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

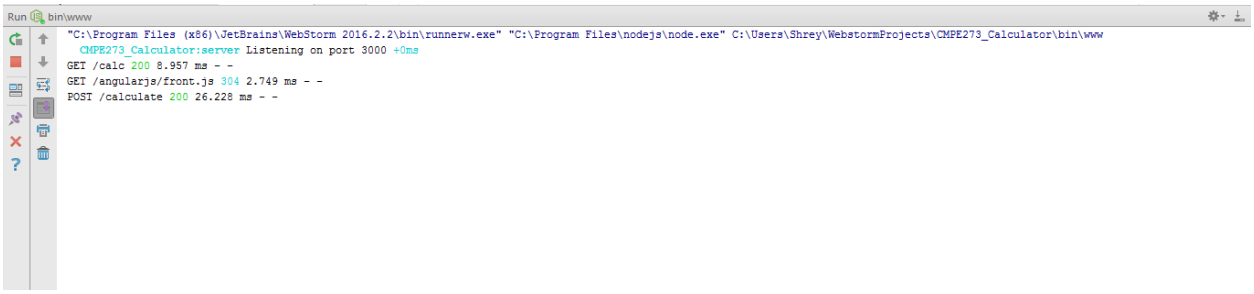
ADD function is shown below:

Any of the 2 values can’t be NaN:

Divide by 0 will display result as INFINITY:

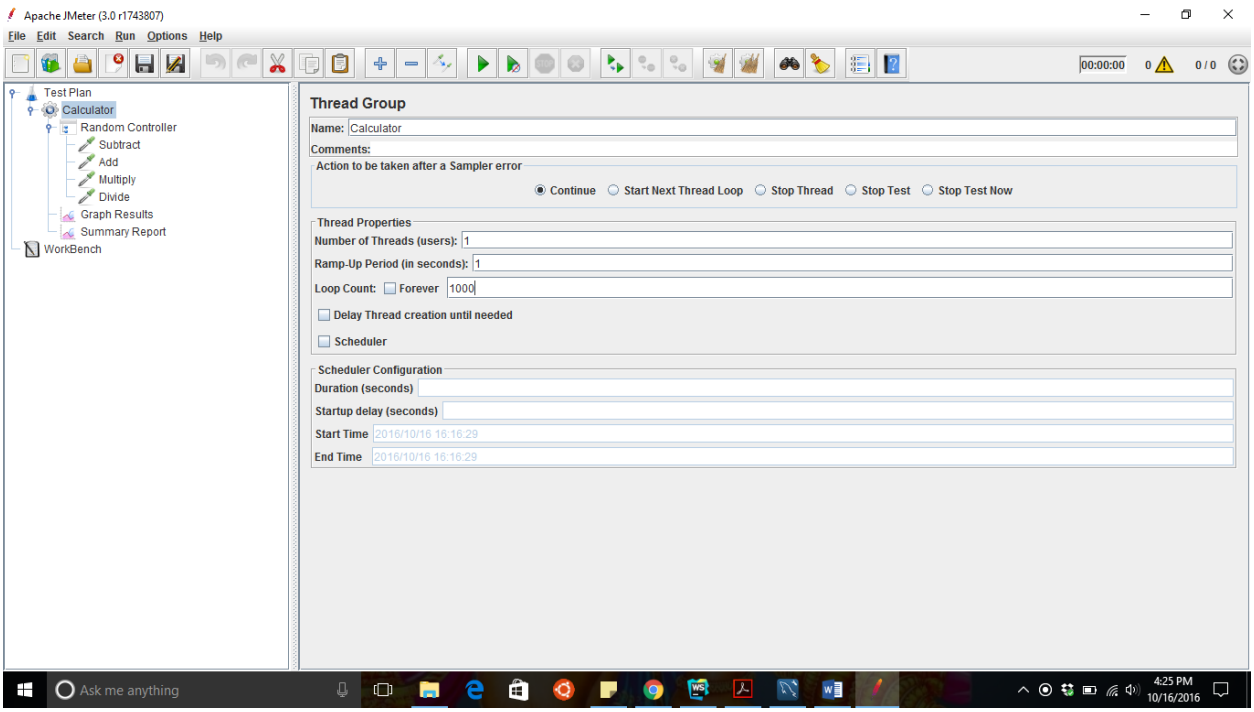
Divide 0 by 0 will display result as UNDEFINED:

Server of Calculator

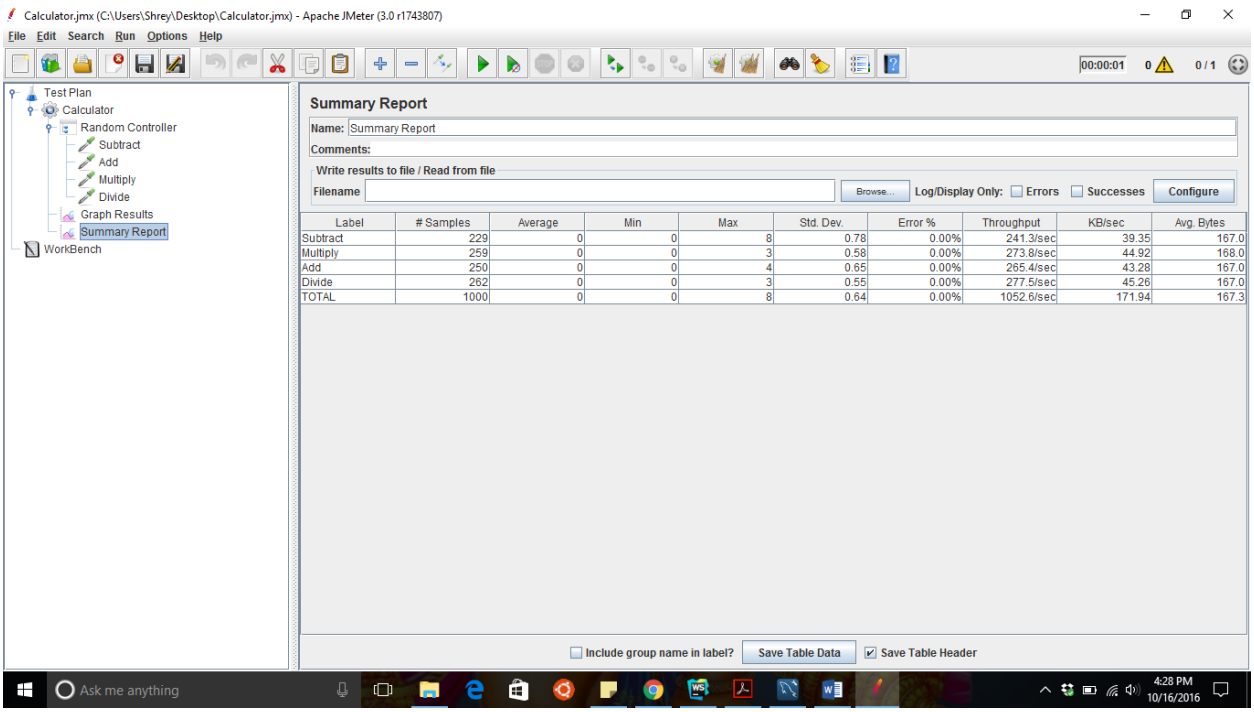


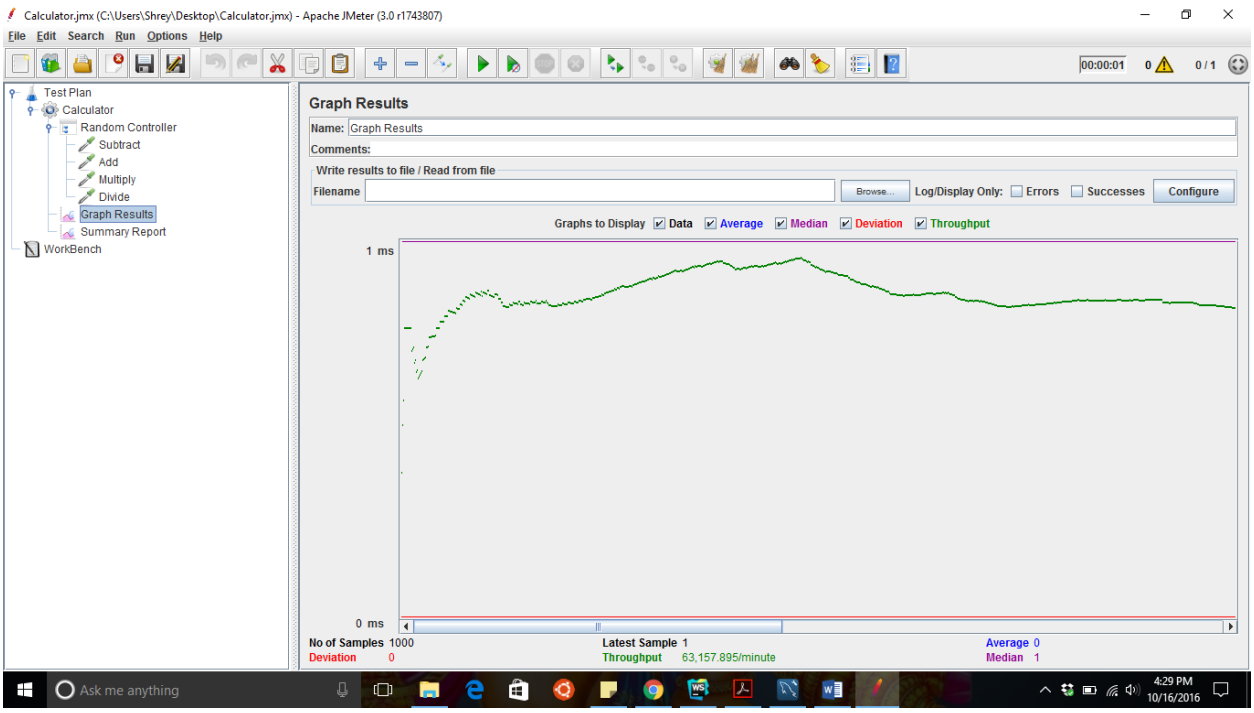
Testing & Performance Graphs

Jmeter Setup of HTTP Request

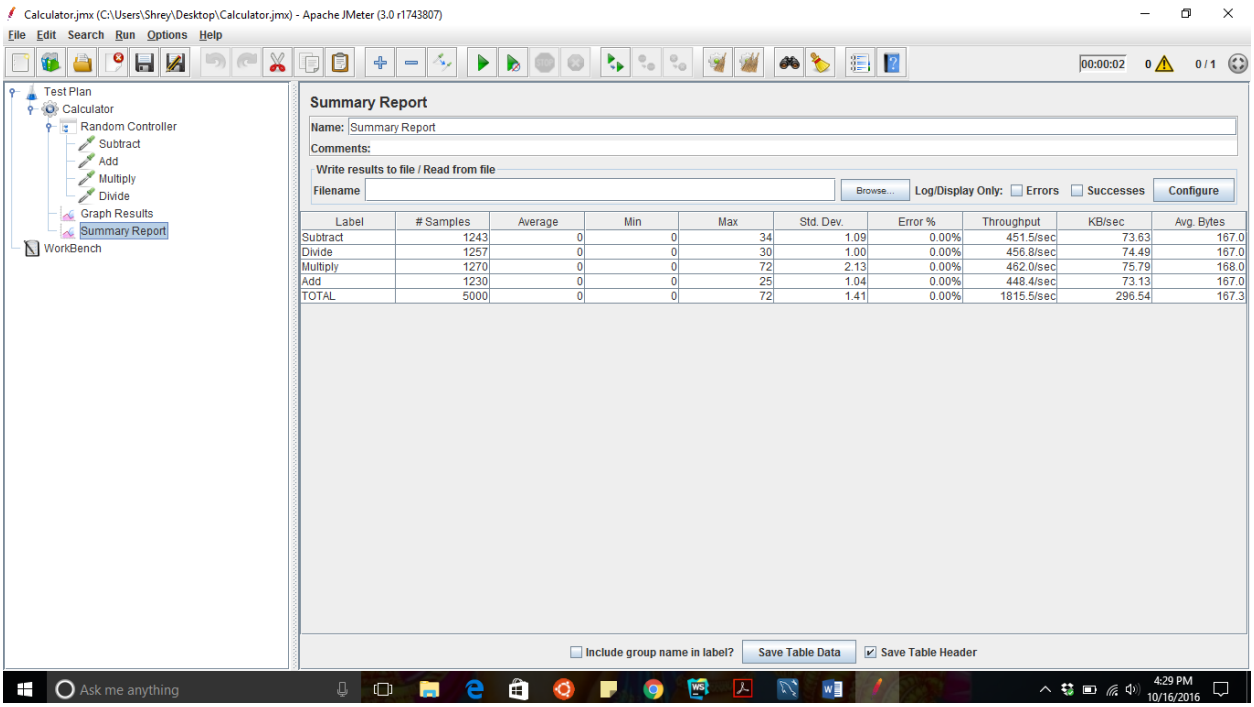


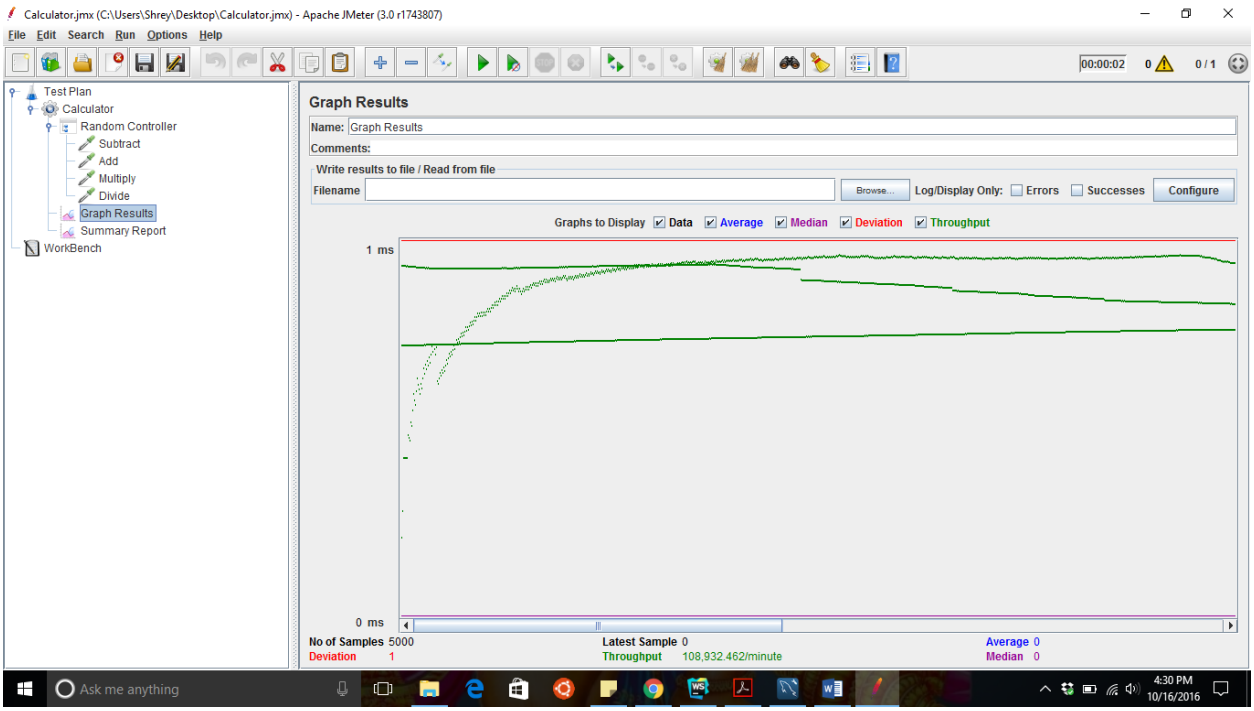
`1)1000 calls


Calculator.jmx (C:\Users\Shrey\Desktop\Calculator.jmx) - Apache JMeter (3.0 r1743807)
File Edit Search Run Options Help
00:00:01 0 0 / 1
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

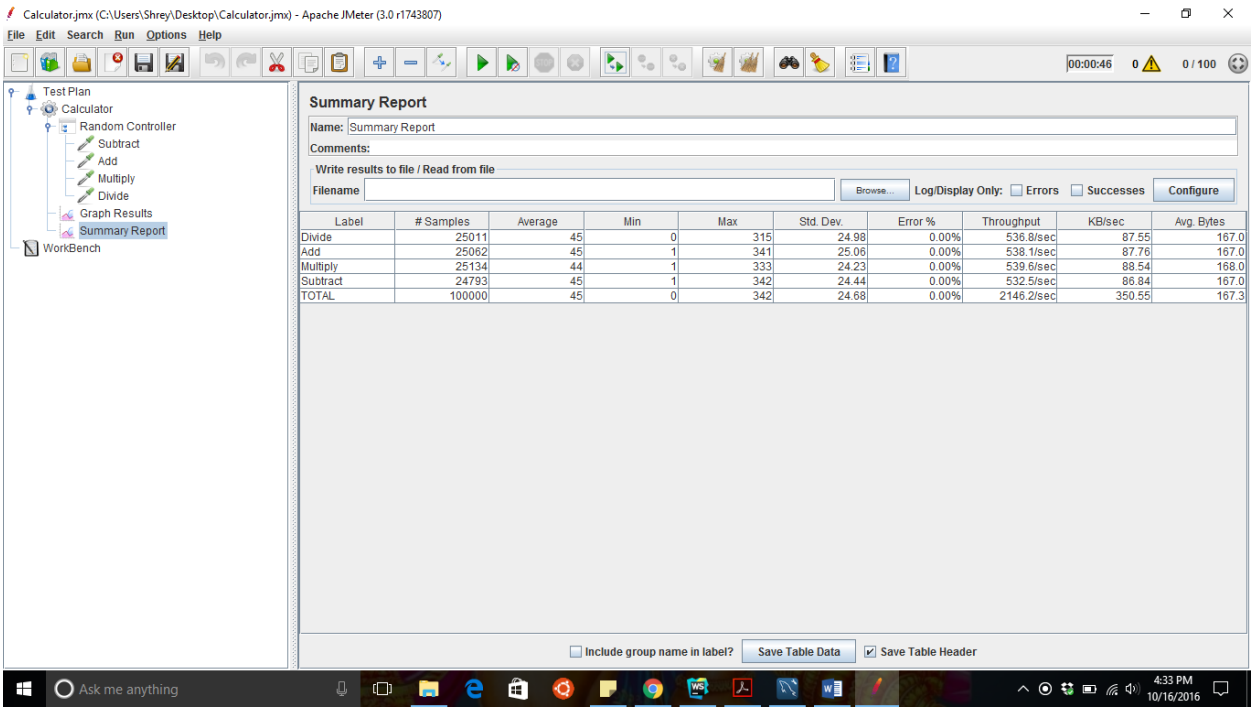

Calculator.jmx (C:\Users\Shrey\Desktop\Calculator.jmx) - Apache JMeter (3.0 r1743807)
File Edit Search Run Options Help
00:00:01 0 0 / 1
Test Plan
Calculator
Random Controller
Subtract
Add
Multiply
Divide
Graph Results
Summary Report
WorkBench
Graph Results
Name: Graph Results
Comments:
Write results to file / Read from file
Filename
Browse...
Log/Display Only: Errors Successes
Configure
Graphs to Display Data Average Median Deviation Throughput
1 ms
0 ms
No of Samples 1000
Latest Sample 1
Average 0
Deviation 0
Throughput 63,157.895/minute
Median 1
Ask me anything
4:29 PM
10/16/2016

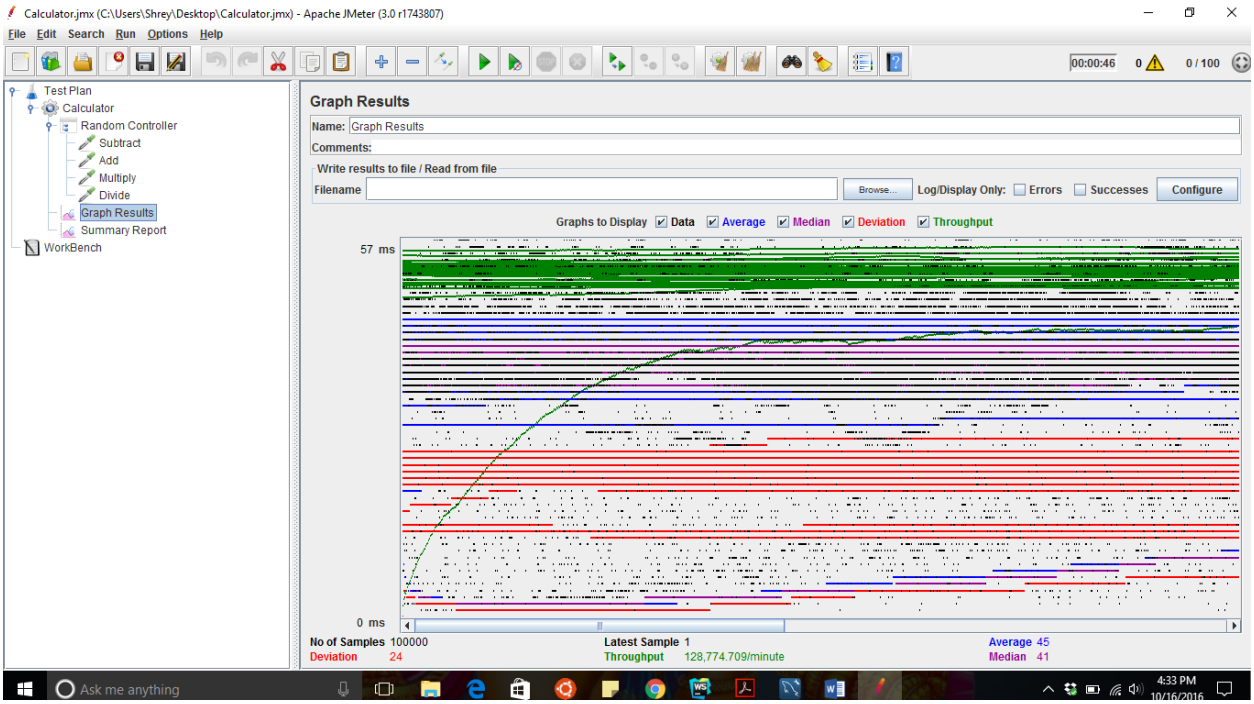
2)5000 calls


Calculator.jmx (C:\Users\Shrey\Desktop\Calculator.jmx) - Apache JMeter (3.0 r1743807)
File Edit Search Run Options Help
00:00:02 0 0 / 1
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 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 Data
Save Table Header
Ask me anything
4:29 PM
10/16/2016

