CMPE-273 ENTERPRISE DISTRIBUTED SYSTEMS (Section 02)

LAB#2 Using REST (Node.js), HTML 5 and Angular JS

NAME: DISHA DEVESHKUMAR SHETH

SJSU ID: 011498240

- **GOALS :**
  -To understand the functionality of RabbitMQ and mongoDB.
  -Implement lab assignment to understand the stateless nature using Node.js, HTML5 and AngularJS.

- **GITHUB LINK(private repository) :**
  https://github.com/disha-sheth/CMPE273-LAB2

- **PURPOSE OF THE SYSTEM :**

➢ **eBay (Simple Marketplace Application) :**

Here, we have developed a simple marketplace application wherein a user can buy as well as sell items. The items sold by a user can be viewed and/or purchased by other users as well. This application showcases the functionalities of MEAN stack wherein node.js, angular.js and express framework are used and backend is handled using MongoDB. RabbitMQ is used for message queuing.

DESIGN:

Entire web-application using n-tier architecture.

Front-end: HTML5, CSS, Bootstrap and AngularJS.

Back-end: MongoDB, NodeJS, Express Framework.

Messaging Service: RabbitMQ.

User Authentication Module: PassportJS

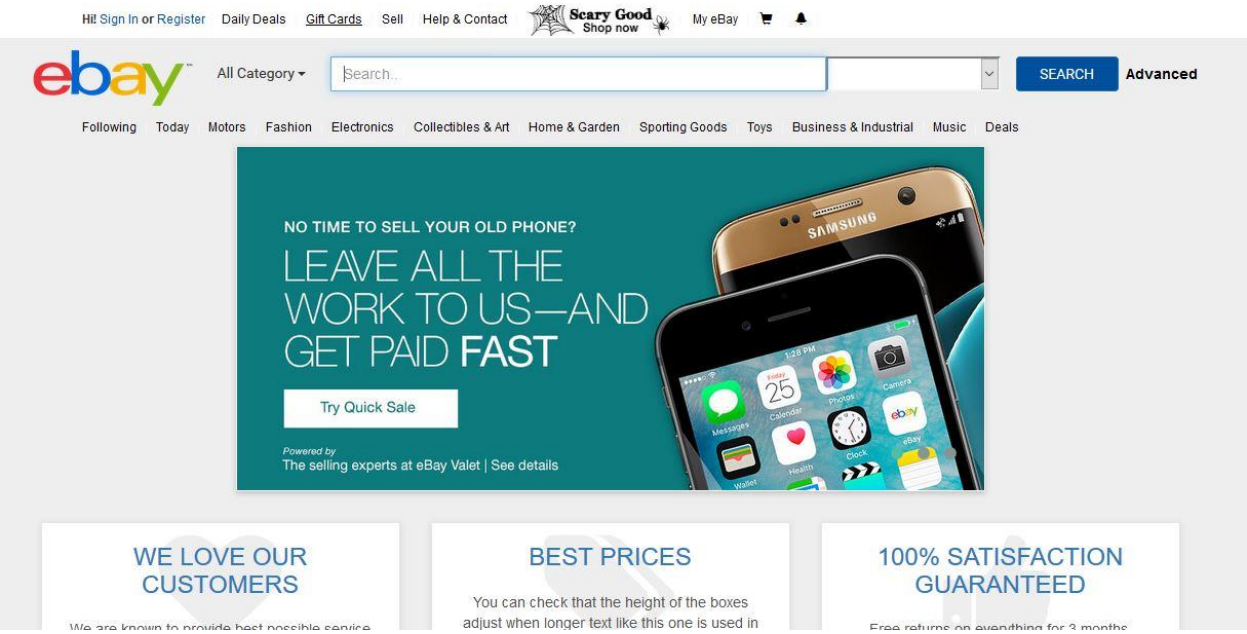
- **PART 1: eBay (Simple Marketplace Application)**

  eBay design has been implemented using Express framework, HTML5 and bootstrap features. AngularJS has been used as middleware and nodeJS handles the server side functionality. For backend database functionality, MongoDB has been used and RabbitMQ implemented for queuing purposes.

