection({
            host: 'localhost',
            user: 'root',
            password: 'root',
            database: 'ebay'
        });
        pool.push(connection);
    }
}

function getConnection(callback) {
    console.log("connection requested");

    if (isConnectionFree()) {
        console.log("connection free");
        callback(pool.pop());
    } else {
        console.log("connection not free");
        if (isQueueFree()) {

            console.log('in queue');
            queue.push(count);
            queueasMap.set(count, false);
            token = count;
            count++;
        }
    }
}

function releaseConnection(connection) {
    pool.push(connection);
    console.log('connection is released');
    queueasMap.set(queue.pop(), true);
    queue.shift();
}

```

```

    }

    function isConnectionFree() {
        if (pool.length <= 0)
            return false;
        else
            return true;
    }

    function isQueueFree() {
        if (queue.length >= qSize)
            return false;
        else
            return true;
    }
}

```

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

- In the application; clients make lots of requests to external APIs.
- Therefore caching is very important to provide a good experience to our users.
- When the new request from the client is cached, we can only cache the URL and not the entire API call.
- This can help to avoid the overloading of the cache so that the performance can remain stabilized.
- When a URL is requested for the first time, a new entry will be created for that URL along with timestamp and index.
- When this URL is requested again, it will try to fetch it from the cache itself. If it is present in cache, then the only need is to update the timestamp.
- So using this way, we can easily implement request caching in order to access the request faster and ultimately increasing the performance.