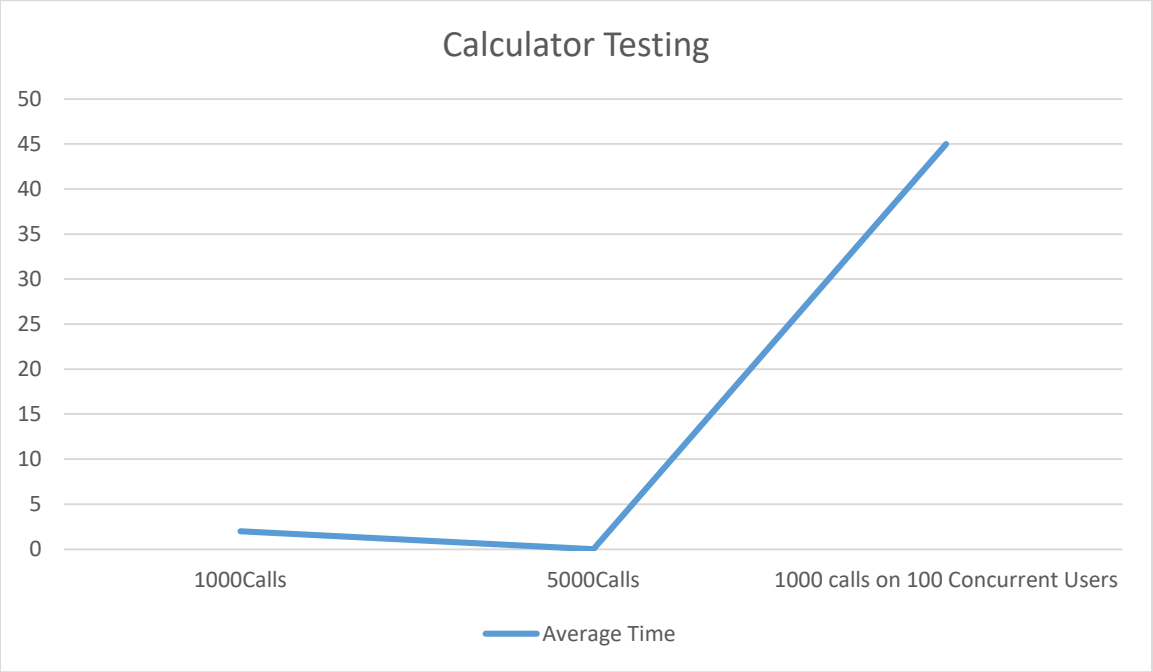

Calculator.jmx (C:\Users\Shrey\Desktop\Calculator.jmx) - Apache JMeter (3.0 r1743807)
File Edit Search Run Options Help
00:00:02 0 0 / 1
Test Plan
Calculator
Random Controller
Subtract
Add
Multiply
Divide
Graph Results
Summary Report
WorkBench
Graph Results
Name: Graph Results
Comments:
Write results to file / Read from file
Filename
Browse...
Log/Display Only: Errors Successes
Configure
Graphs to Display Data Average Median Deviation Throughput
1 ms
0 ms
No of Samples 5000
Latest Sample 0
Average 0
Deviation 1
Throughput 108,932.462/minute
Median 0
Ask me anything
4:30 PM
10/16/2016

3)1000 calls on 100 concurrent users





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

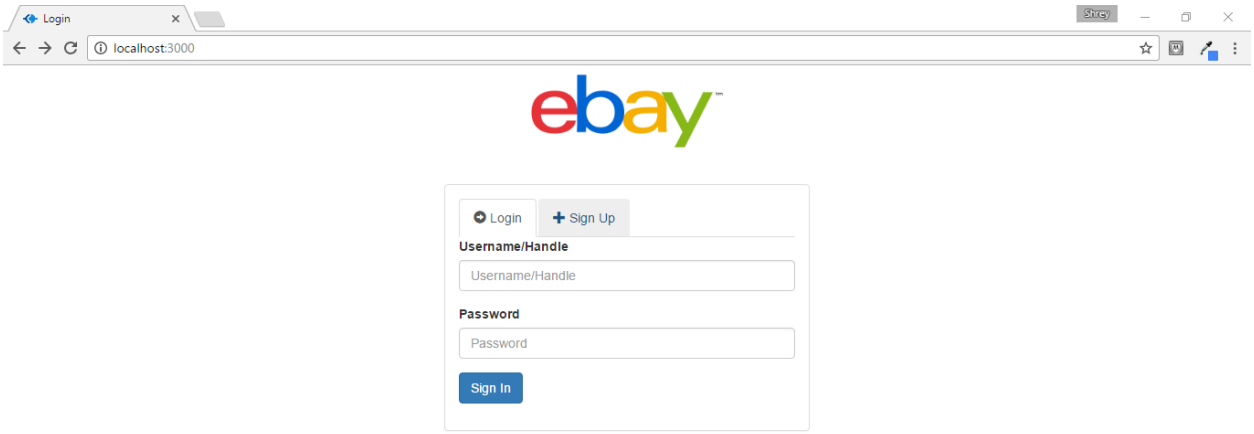
1. Main page which will show the login and signup options.
2. Home page which will be shown only after successful login. It will display All Advertisements which are posted by other users.
3.All bids page
4. Profile page which shows further details of the user.
5.Advertisement posting page
6.Bid posting 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.

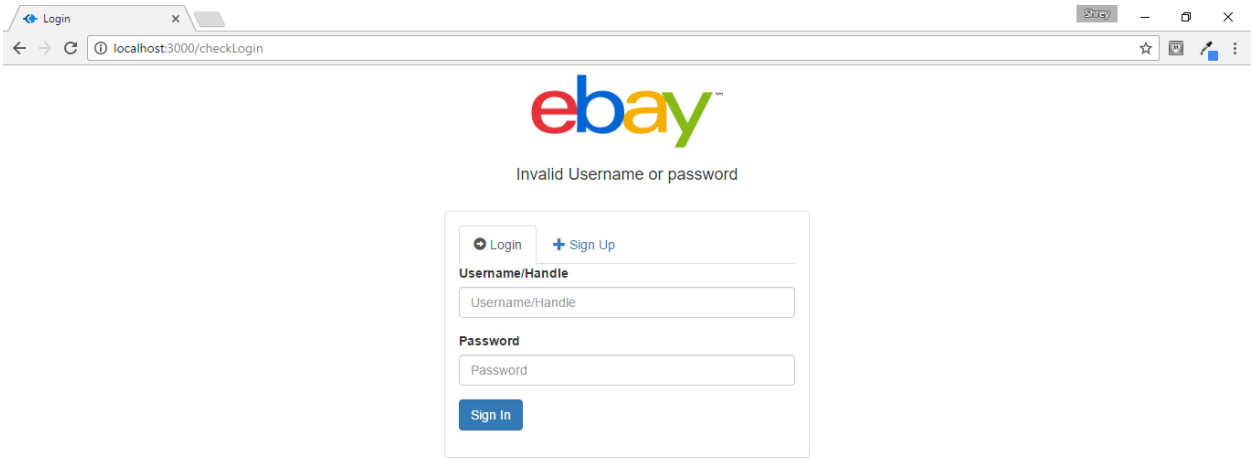
- Frond End: Angular.js,Bootstrap,Html
- Back End: Node.js
- Database: MySQL

Results

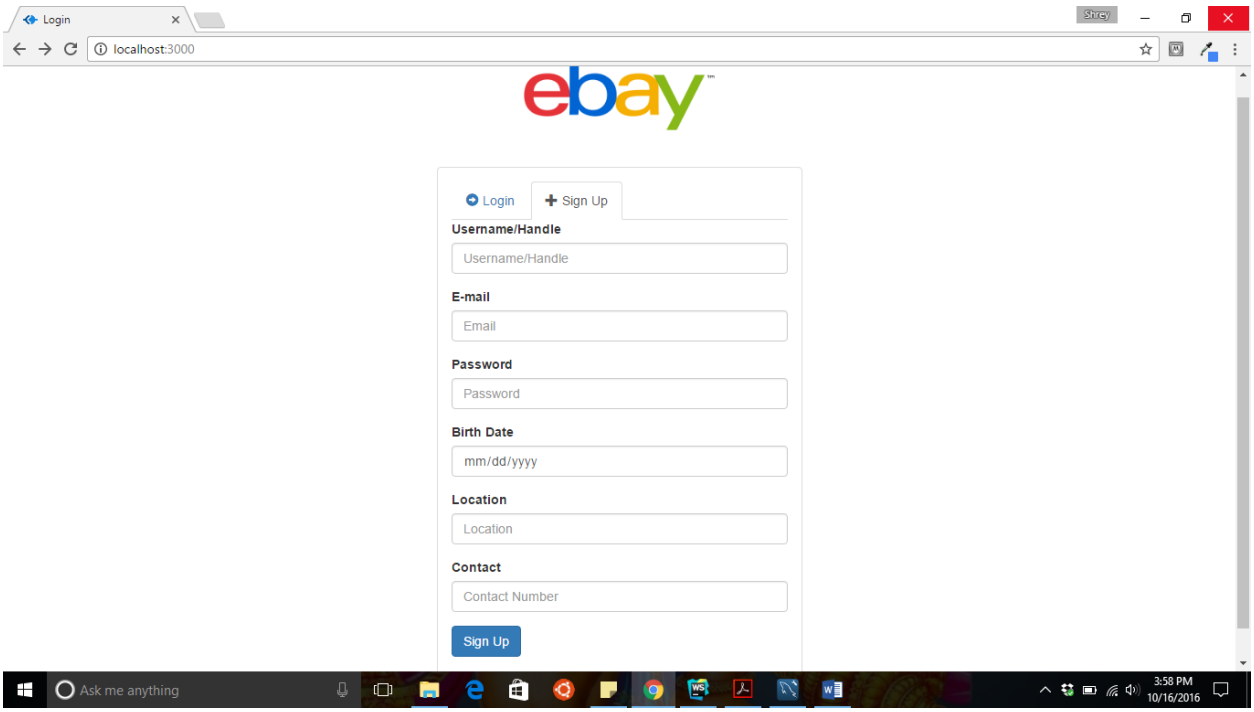
- **Screenshots**
- **Login Page**
- First page to be displayed after running the application. User Can not use the application without Logging in.



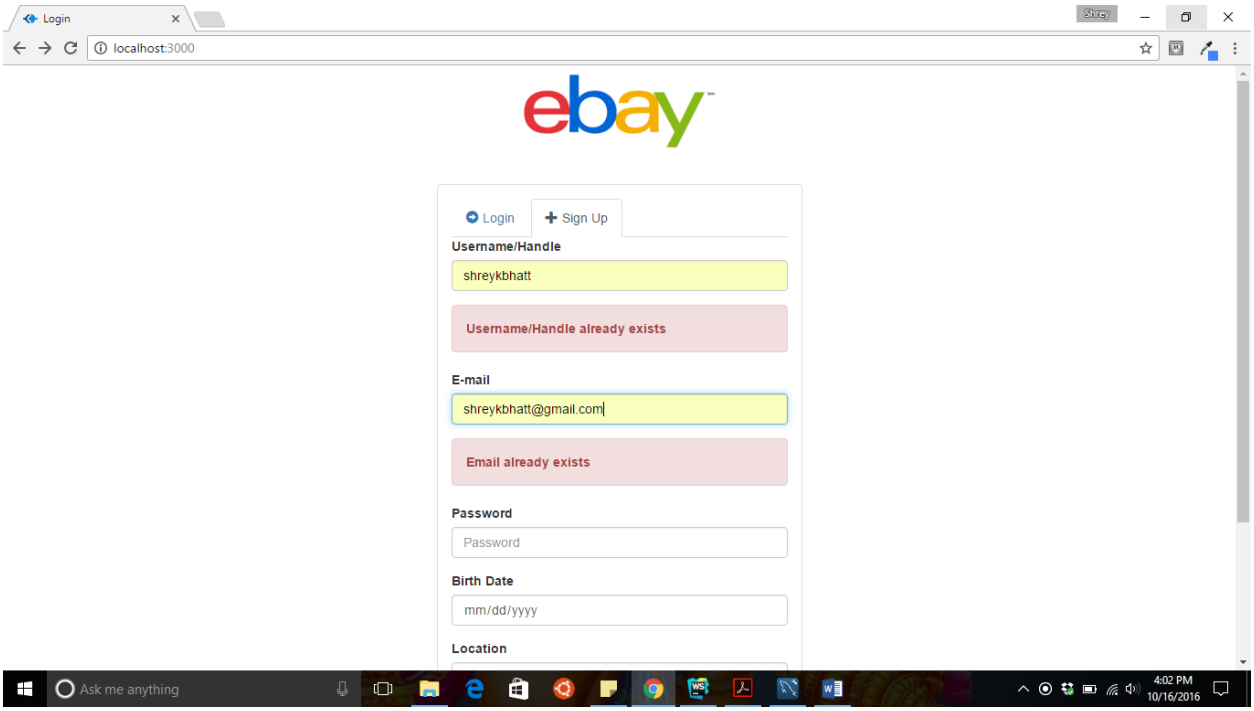
- **Invalid Username/password**
- If the username/password is not correct it will display “invalid username or password” message.



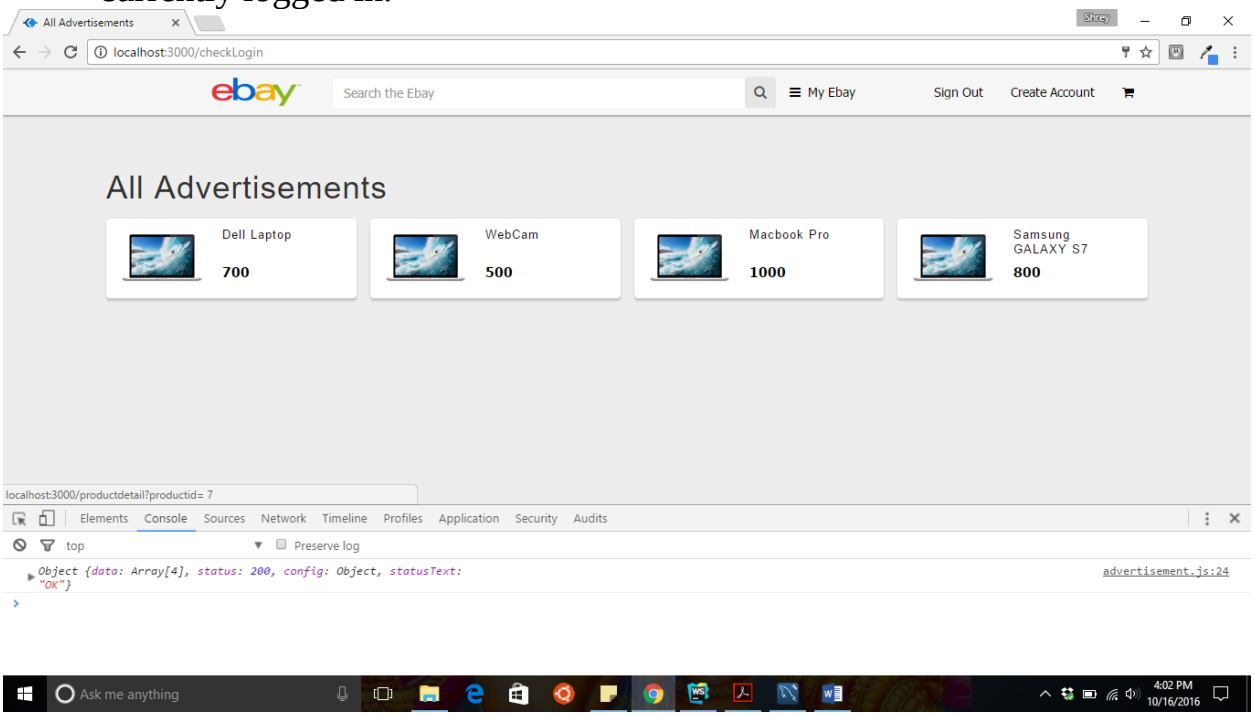
- **SignUp/Register Page**
- Enter Username/Ebay Handle,E-mail,Password,BirthDate,Location and Contact Details.



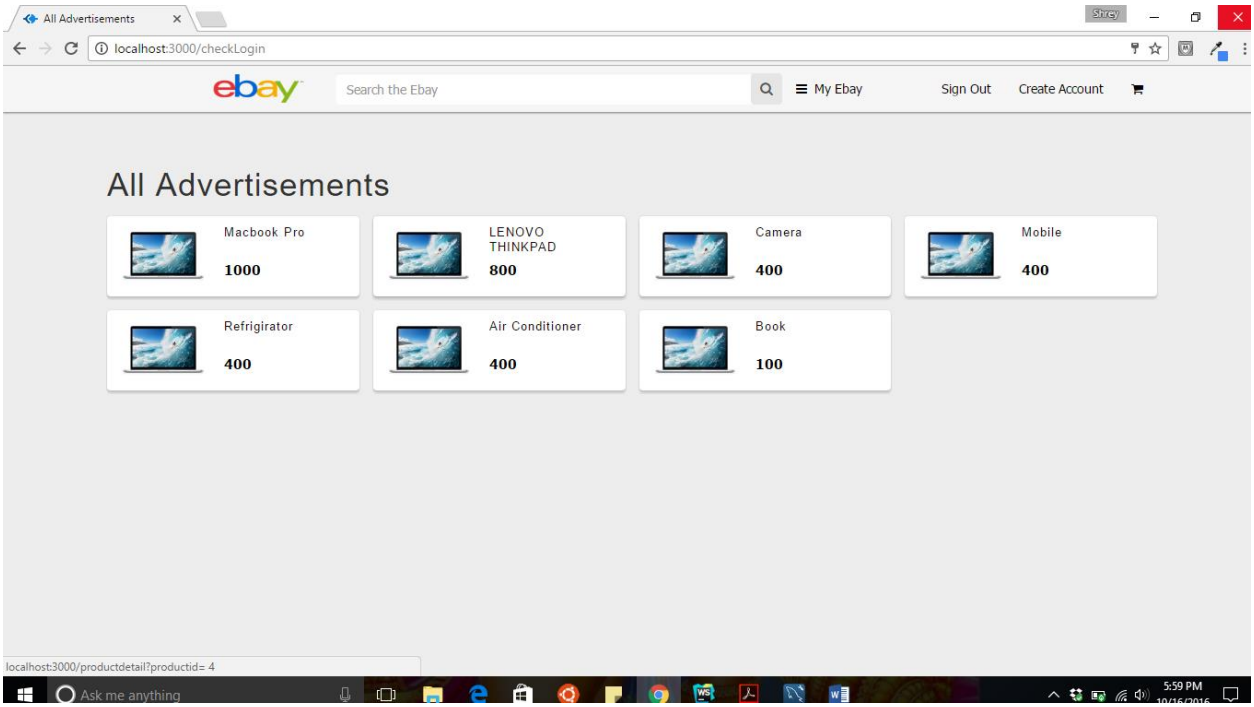
- If the User Already Exists then It will not allow to add.



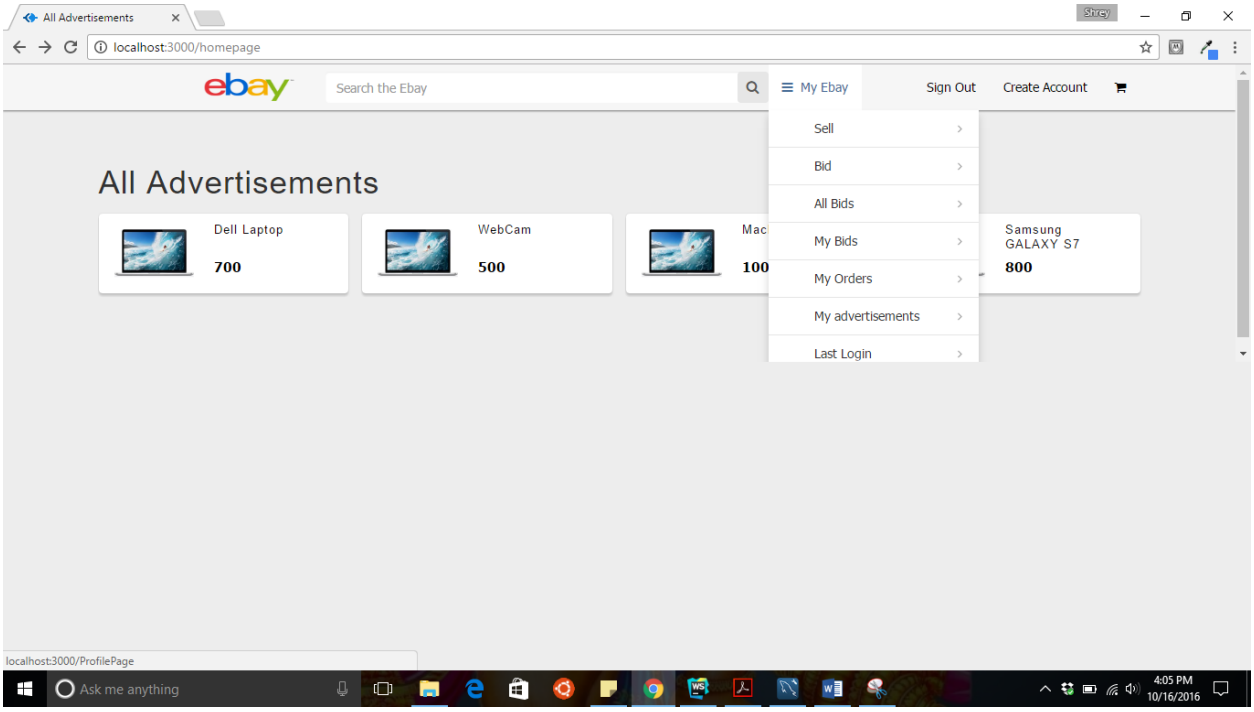
- **After Login Screen**
- It will display all the advertisement which are not posted by the usesr who is currently logged in.