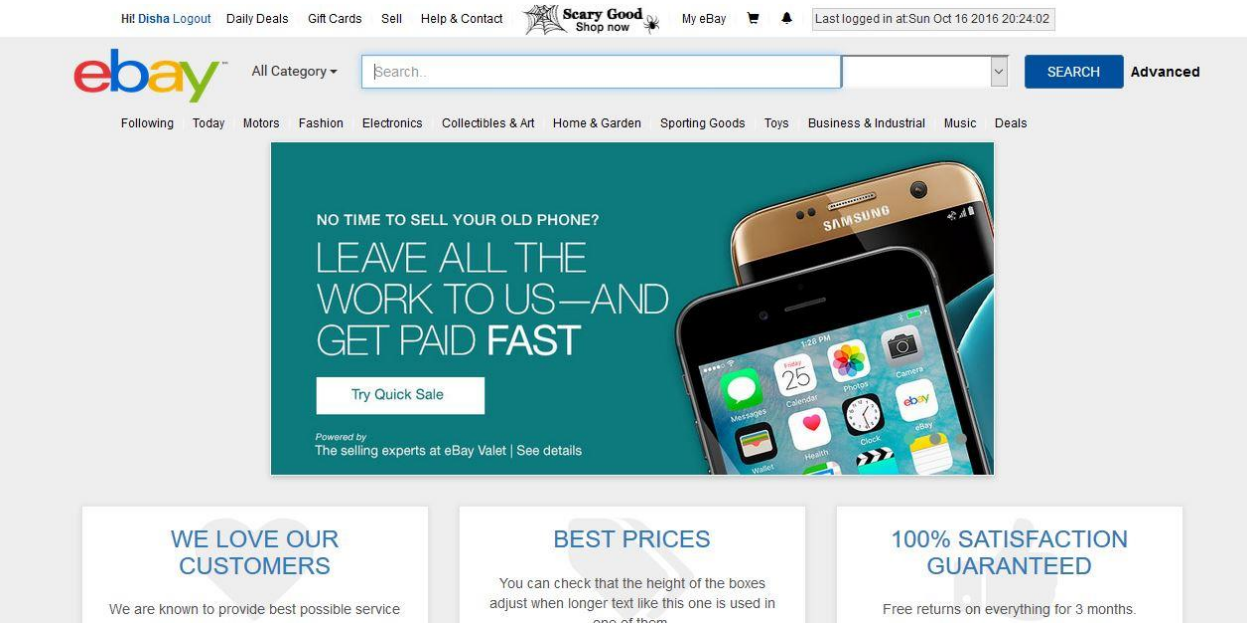
- HOMEPAGE :

  The homepage imitating eBay's homepage :

  Before logging in :



After logging in:



- SIGN/SIGNUP FOR A USER :

Signup for new users :

On registering, the details of the user are stored in database wherein the password of the user is encrypted for security reasons. Also, ebay_handle for the user is created which is unique for every user.