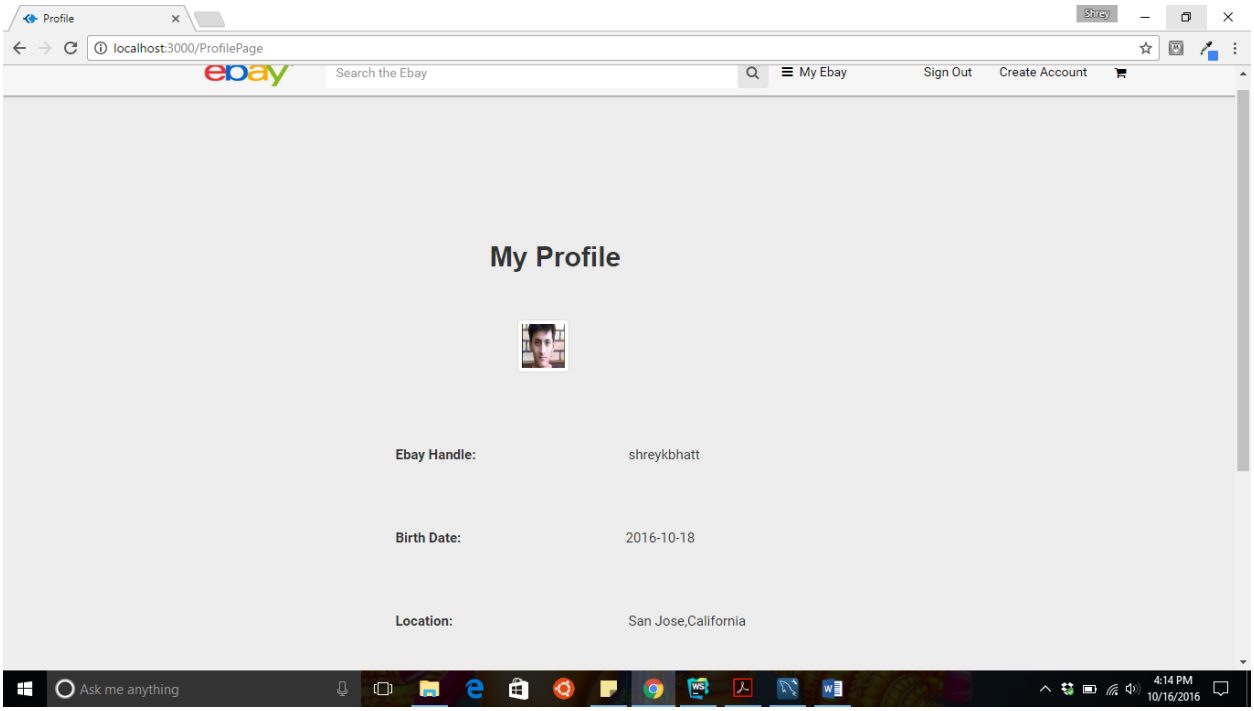
- **HomePage**
- On clicking the Ebay icon the Homepage will be opened.
- It will display All the advertisements which are not posted by the user who is currently logged in.



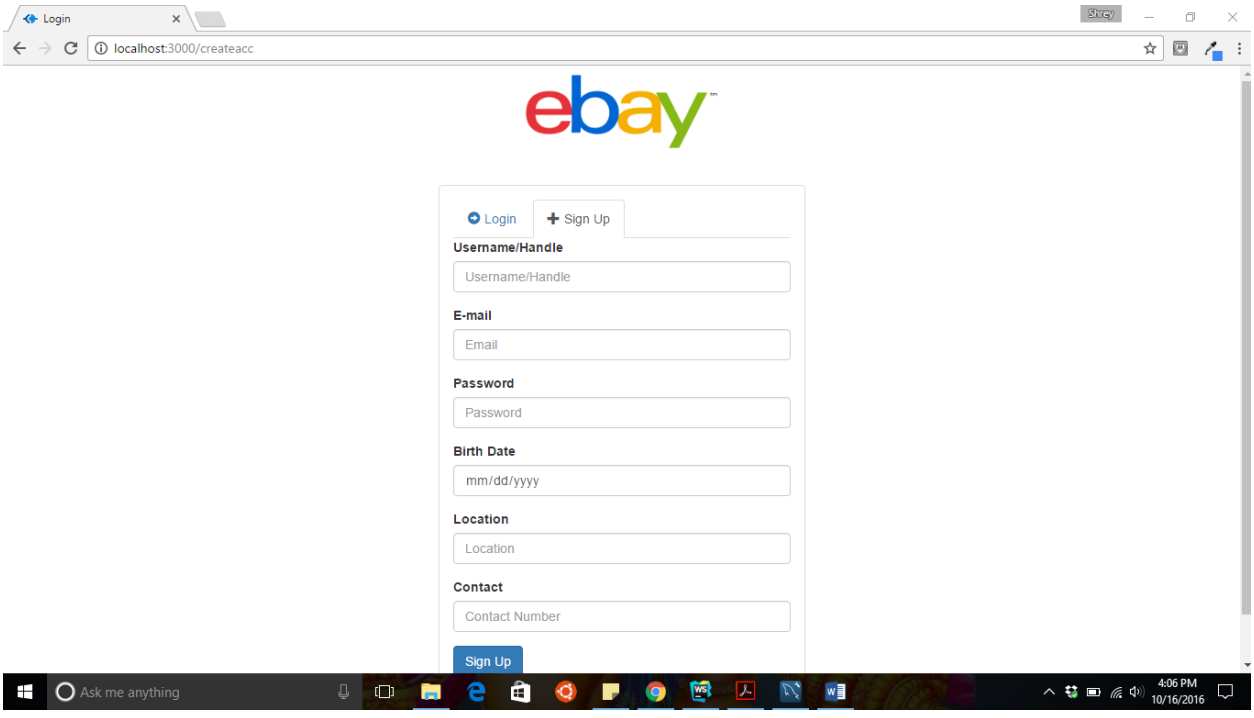
- **Menu Pops Up on clicking the MyEbay**



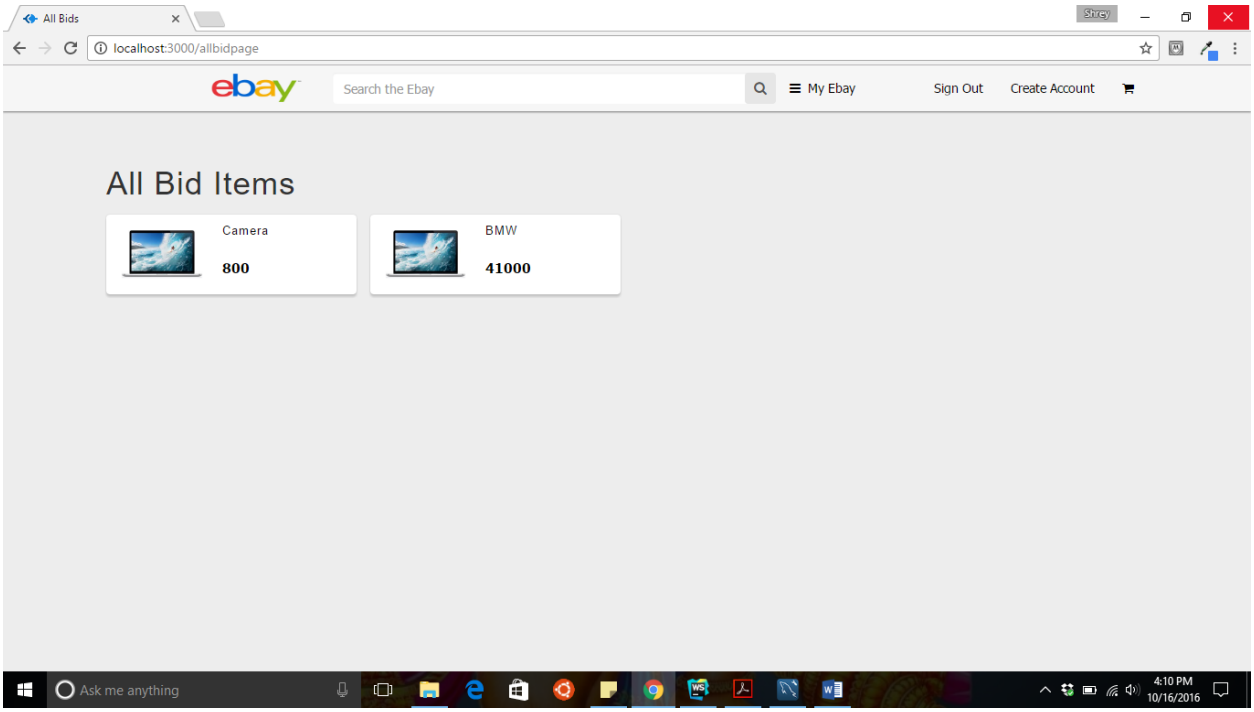
- **ProfilePage**



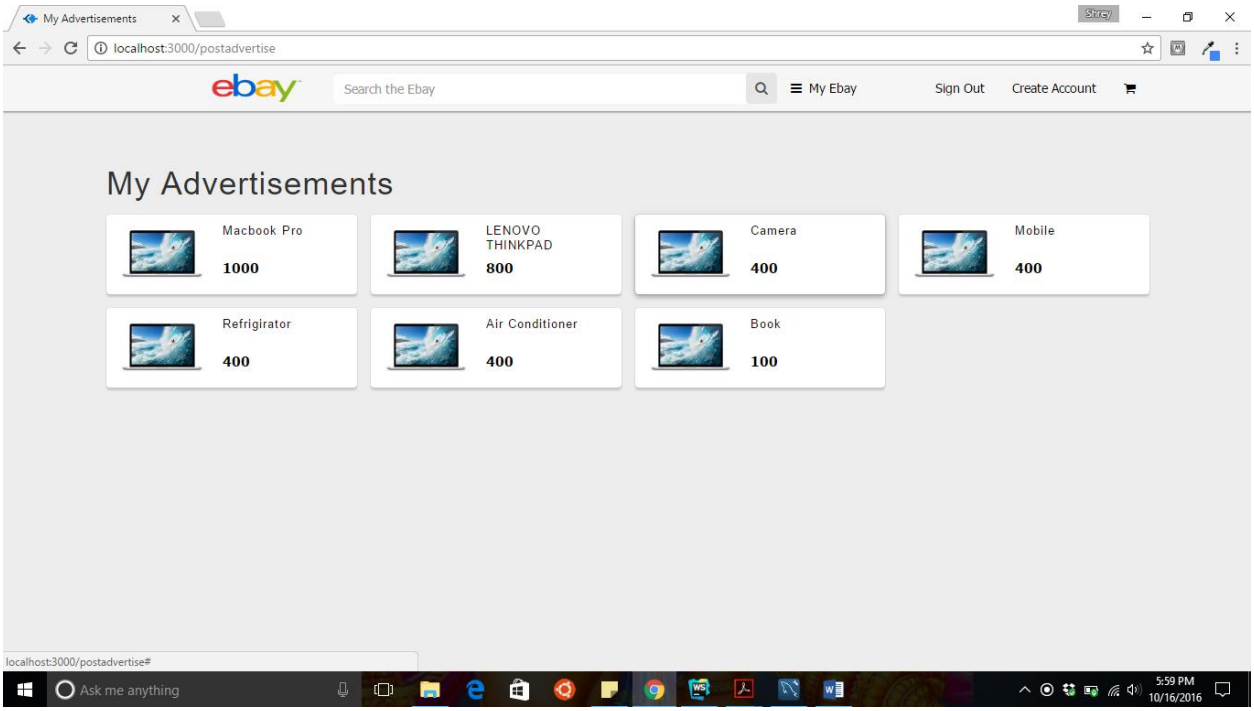
- **Create Account**
- On clicking the create account page in the title bar the registration/SignUp page appears.



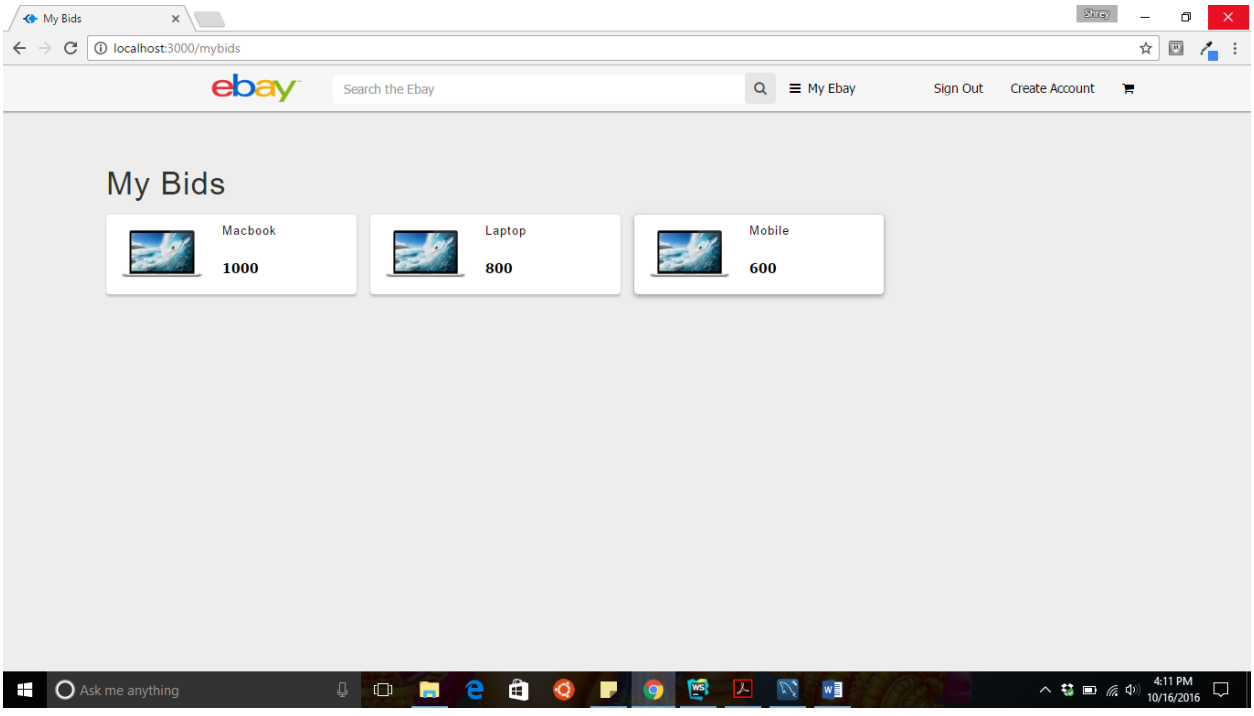
- **All Bids**
- This page displays all the products available for bidding which are not posted by the user who is currently logged in.



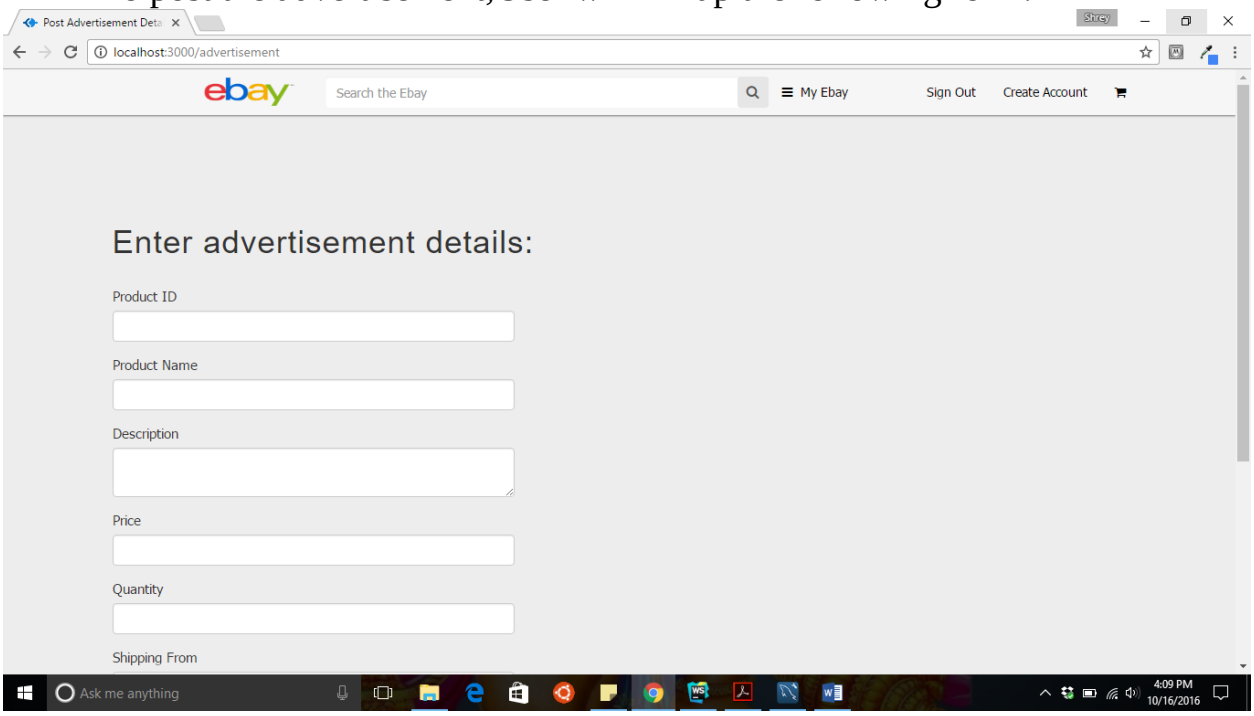
- **My Advertisements**
- This page displays all the advertisements which are posted by the user who is currently logged in.



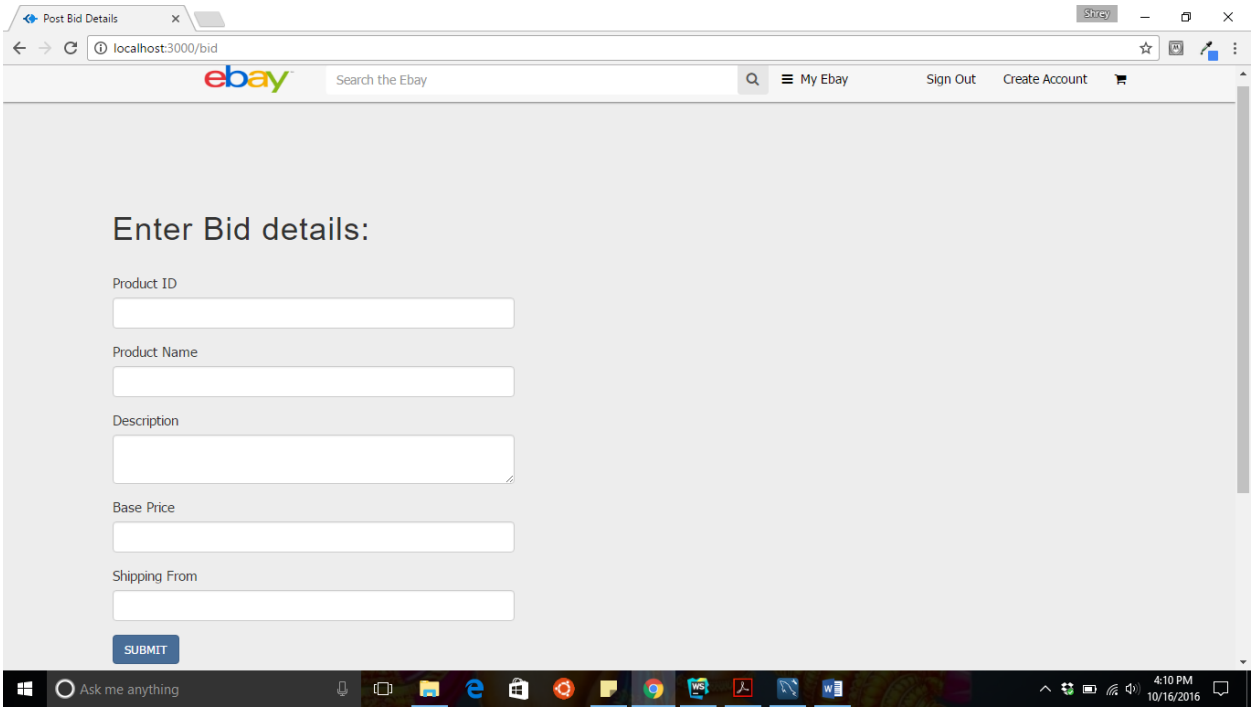
- **My Bids**
- This page displays all the products available for bidding which are posted by the user who is currently logged in.



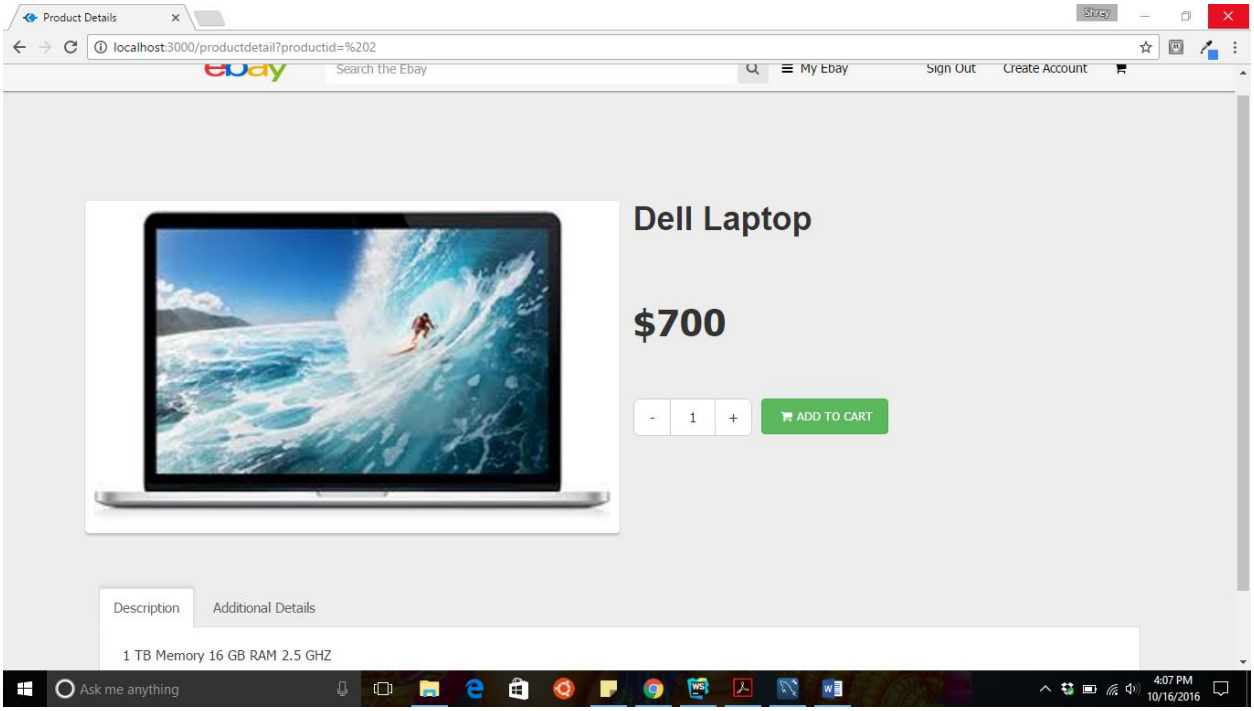
- **Post Advertise**
- To post the advertisement,User will fill up the following form.