ebay

Sign In | Register

Email

Reenter email

Password

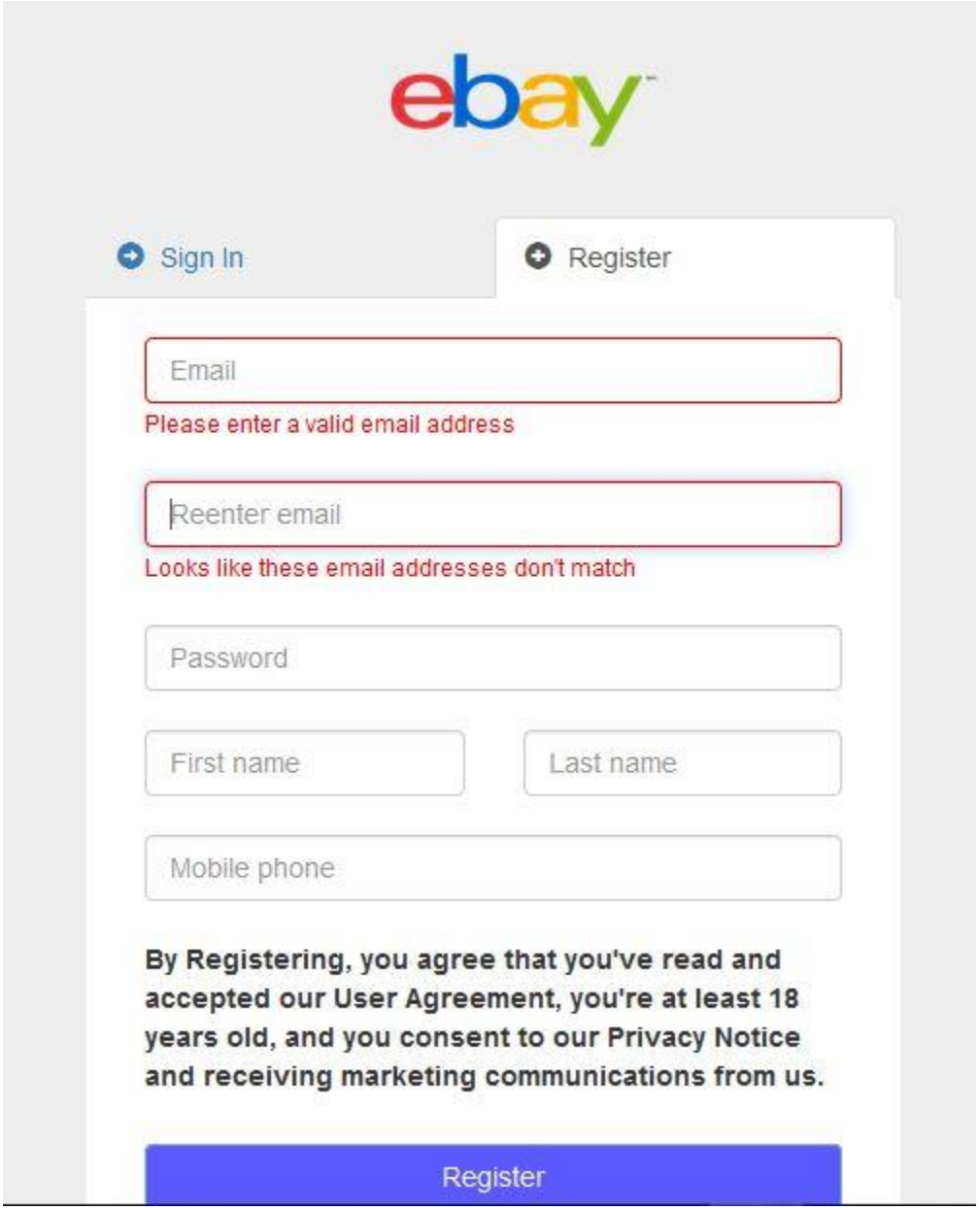
First name | Last name

Mobile phone

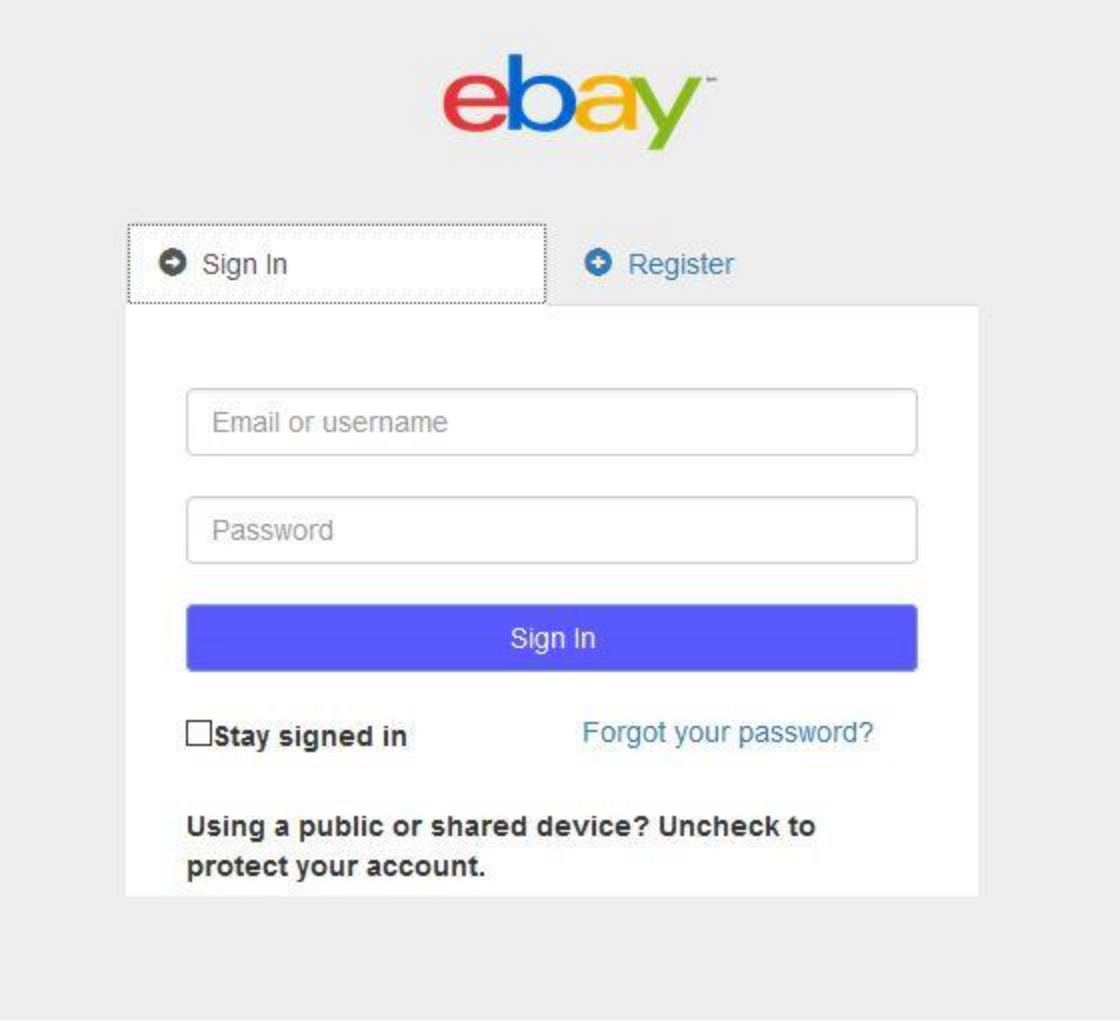
By Registering, you agree that you've read and accepted our User Agreement, you're at least 18 years old, and you consent to our Privacy Notice and receiving marketing communications from us.

Register

-Required validations on email, password, first name, last name and mobile phone have been put to ensure that the user registers correctly using appropriate values.

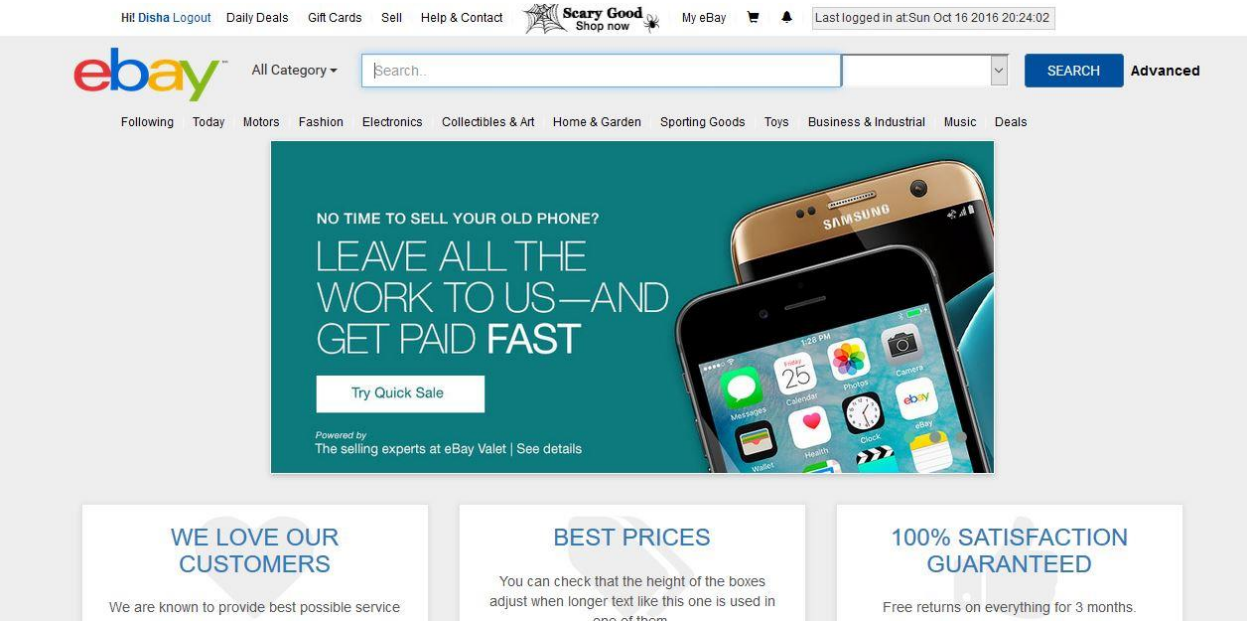


Once registered, the user can login with registered email ID and password :



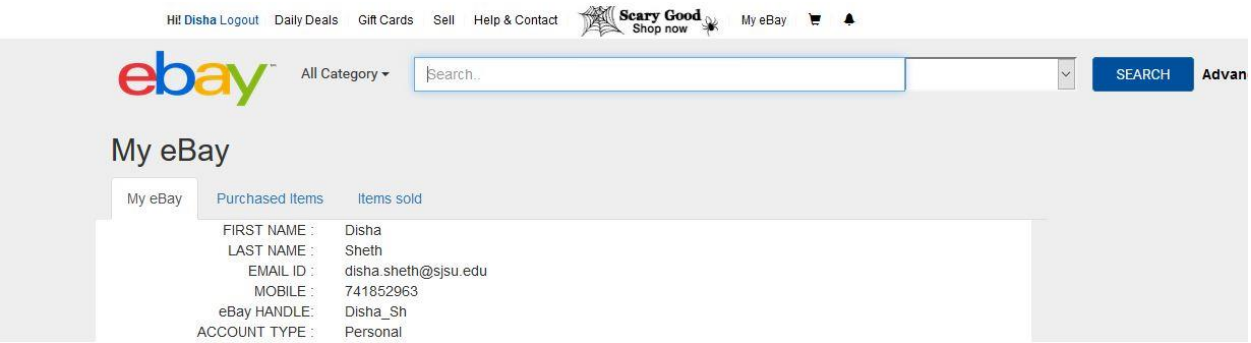
On logging in, the user can access various functionalities like :

- VIEWING LAST LOGGED IN:

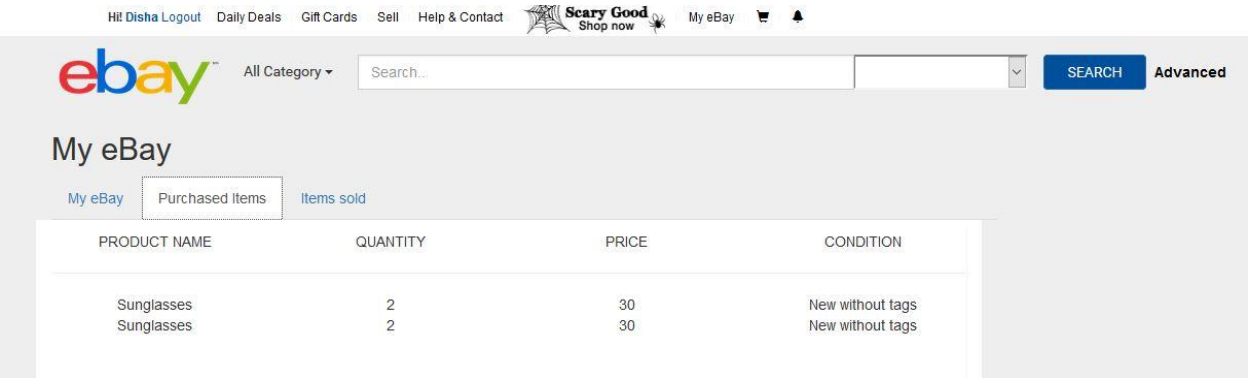


- VIEW USER PROFILE :