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

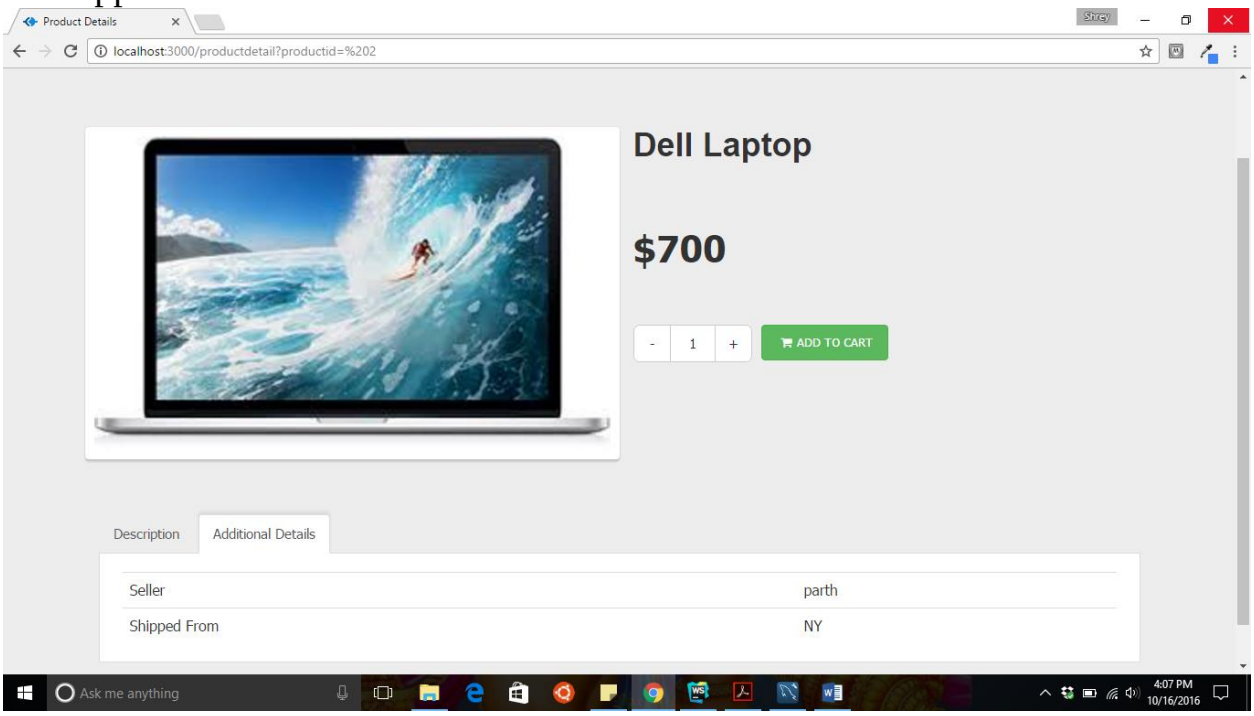


- **Product Details**
- It will display product details like Product name,Price and Quantity(can be edited) and product description.

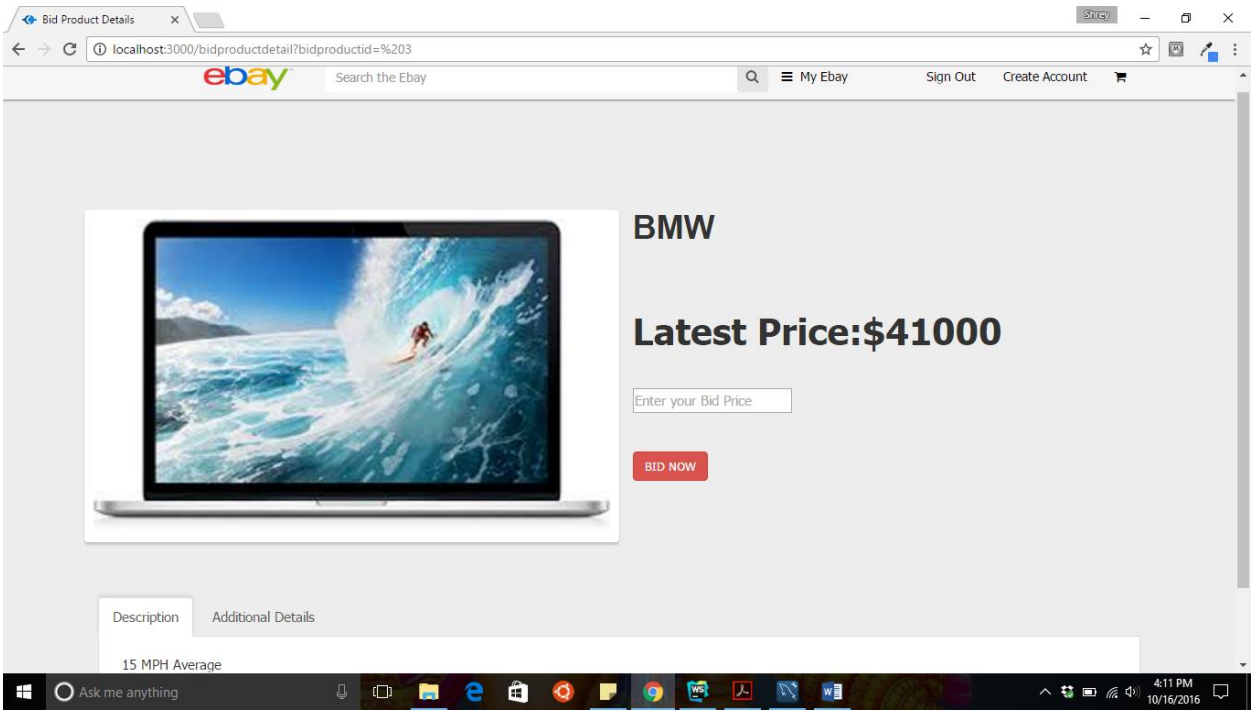


- **Additional Details**

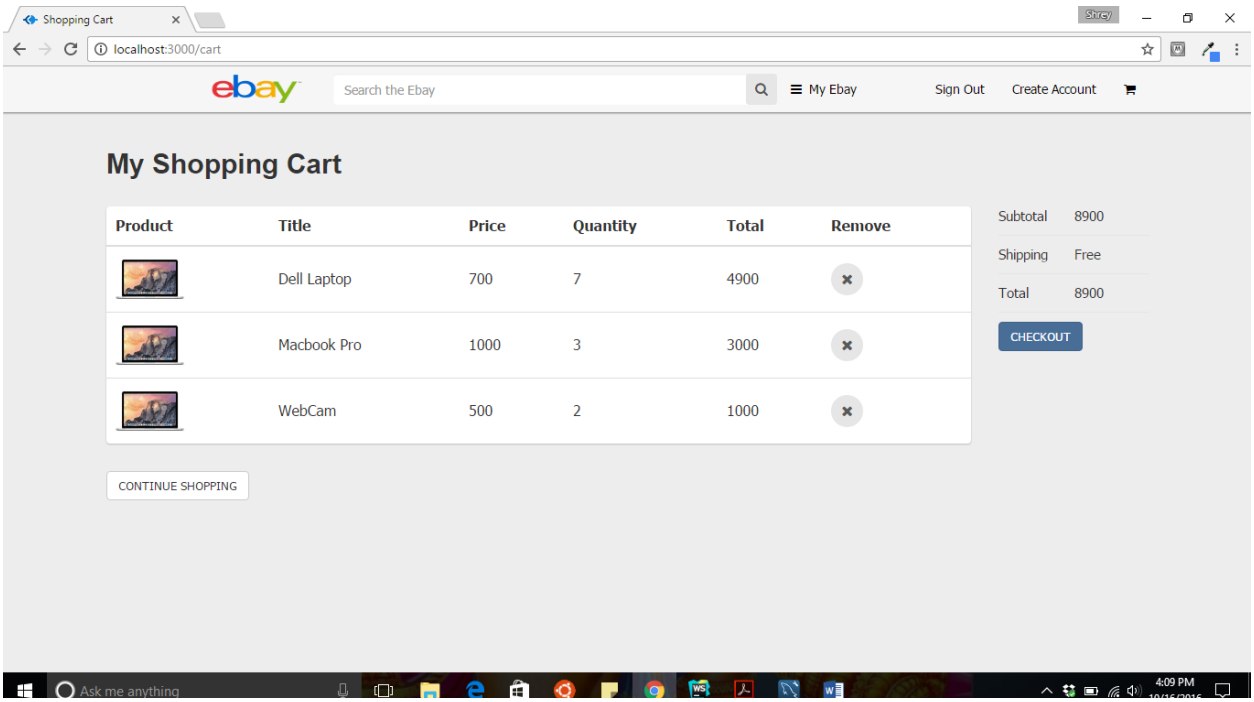
Additional details will contain Seller and the location from where the product is shipped.



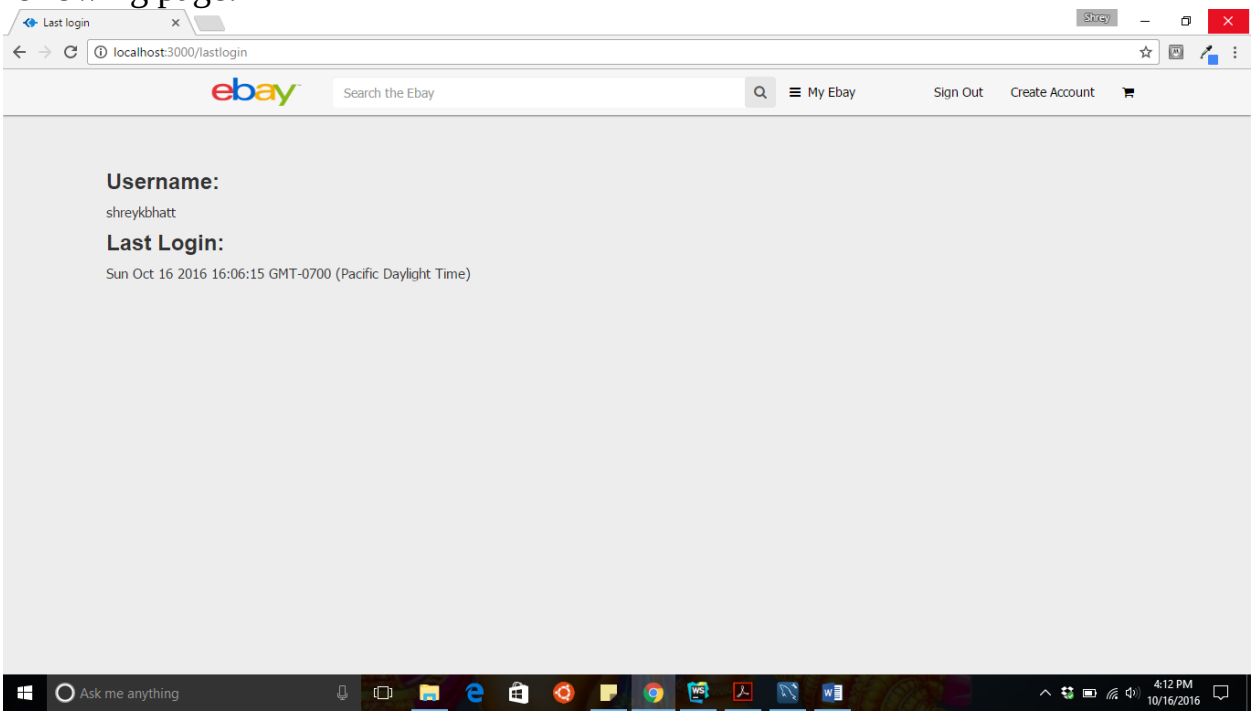
- **Bid Product Details**
- It will display product details like Product name, Base Price and product description.
- User can Bid on the product by entering the Bid price.After bidding the entry for that product and that particular user will be their in the bid and biddinghistory table.So when User will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



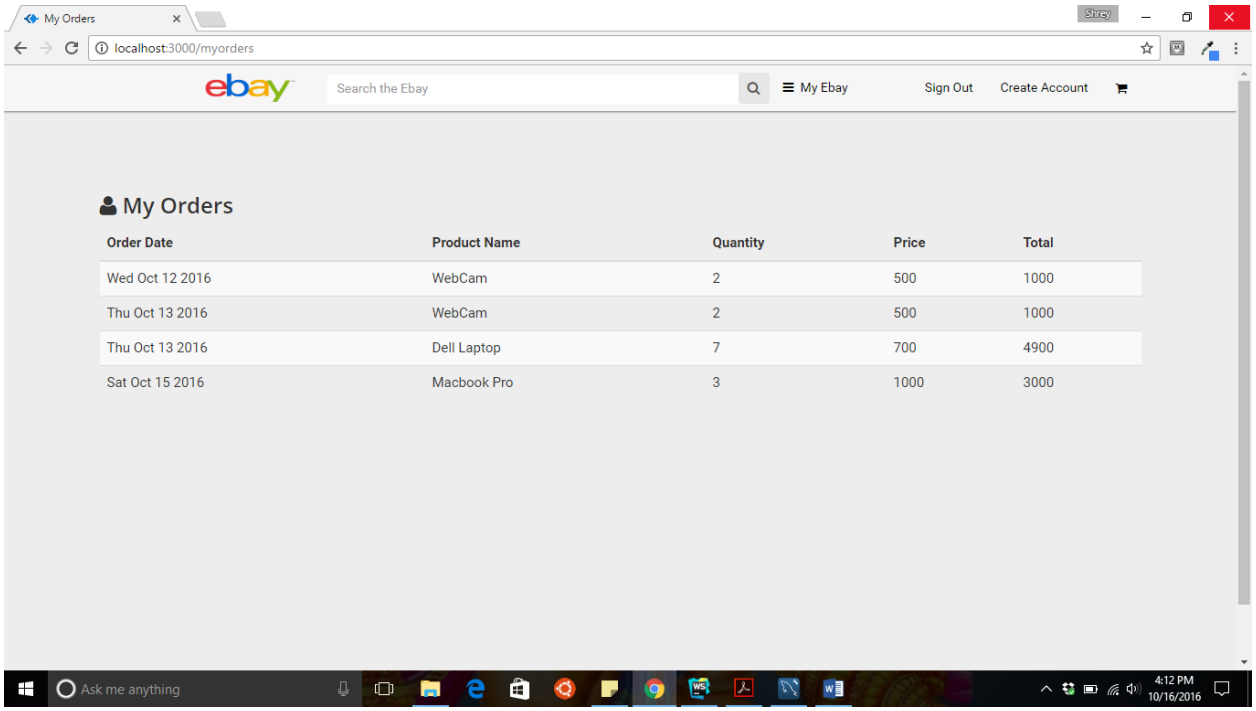
- **Shopping-Cart**
- User can add or remove items from the shopping card.



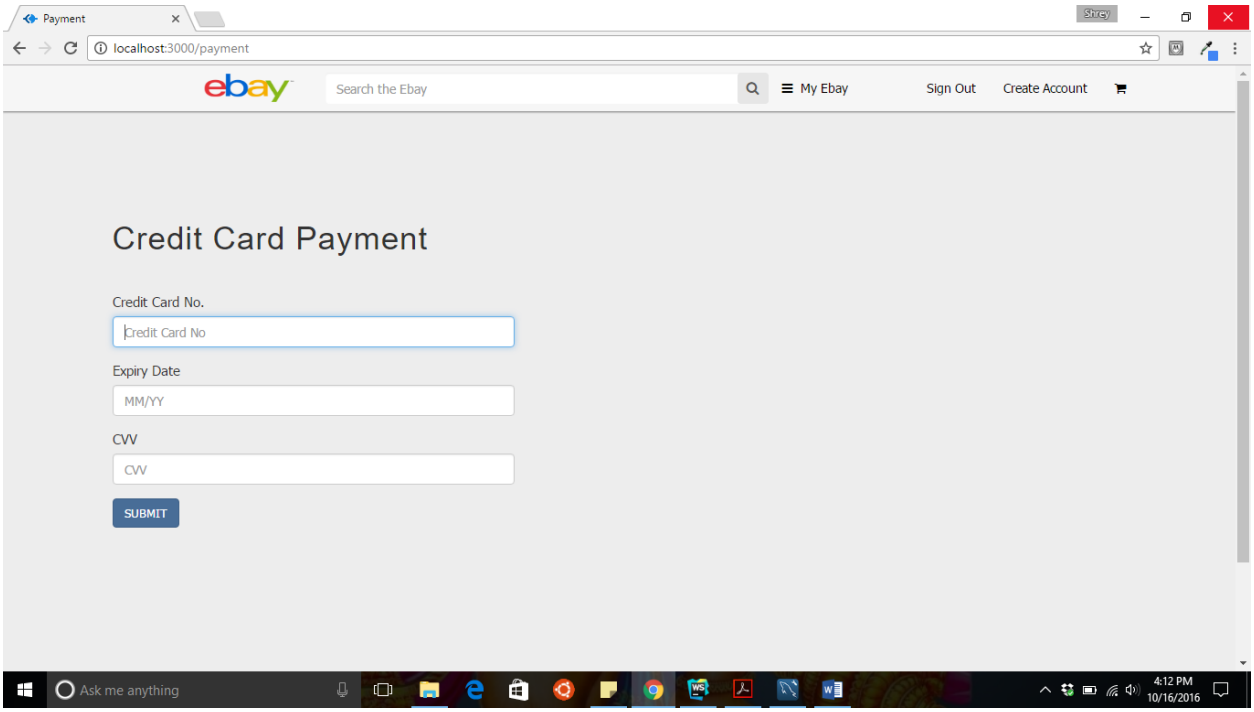
- **Last Login Page**
- Last Login time is stored in the user database. It will be updated everytime user logs out. So username and lastlogin time will be displayed in the following page.



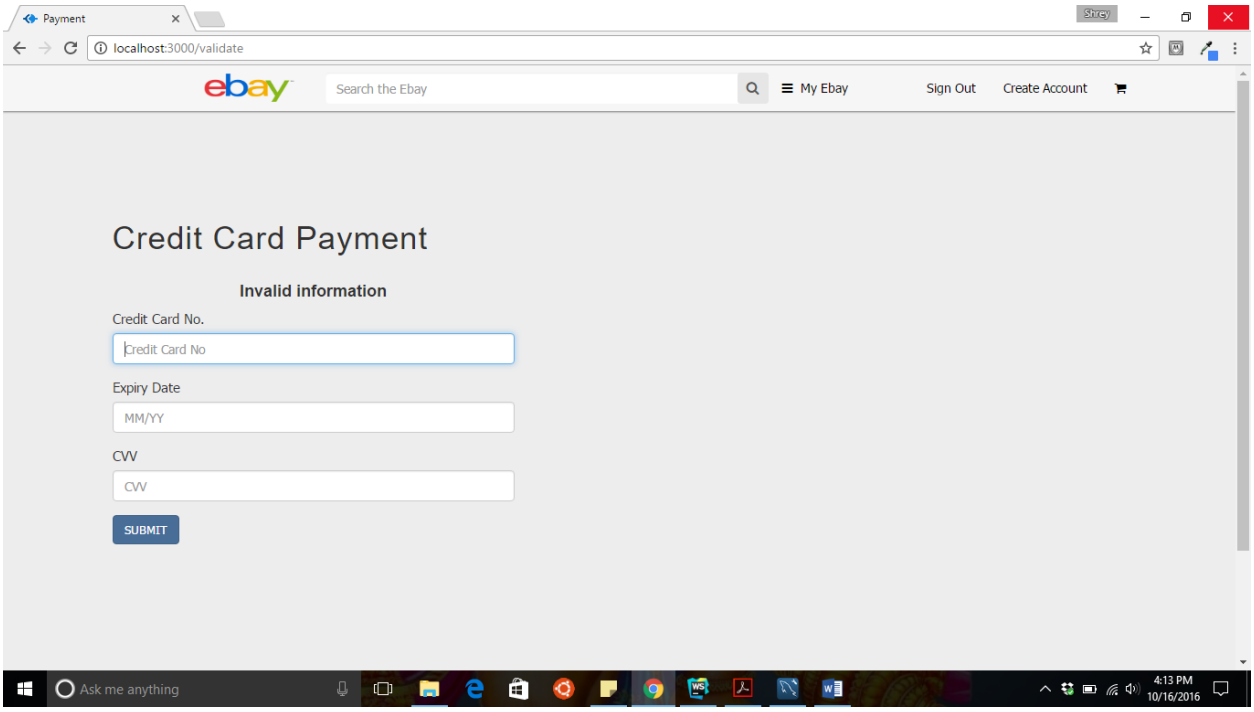
- **My Orders**
- Orders of the current user will be displayed on the following page.



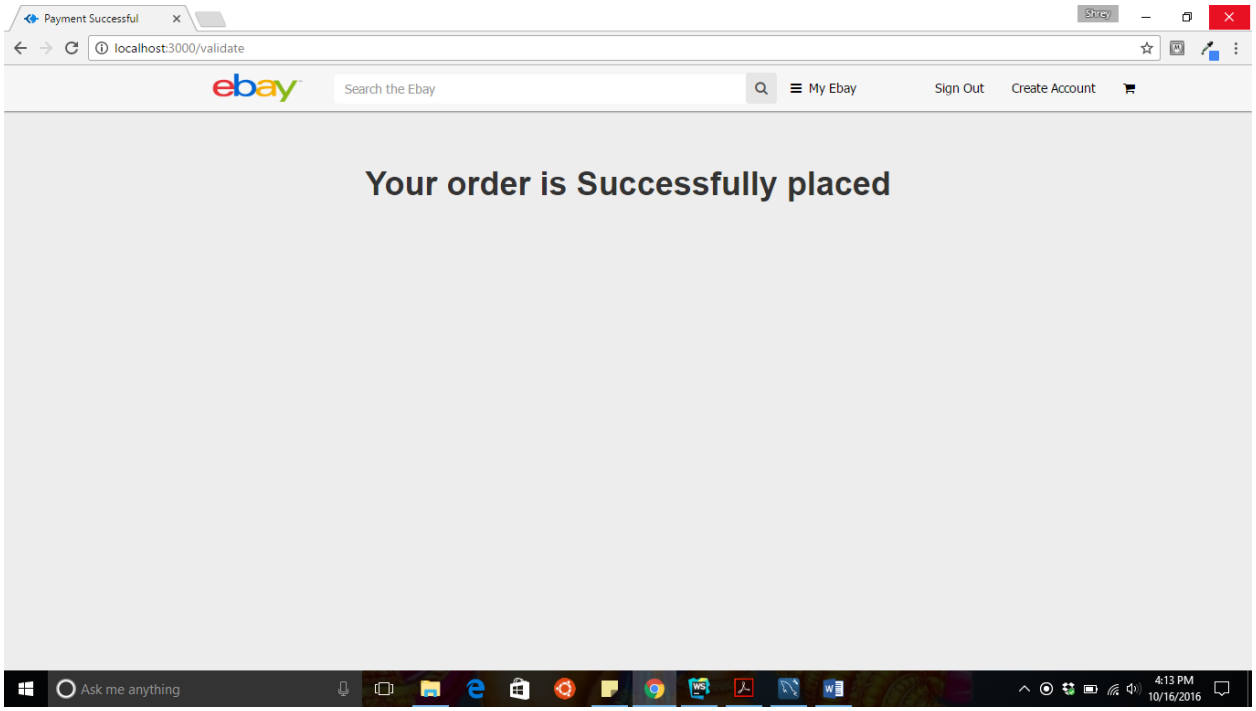
- **Credit Cart Validation**
- On clicking the checkout, the credit card payment page will be rendered.



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

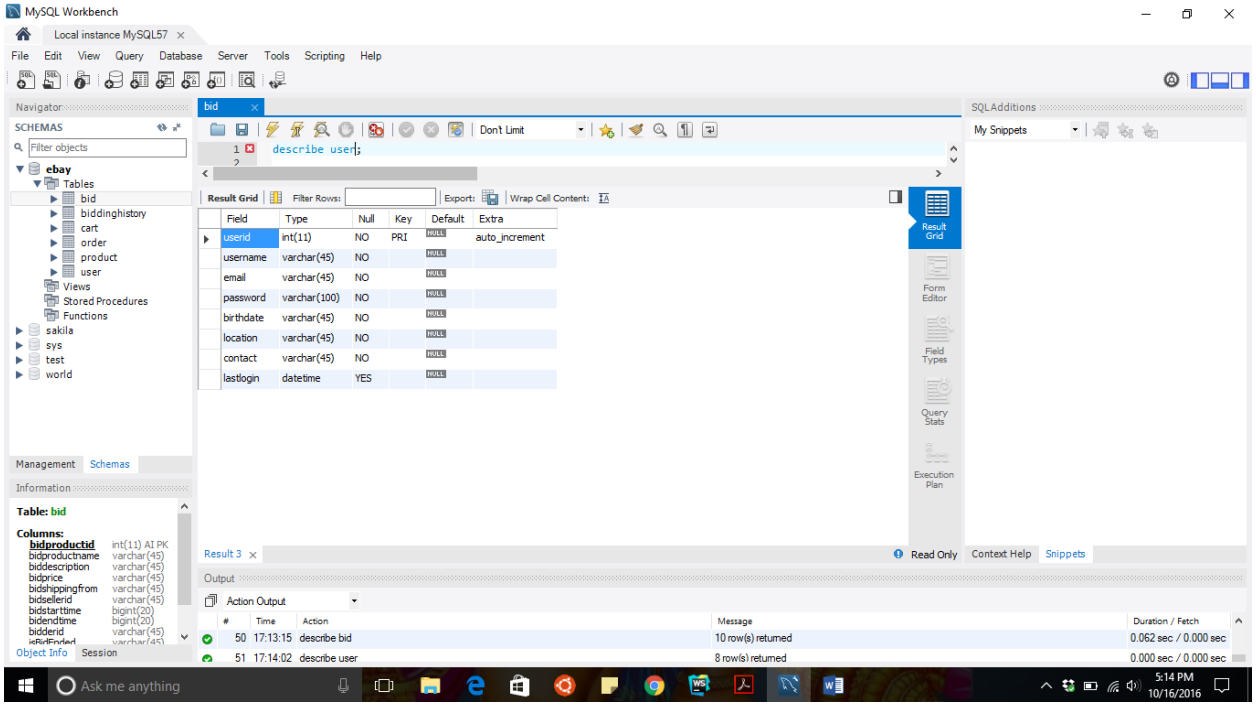


- **Payment Successful**
- If the payment will be successful the the “Payment Successful” page will be rendered.The cart will be empty,the order table will have entry for all the products for current user.

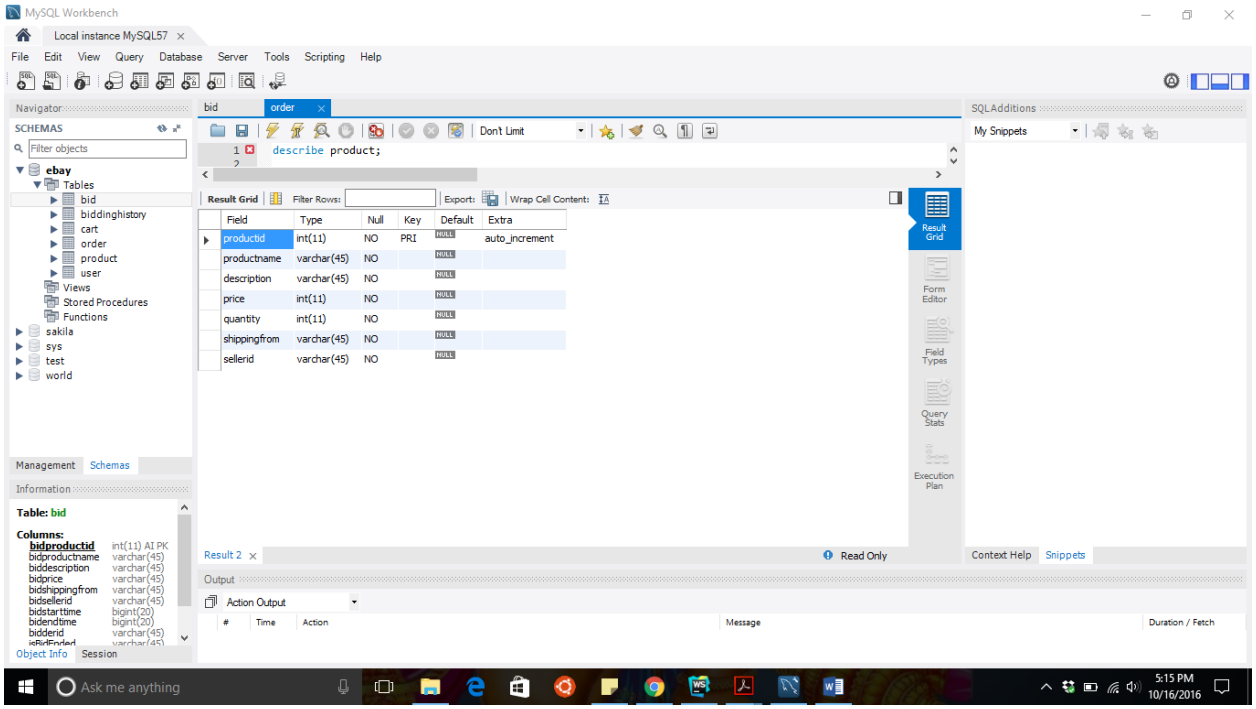

Payment Successful
localhost:3000/validate
ebay
Search the Ebay
My Ebay
Sign Out
Create Account
Your order is Successfully placed
Ask me anything
4:13 PM
10/16/2016