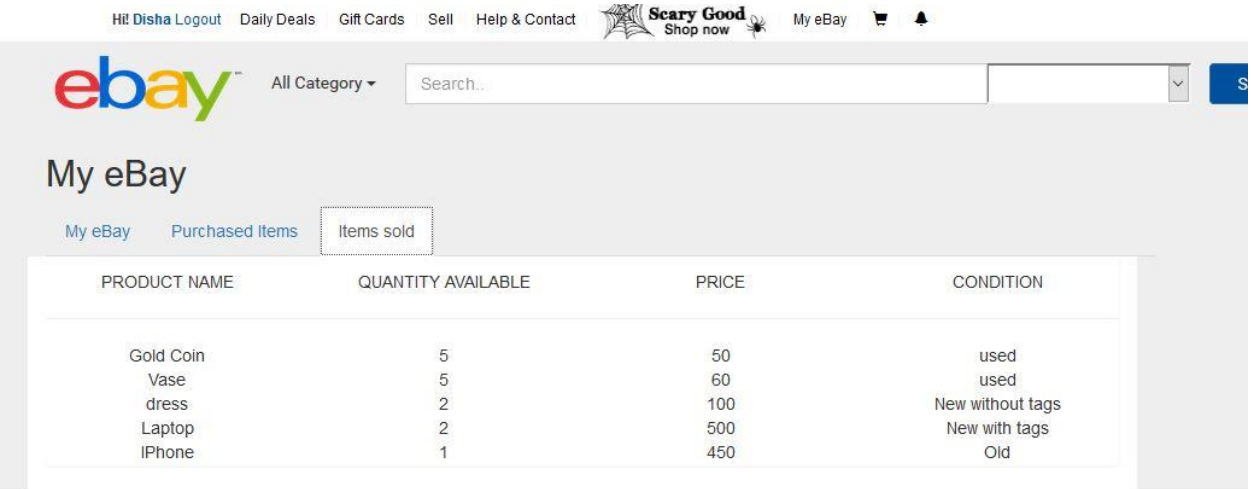
On logging in, the user can view his own profile :



View items purchased by the user till date :

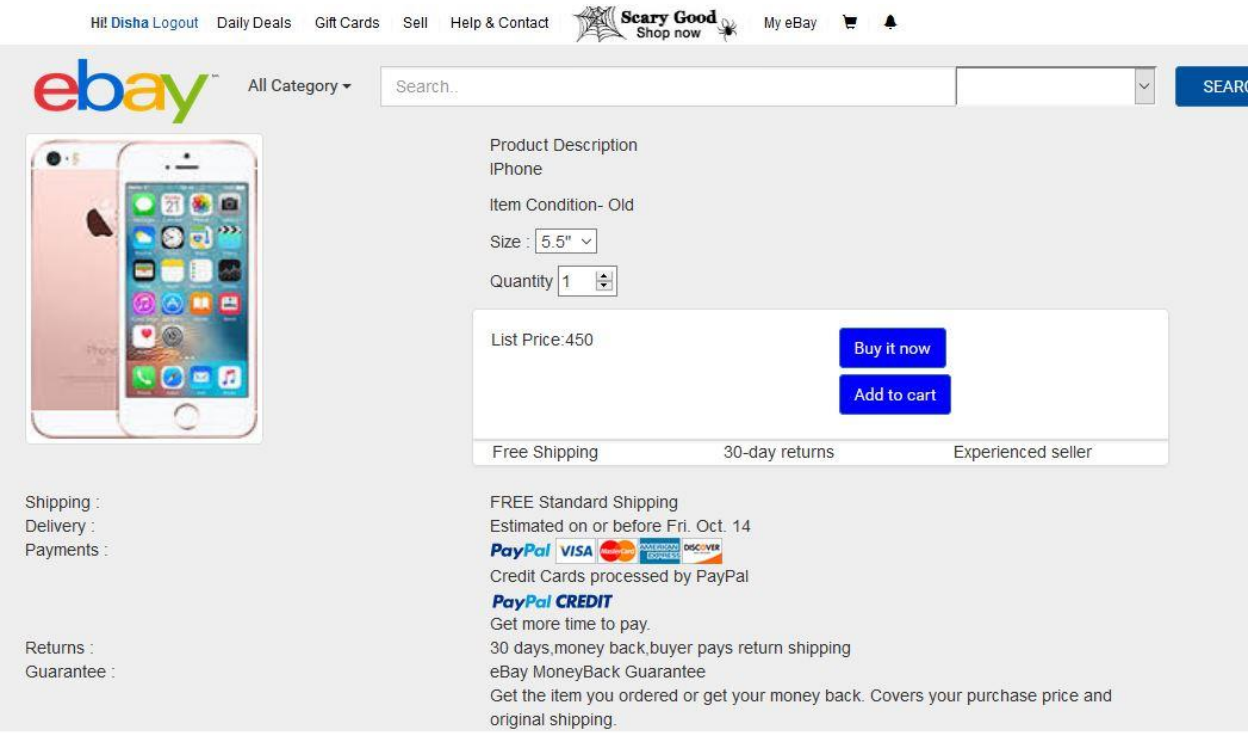


Items sold by the user :