- **Database: Ebay**

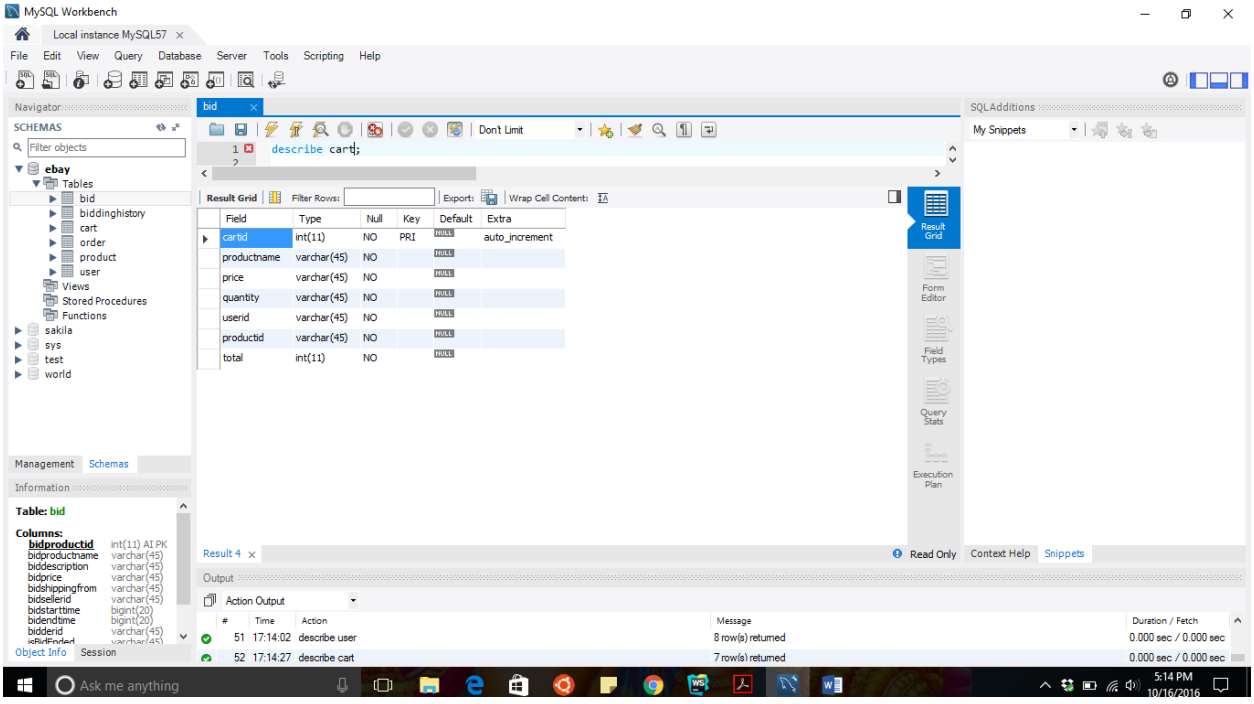
- **User Table**



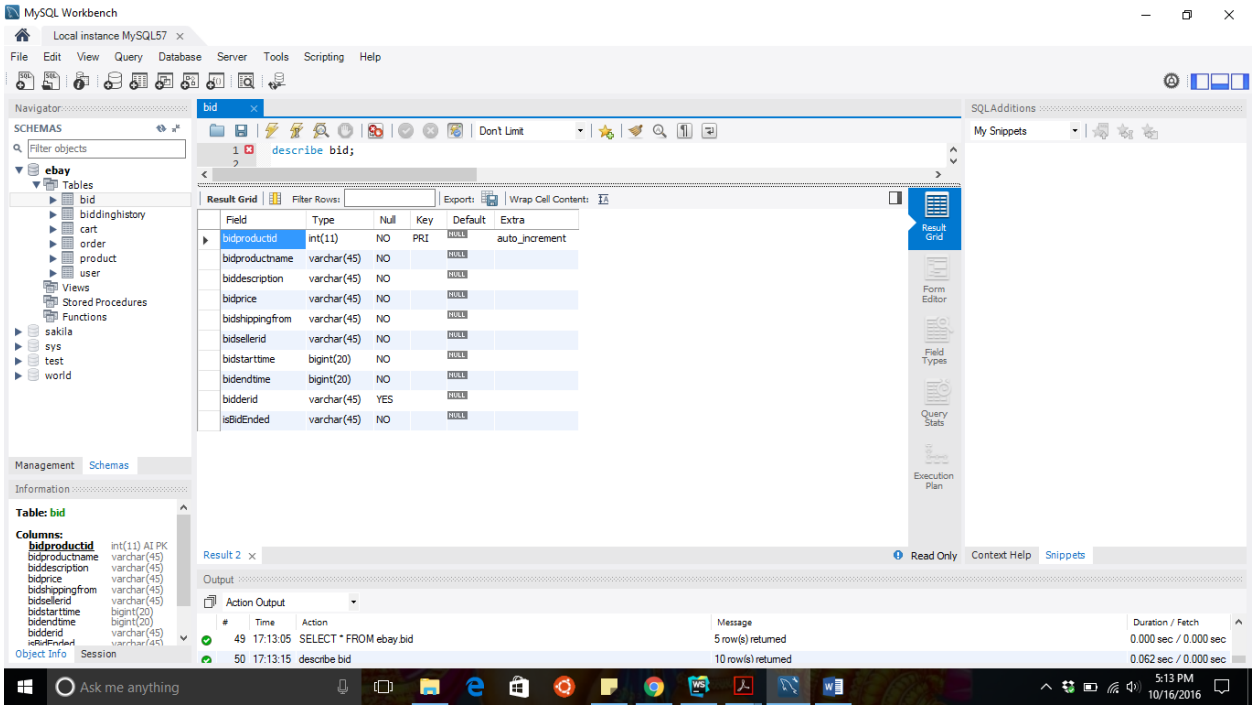
- **Product Table**



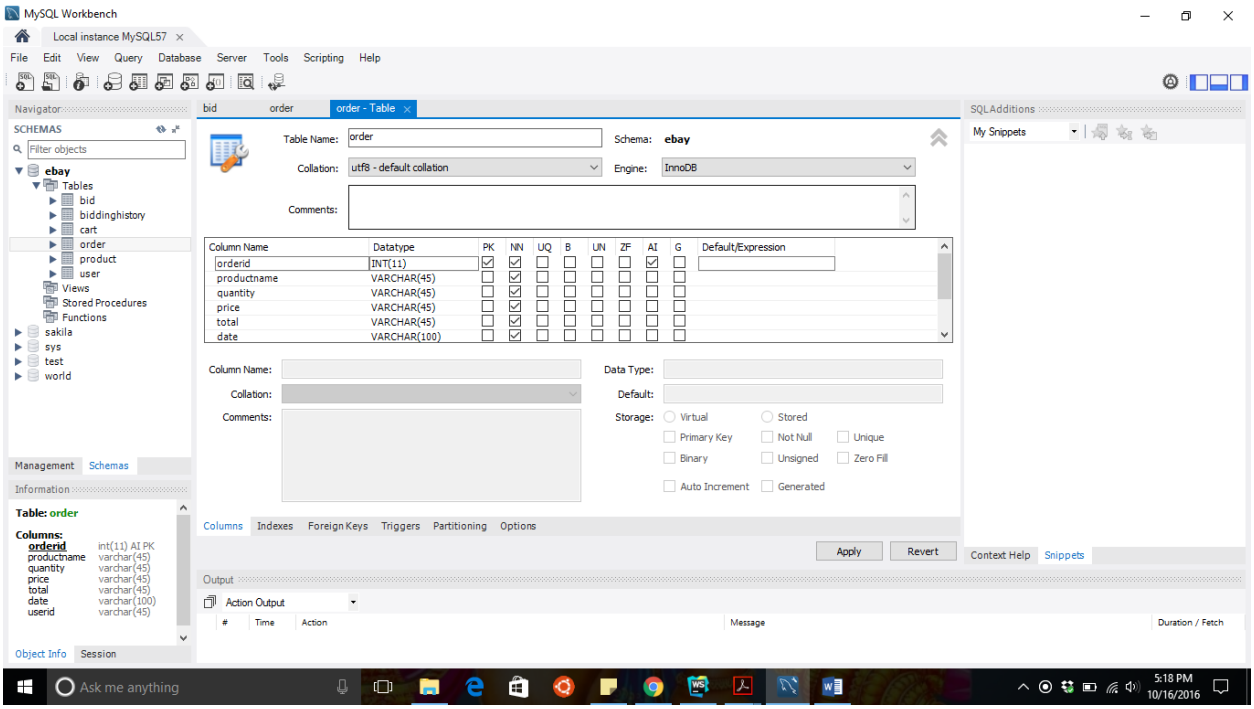
- **Cart Table**



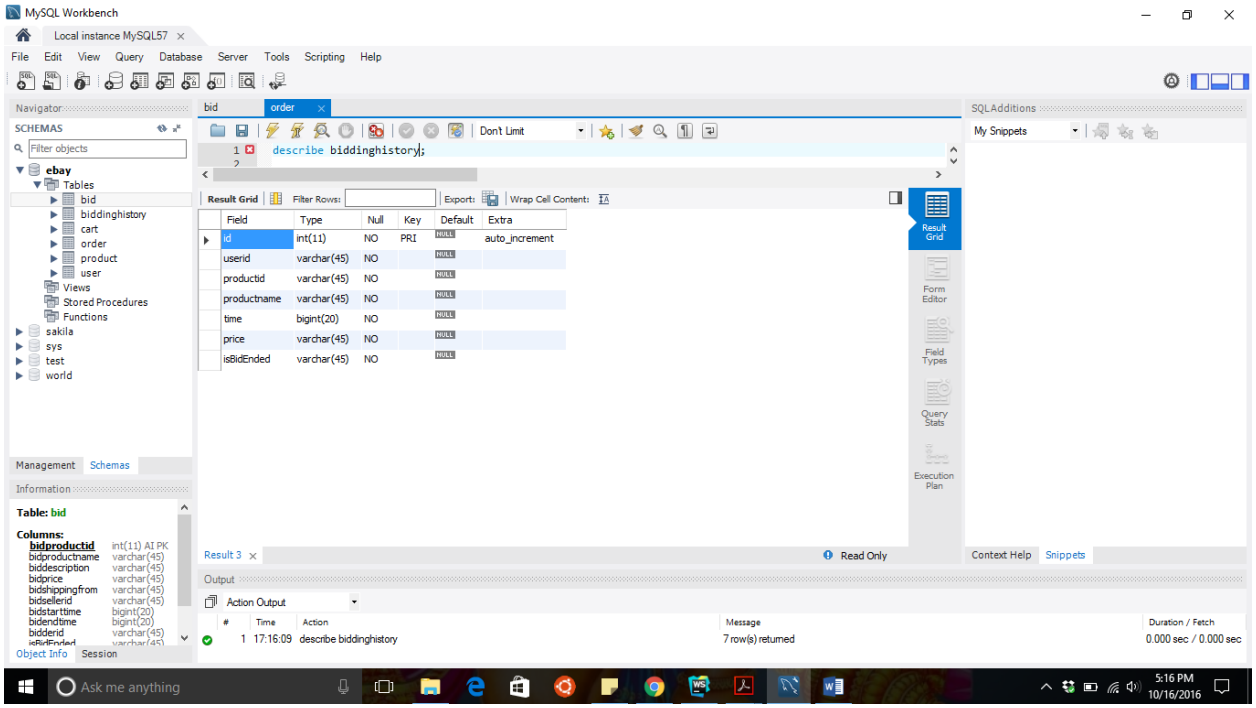
- **Bid Table**



- **Order Table**



- **BiddingHistory Table**

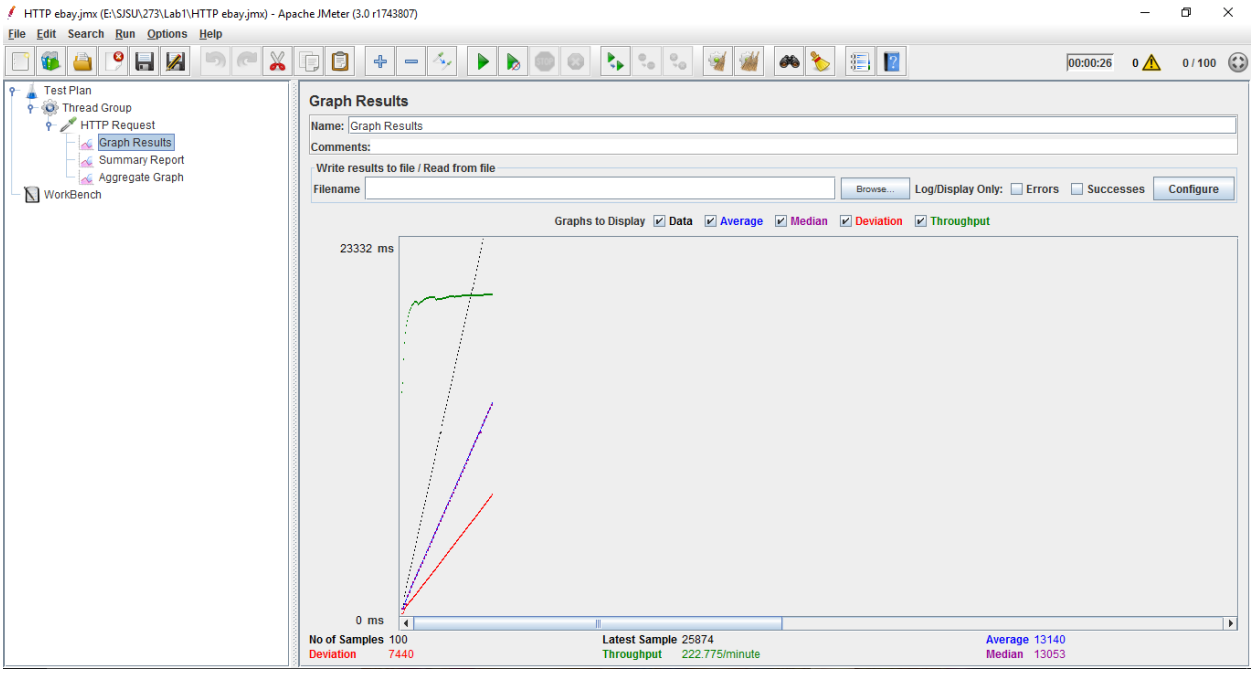


Testing

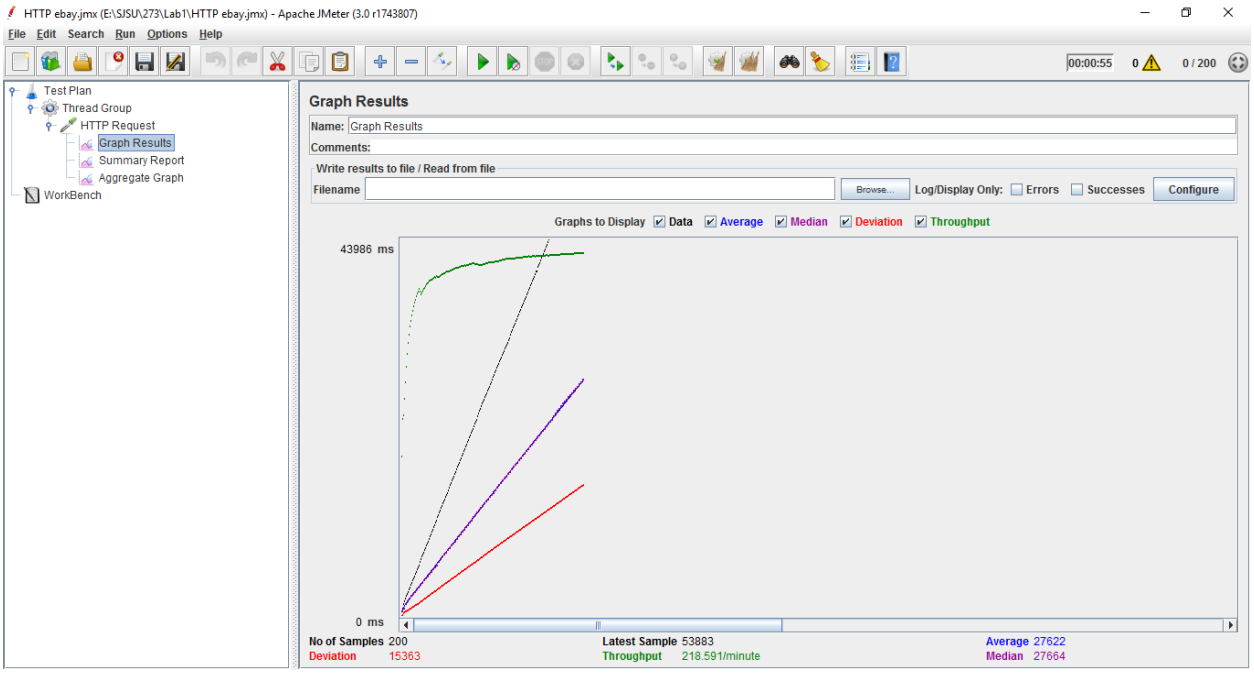
- **JMeter Screenshots**

1.Without Connection Pooling

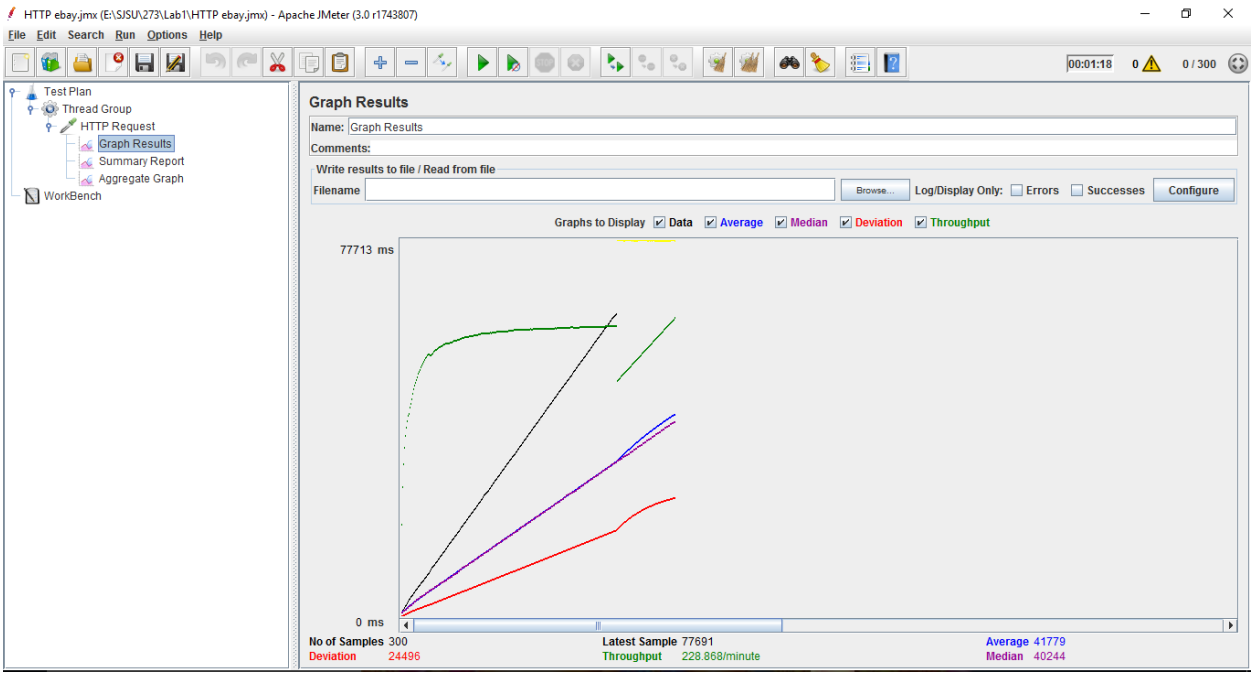
1)100 concurrent users



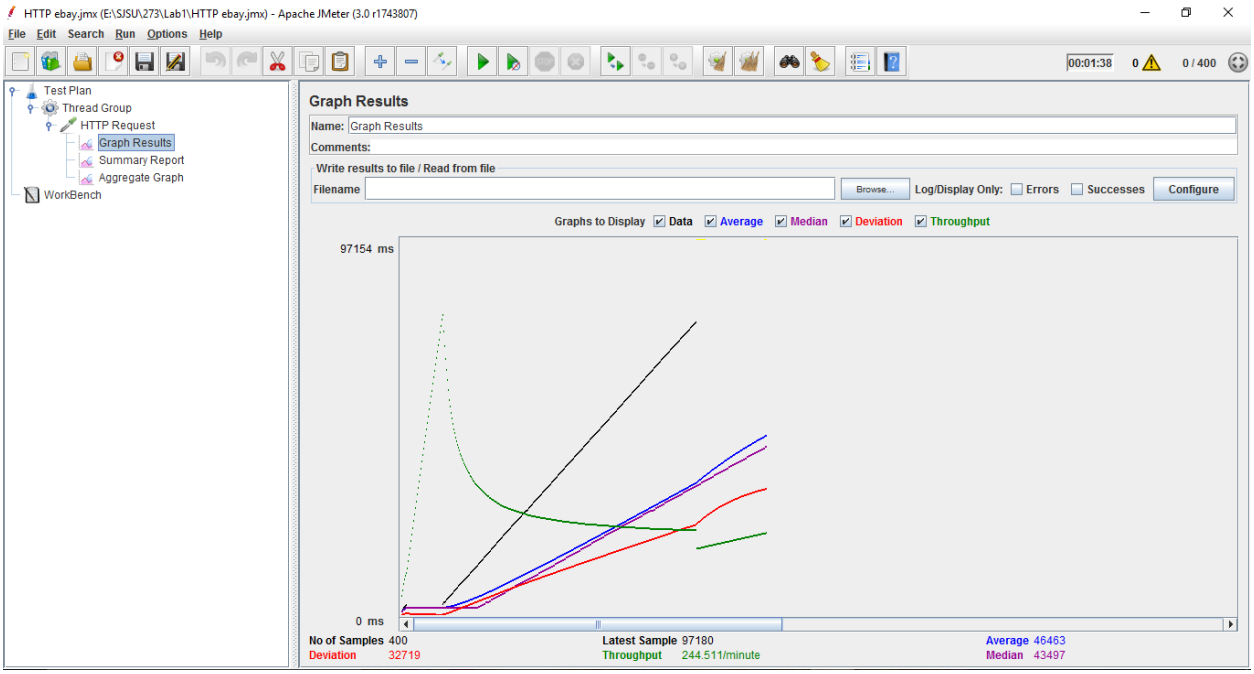
2)200 concurrent users



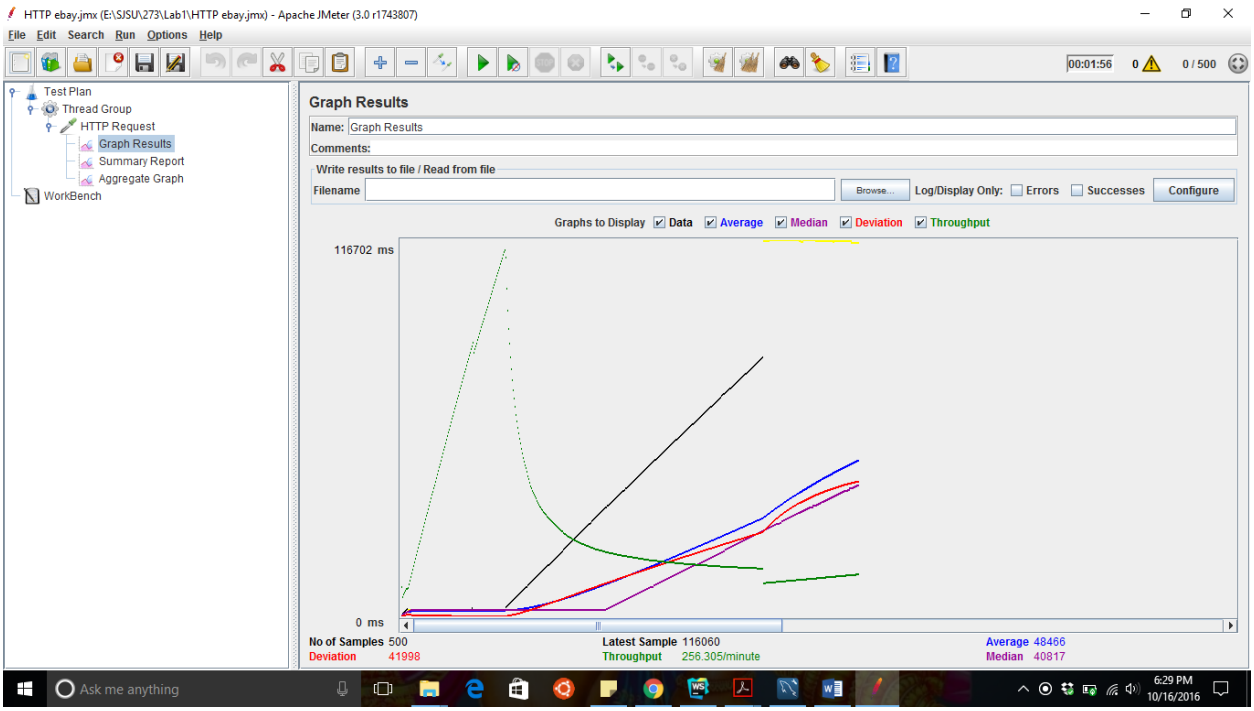
3)300 concurrent users



4)400 concurrent users

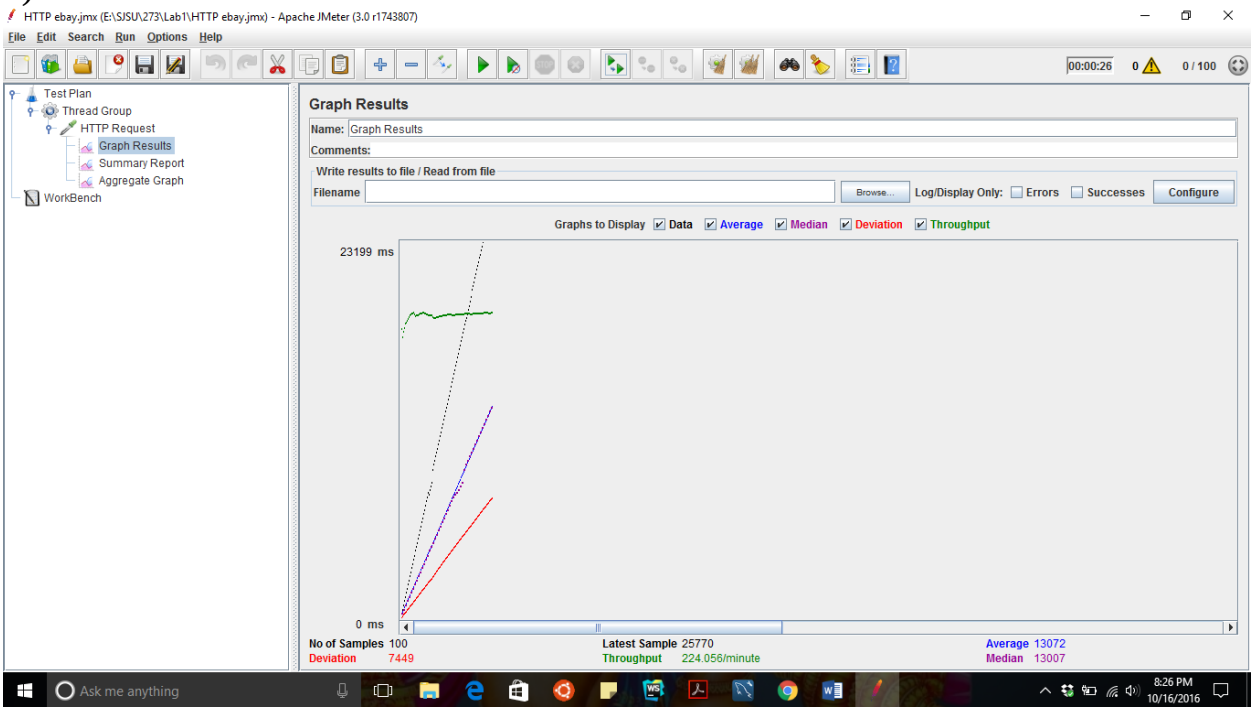


5)500 concurrent Users

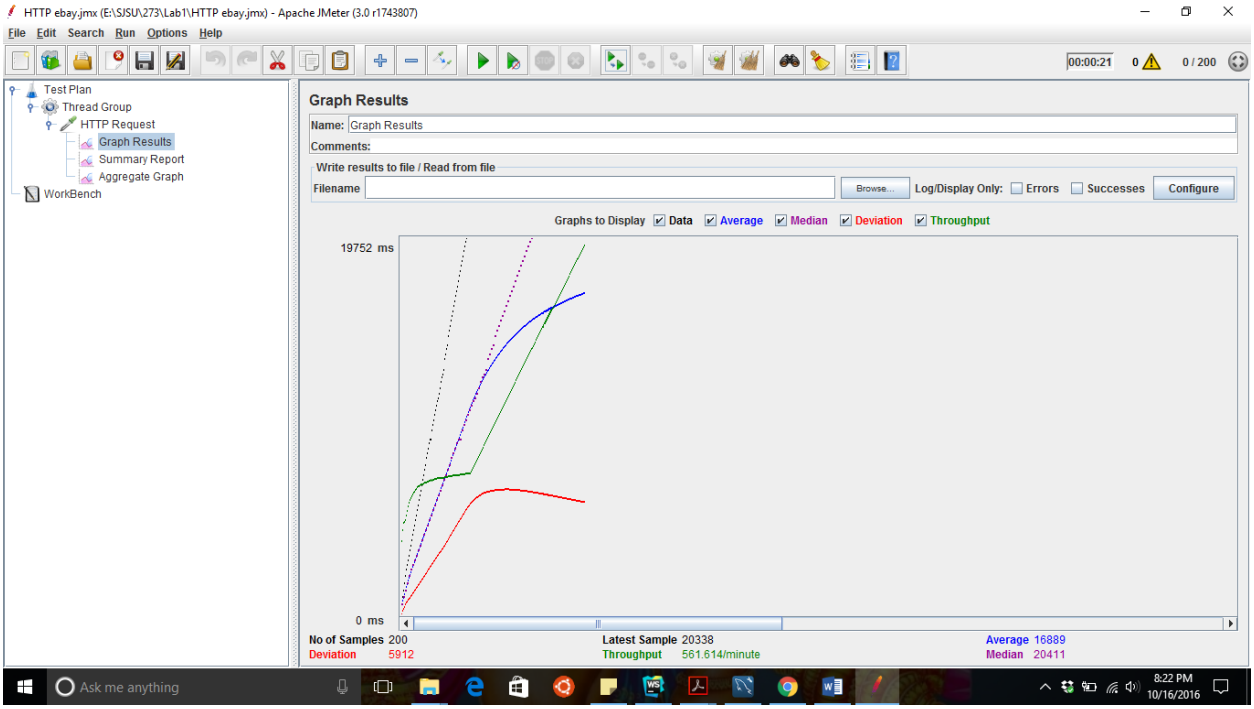


2.With Connection Pooling

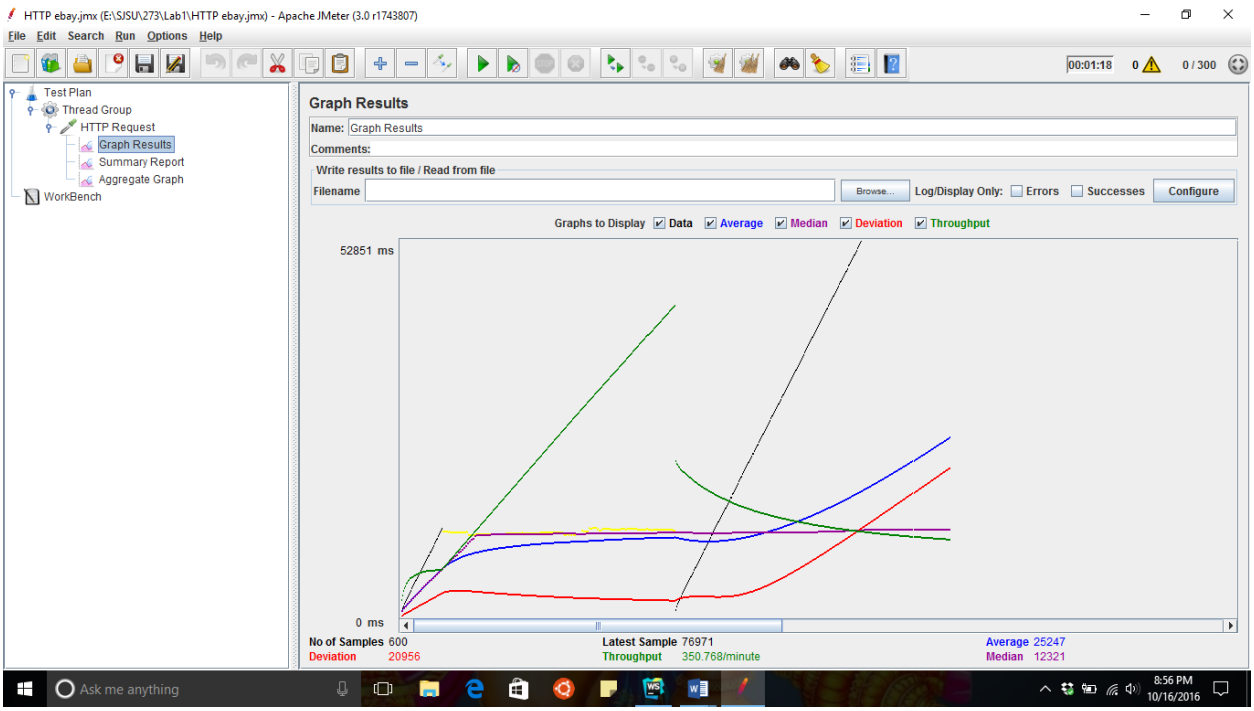
1)100 concurrent users



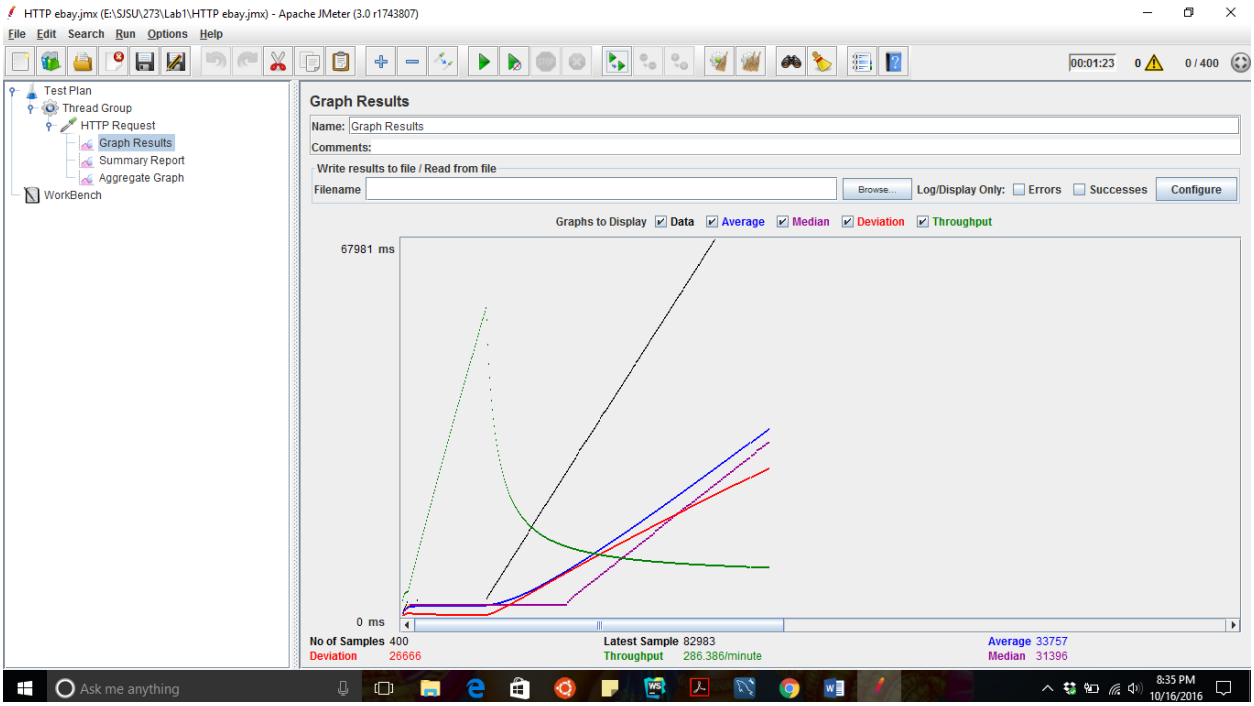
2)200 concurrent users



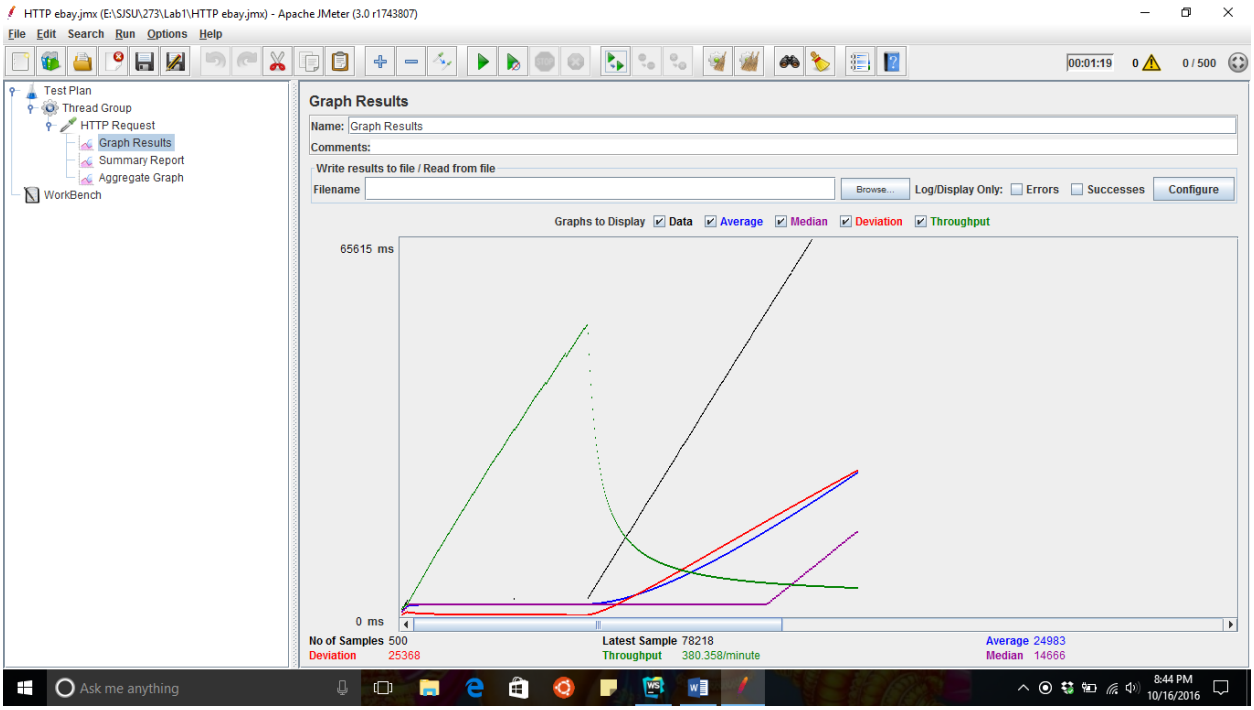
3)300 concurrent users



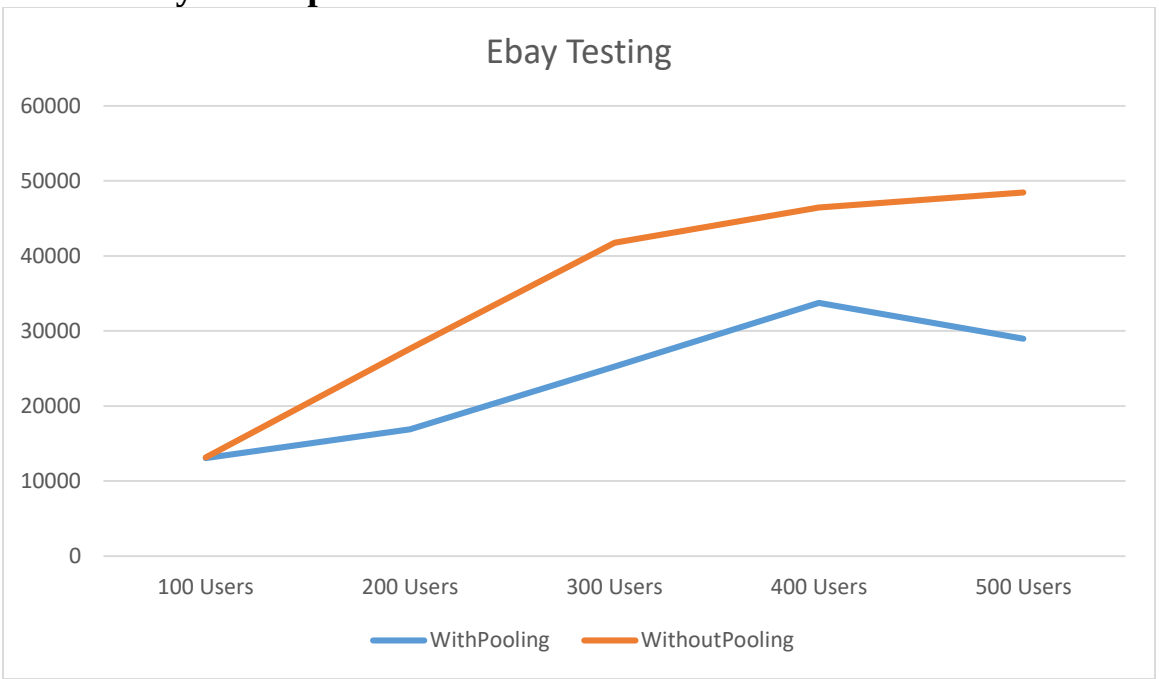
4)400 concurrent users



5)500 concurrent users

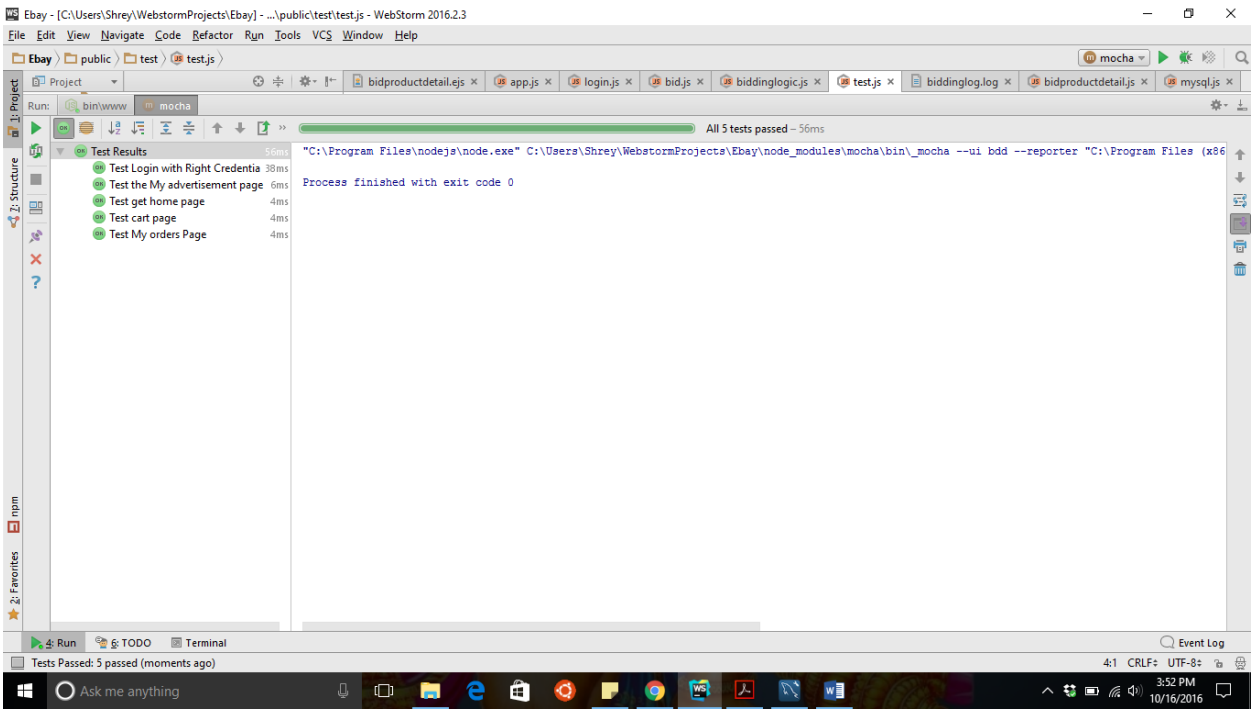


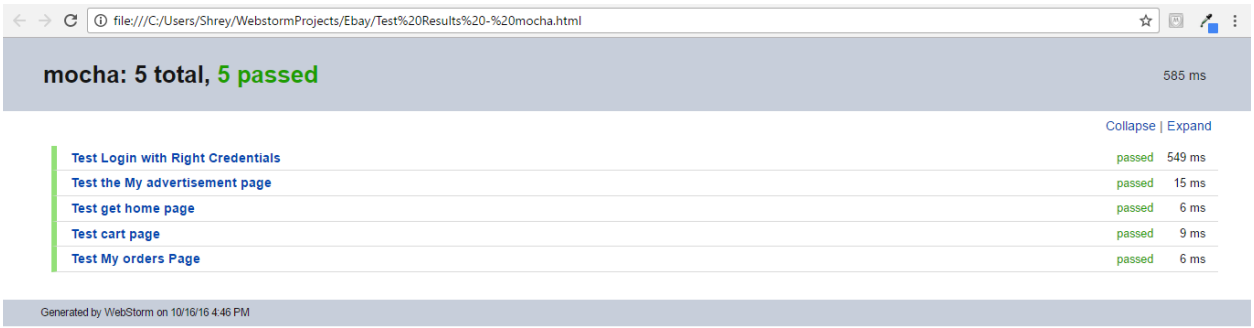
- **Analysis Graph**



- **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();
    });
});
```


Ebay - [C:\Users\Shrey\WebstormProjects\Ebay] - ...\public\test\test.js - WebStorm 2016.2.3
File Edit View Navigate Code Refactor Run Tools VCS Window Help
Test Results 56ms
Test Login with Right Credentia 38ms
Test the My advertisement page 6ms
Test get home page 4ms
Test cart page 4ms
Test My orders Page 4ms
All 5 tests passed – 56ms
"C:\Program Files\nodejs\node.exe" C:\Users\Shrey\WebstormProjects\Ebay\node_modules\mocha\bin_mocha --ui bdd --reporter "C:\Program Files (x86
Process finished with exit code 0
4: Run
6: TODO
Terminal
Event Log
Tests Passed: 5 passed (moments ago)
4:1 CRLF: UTF-8:
Ask me anything
3:52 PM
10/16/2016


file:///C:/Users/Shrey/WebstormProjects/Ebay/Test%20Results%20-%20mocha.html
mocha: 5 total, 5 passed
585 ms
Collapse | Expand
Test Login with Right Credentials passed 549 ms
Test the My advertisement page passed 15 ms
Test get home page passed 6 ms
Test cart page passed 9 ms
Test My orders Page passed 6 ms
Generated by WebStorm on 10/16/16 4:46 PM

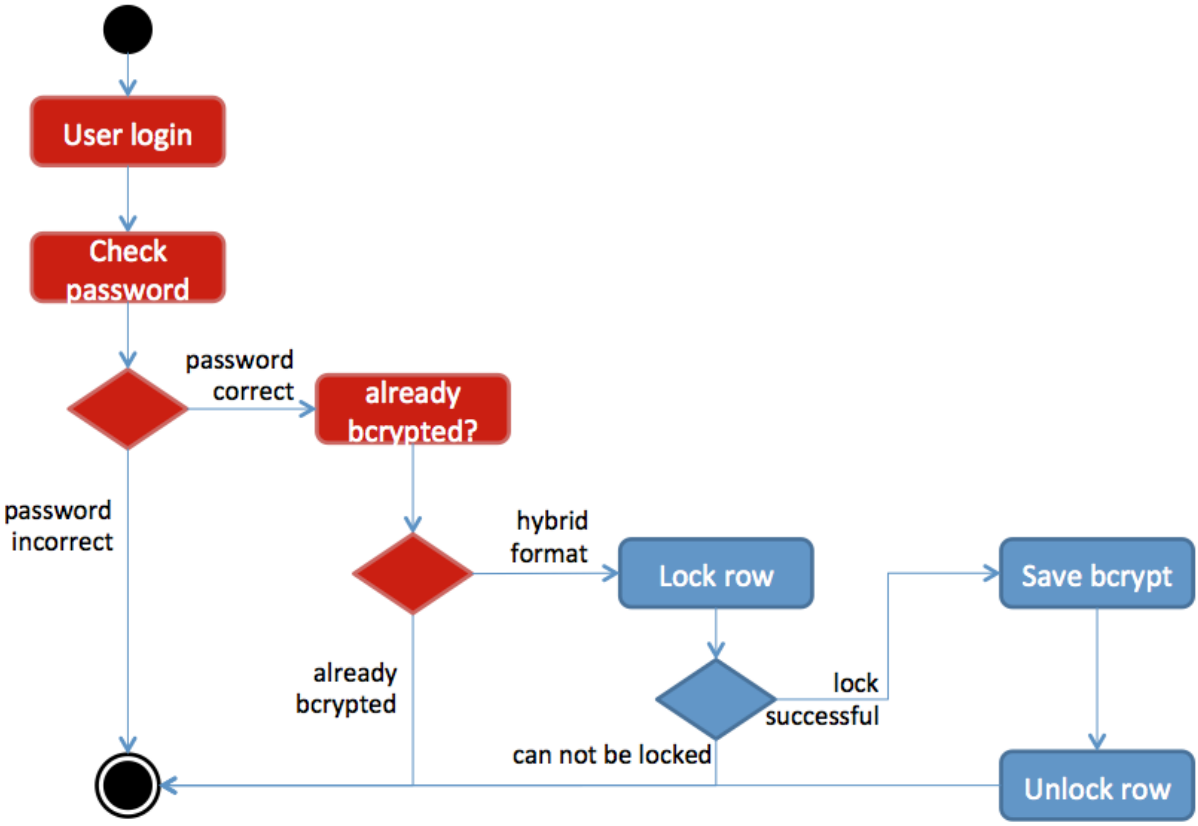
Part 3 : Answers

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)) {
:
:
```

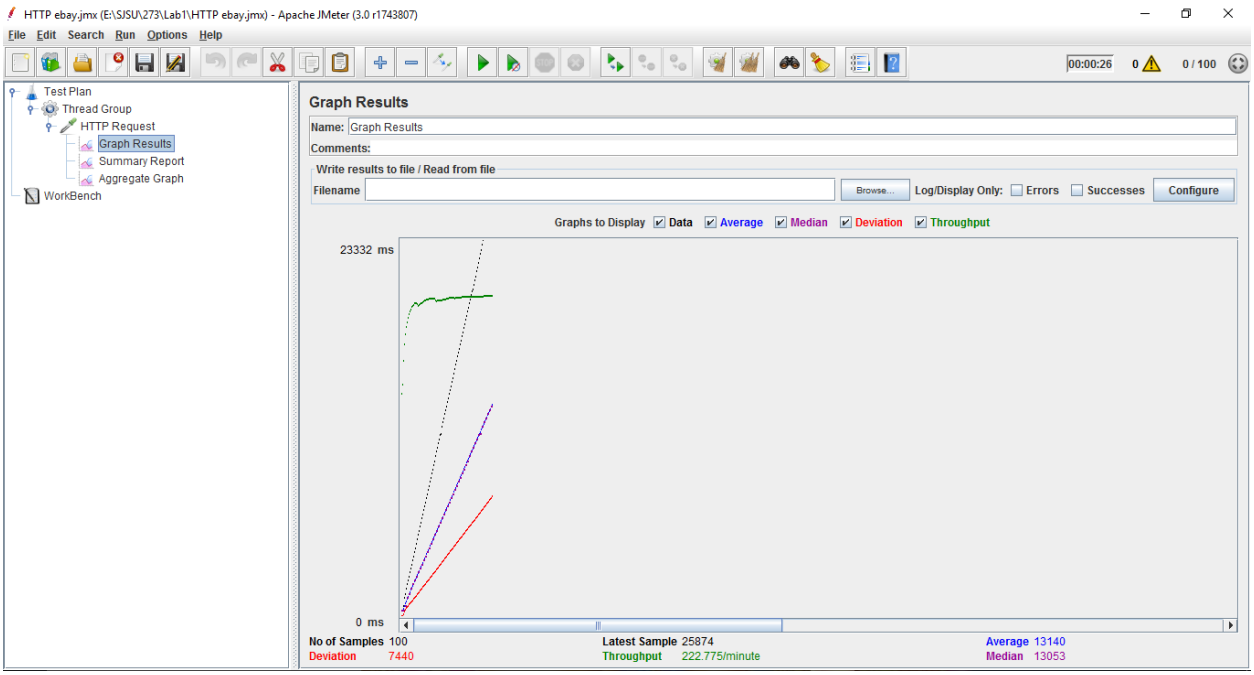
- **Working Of Bcrypt algorithm:-**



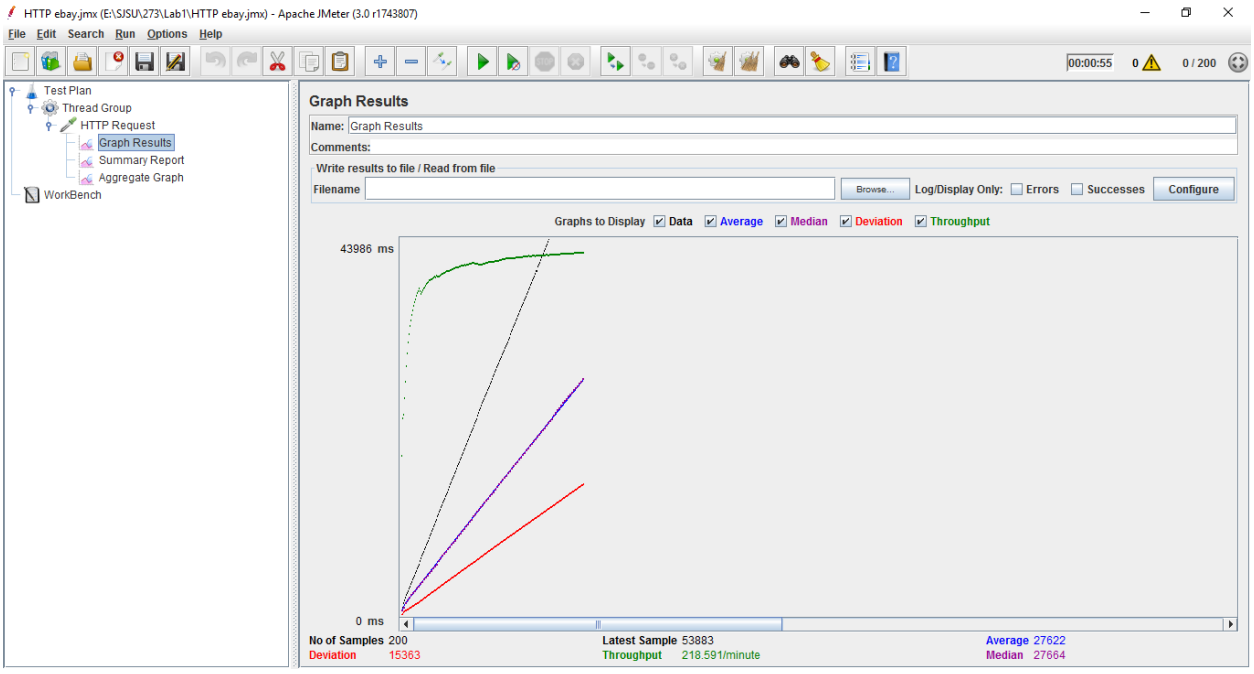
2. **Compare the Results of the graphs with and without connection pooling. Explain the results in detail. Describe the algorithm of connection pooling used in your application**