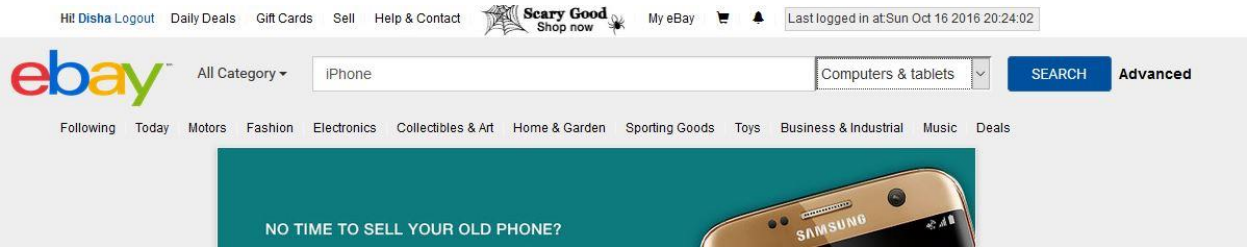
- VIEW PRODUCT DETAILS :

User can view more product details searched from search bar or from searching category/sub-category-wise :



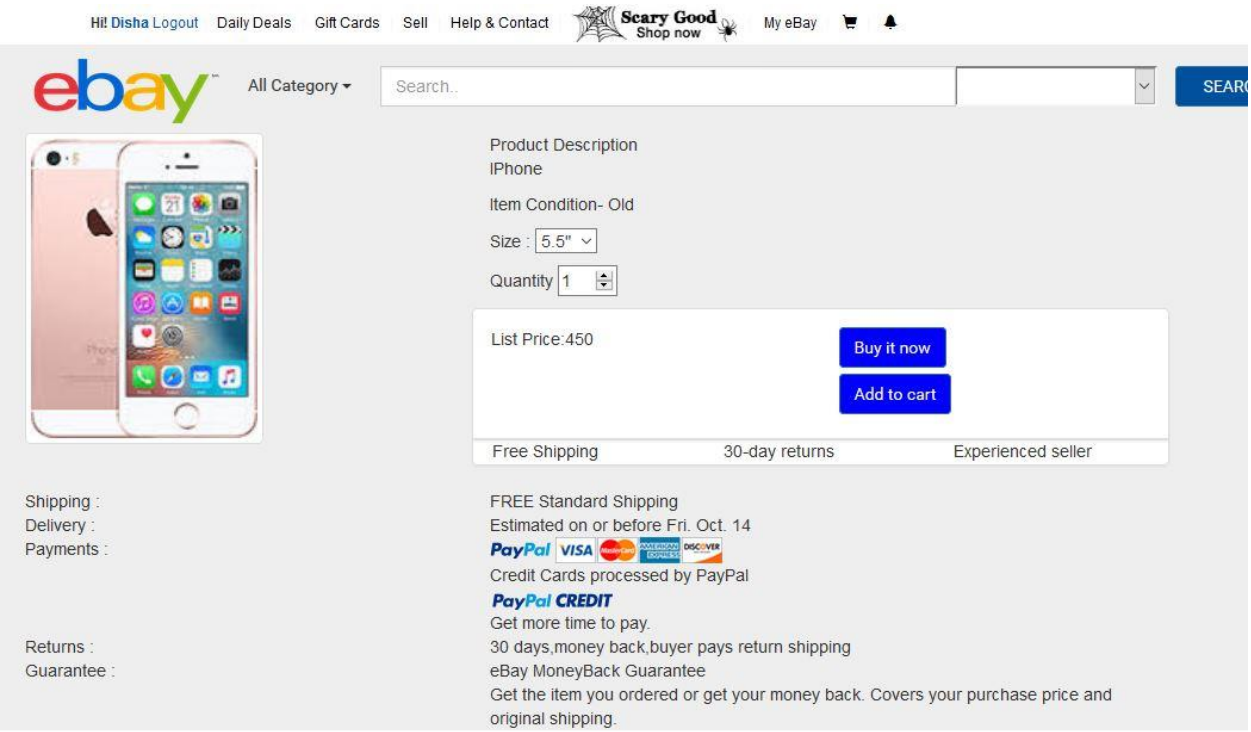
- SEARCH BY PRODUCT :

Searchbar on almost all the pages of the marketplace helps easy searching of products:



A user can view products uploaded for selling by other users as well.Results of above search will be :

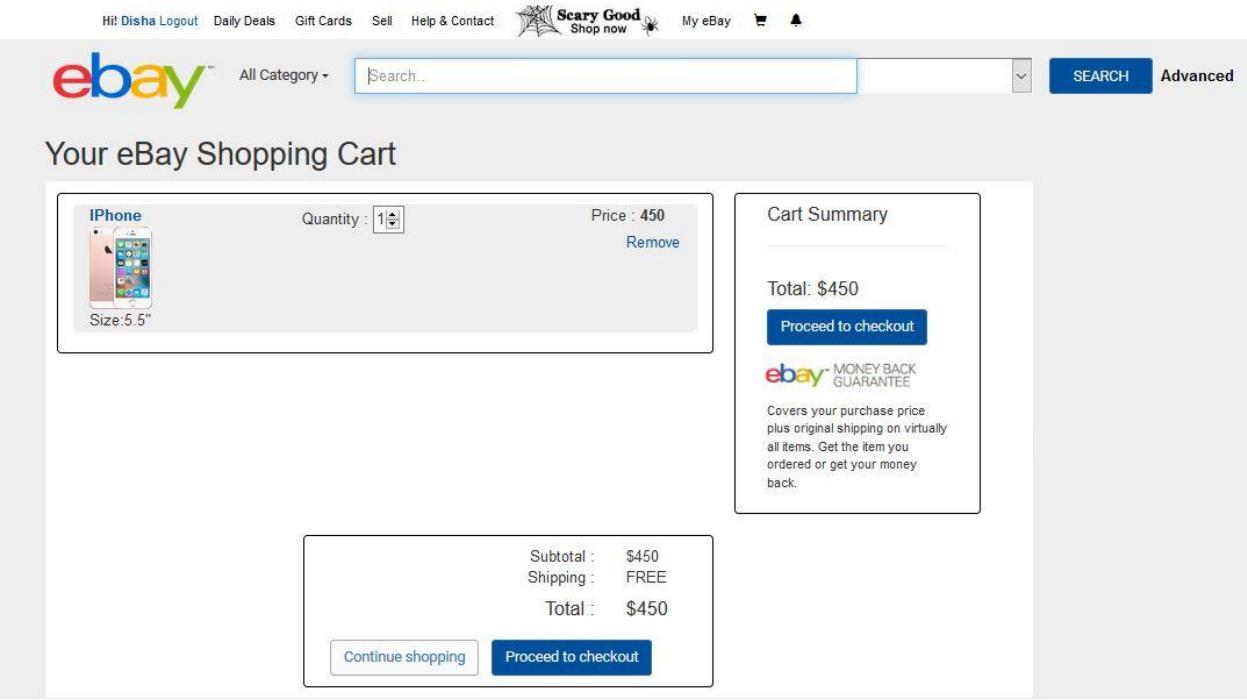
And clicking on More info will direct it to the corresponding product page :



Logged in user can add a product to cart for buying it later by clicking on “Add to Cart” or buy it immediately by clicking on “Buy it Now”

- ADD/REMOVE PRODUCT TO/FROM CART :

On adding the searched iPhone from page showing product details,



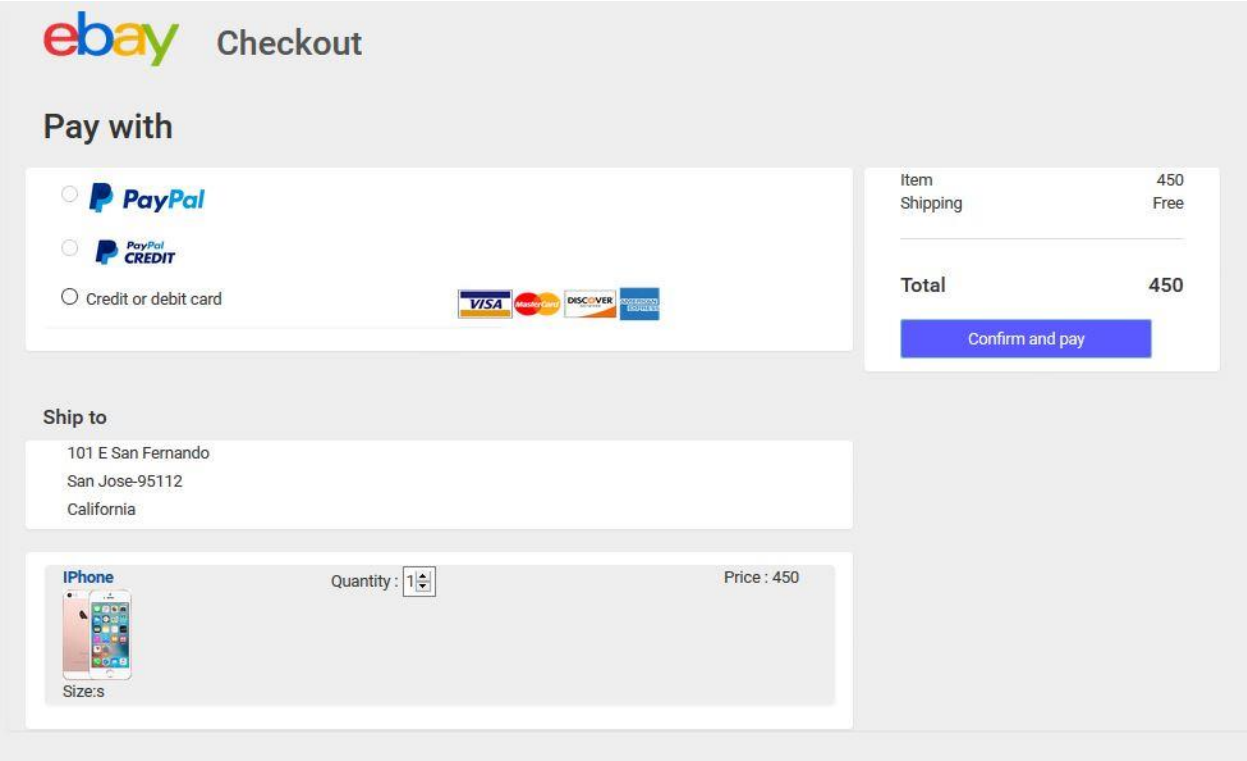
User can add/remove products from cart as per his desire. The products, on remove, will be removed from the user's cart which can be added again later.