1.Without Connection Pooling

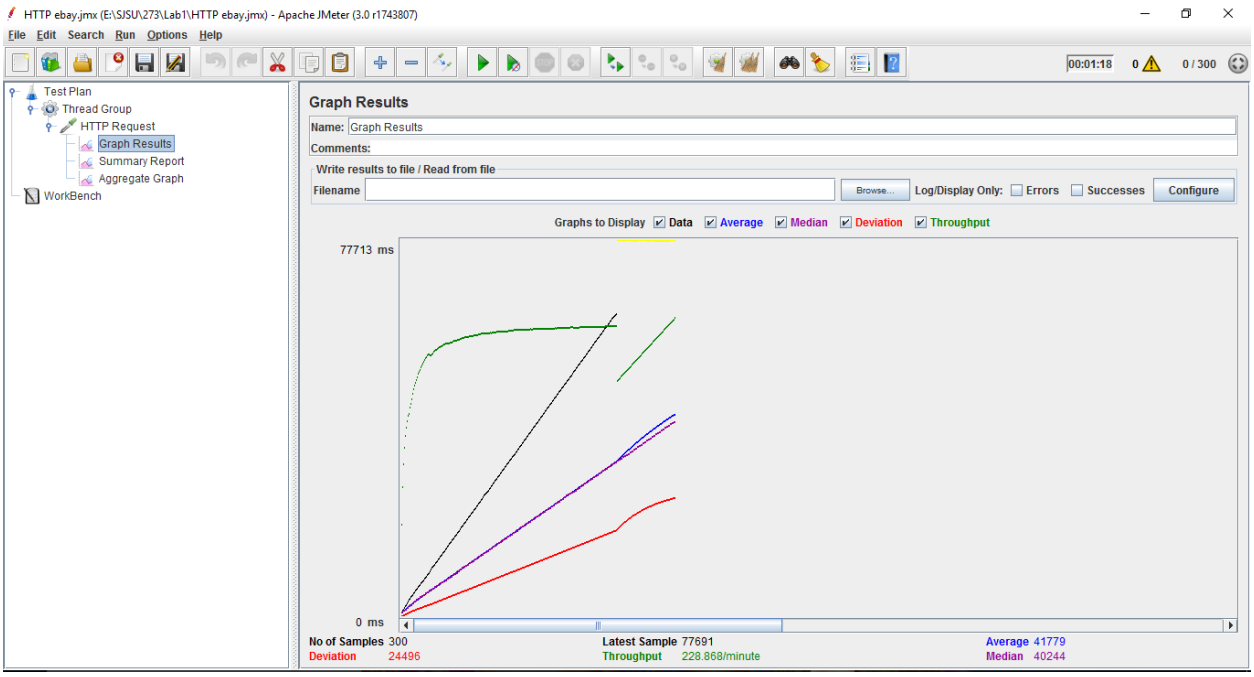
1)100 concurrent users



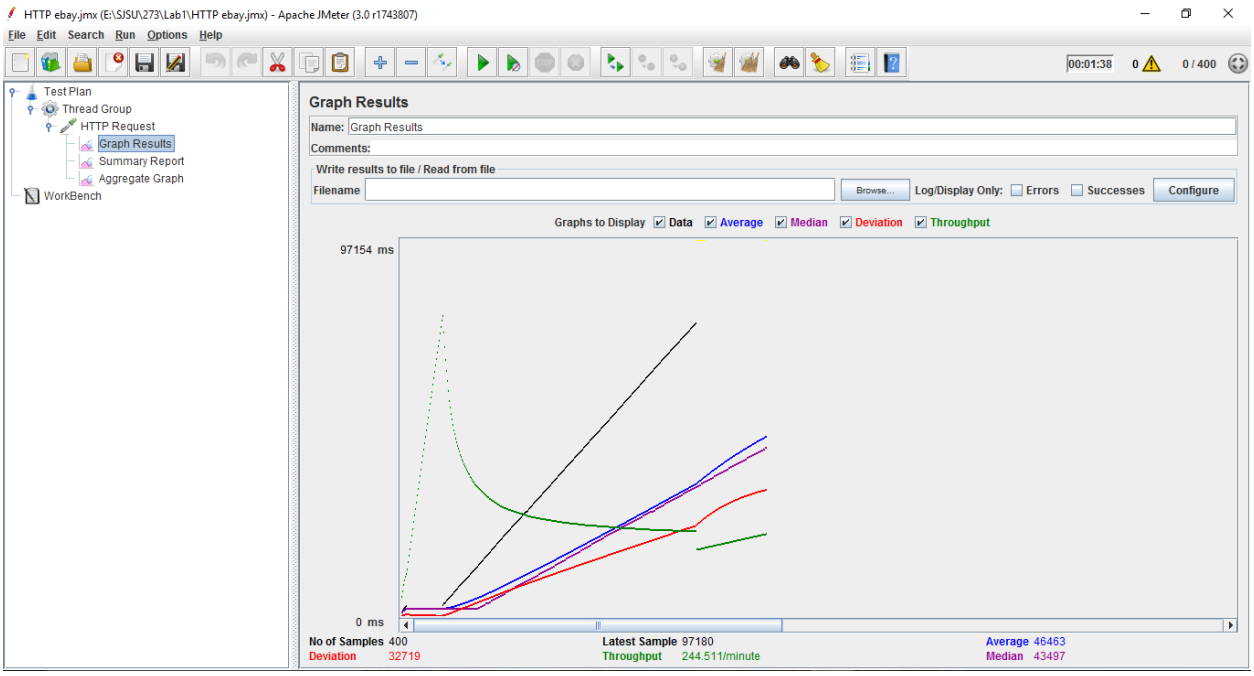
2)200 concurrent users



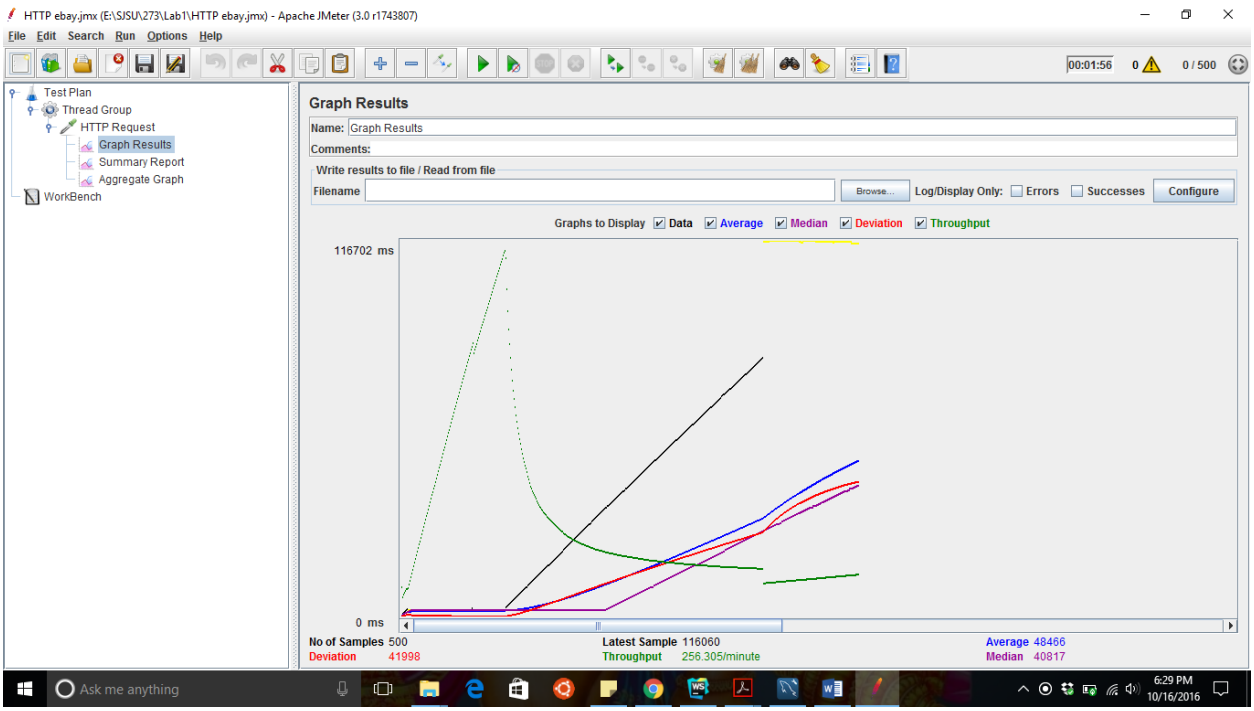
3)300 concurrent users



4)400 concurrent users

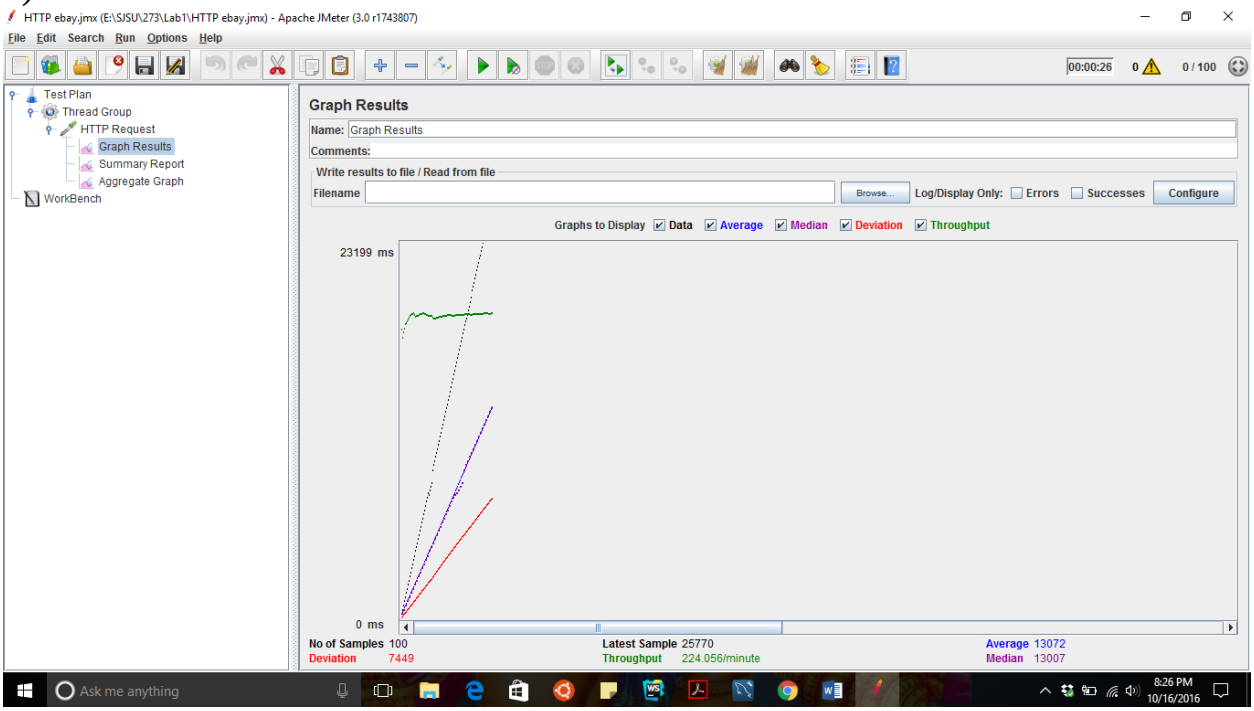


5)500 concurrent users

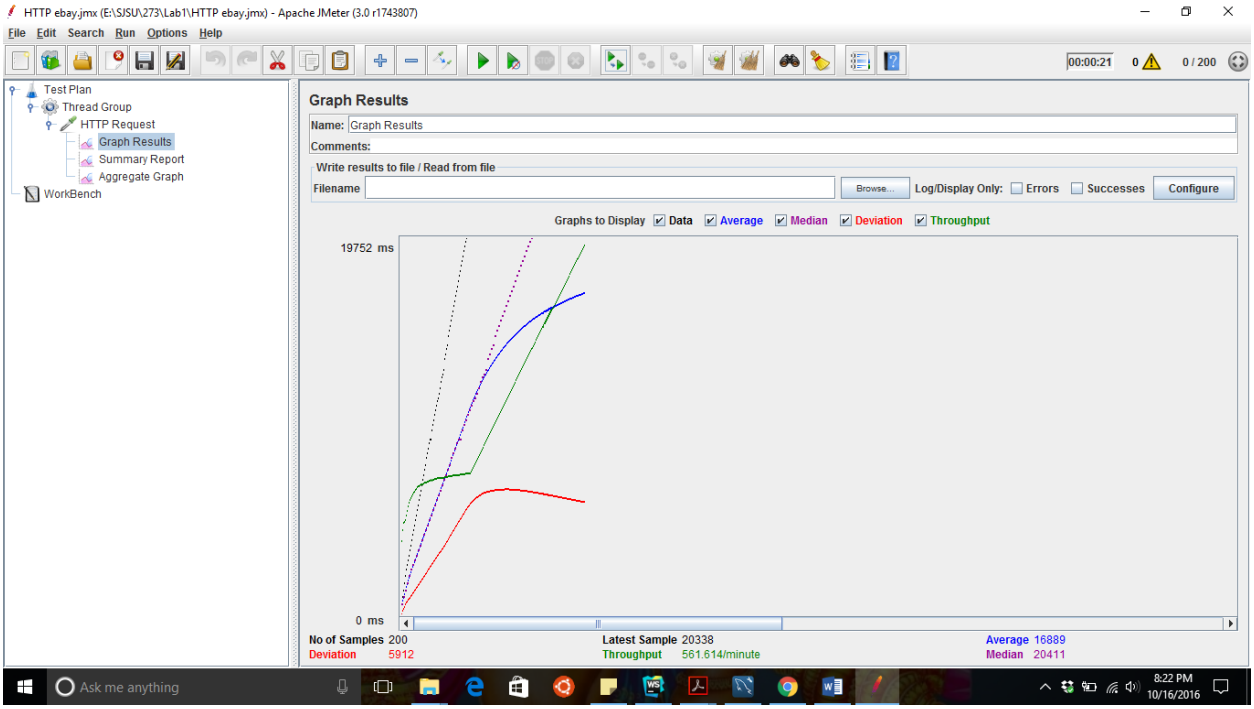


2.With Connection Pooling

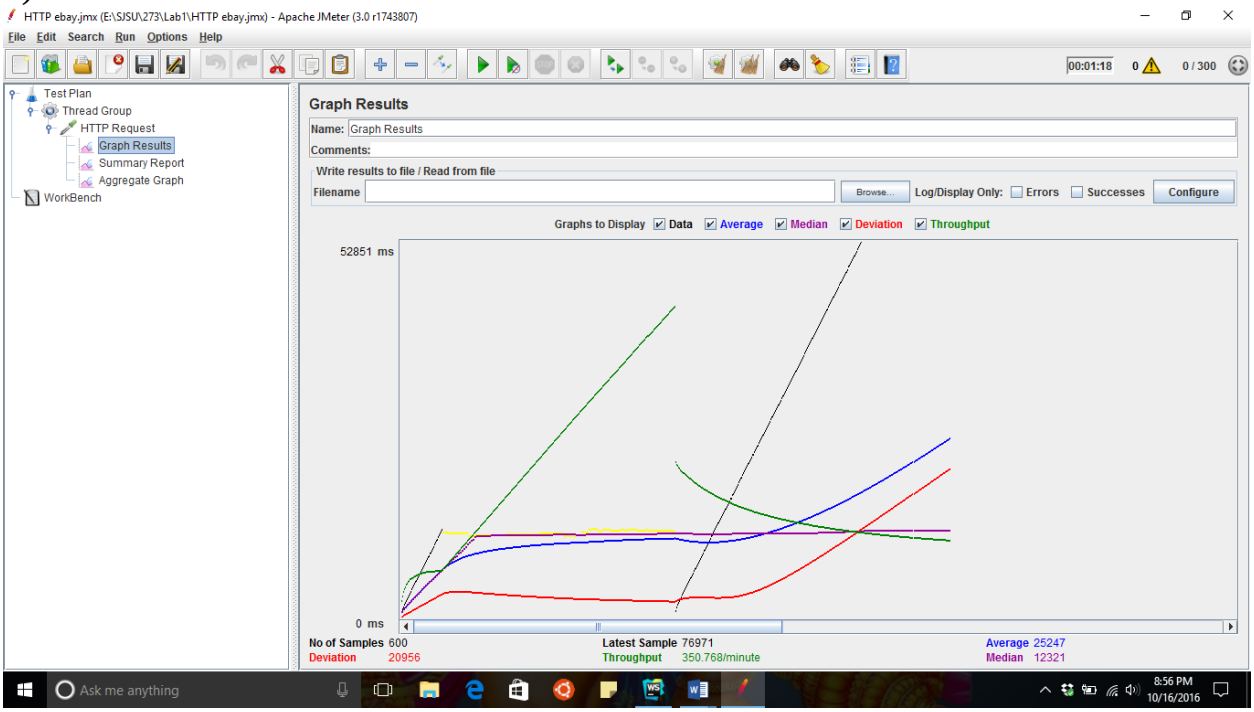
1)100 concurrent users



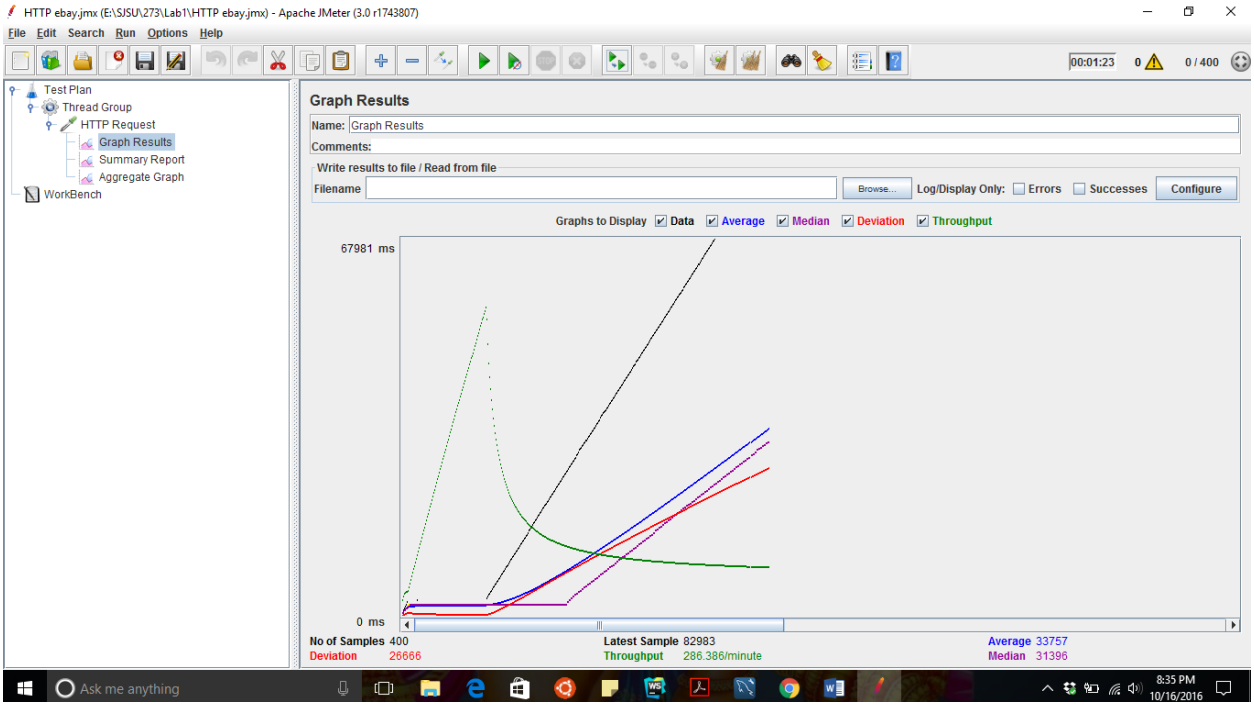
2)200 concurrent users



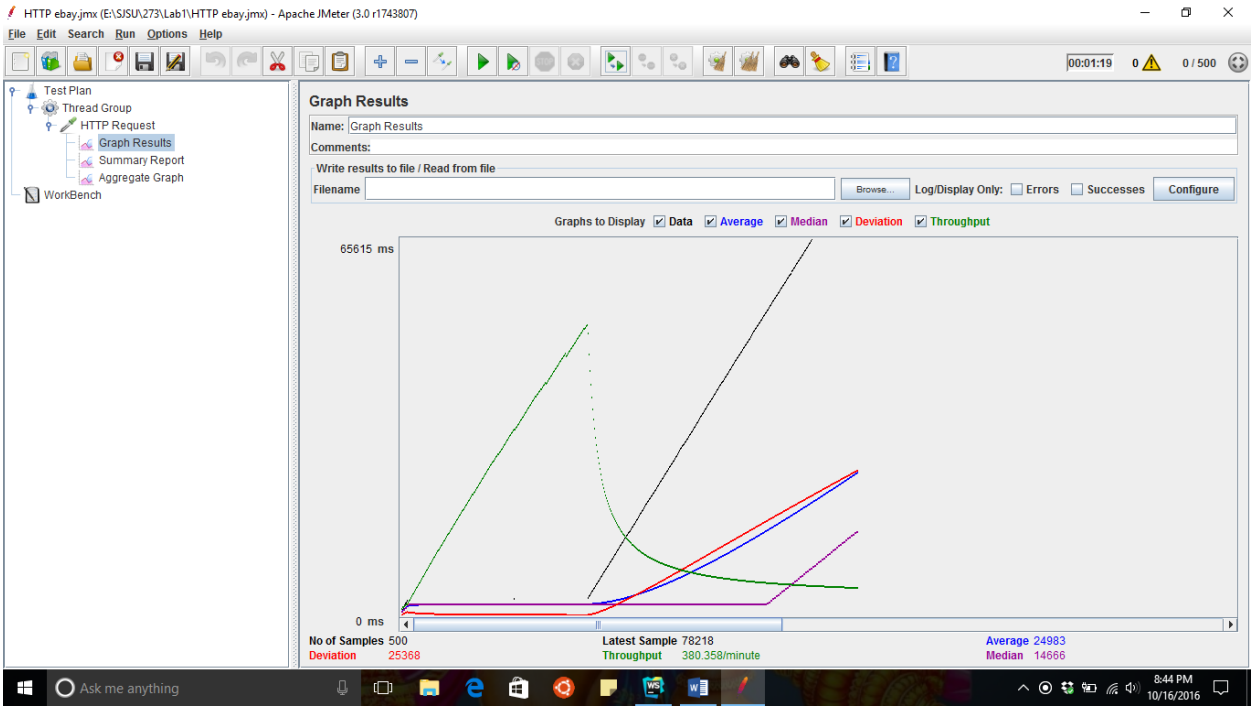
3)300 concurrent users



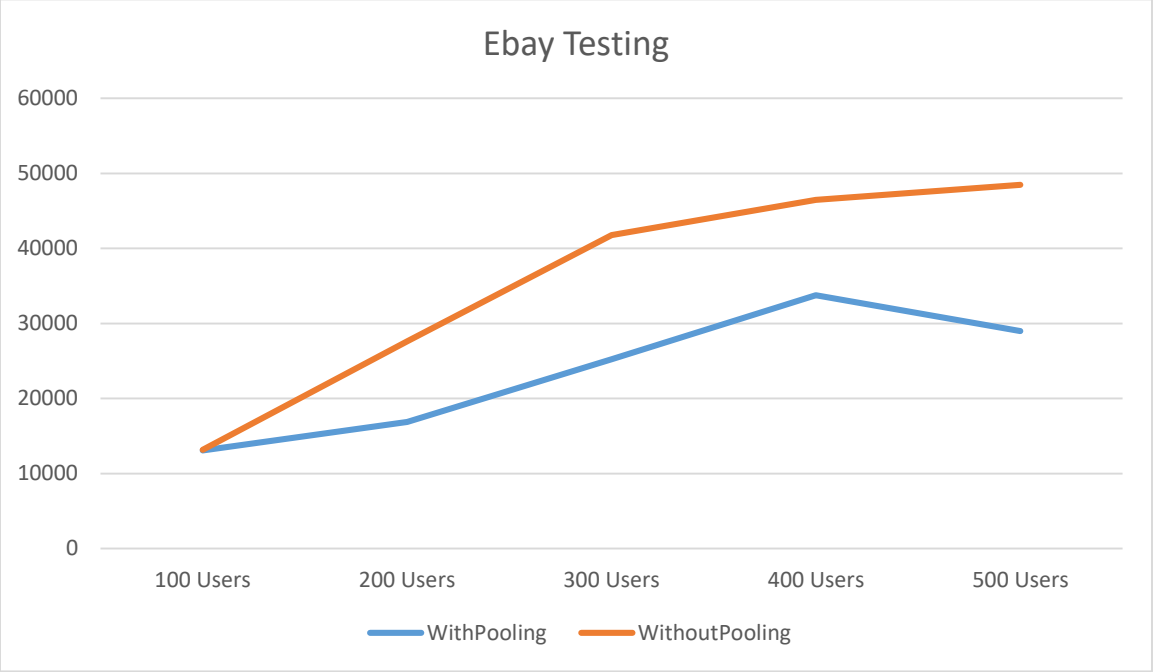
4)400 concurrent users



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++;


      } else {

         console.log('connection refused');
         return null;
      }
   }
}


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.