- BUY NOW:

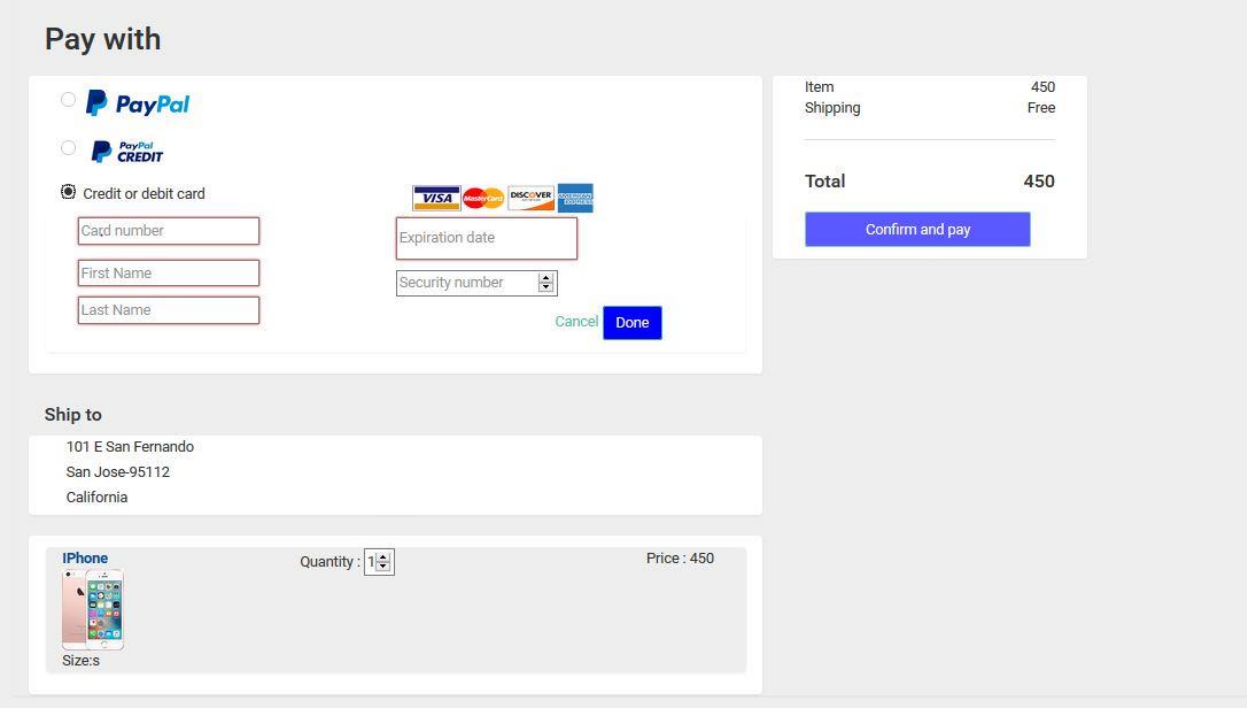
  Clicking on “Buy Now” from the page showing product details will directly redirect to the checkout for that particular product:

- PROCEED TO CHECKOUT :

  Proceed to checkout from Cart or checking out directly by doing “Buy Now” redirects to checkout page as under:



On selecting “credit or debit card options”, a pane is available which allows user to enter card details and validates these details.



Clicking on confirm and pay after validating the card details redirects back to the home page for the user to continue shopping. The inventory is taken care of after payment wherein the purchased amount and products get added to user’s purchase history and gets deducted from the seller’s inventory.

- **USER LOG GENERATION :**

User logs are generated wherein the login and surfing by a user is being recorded and can be analyzed to find out which user has traversed through which pages on the site and has viewed and added which products to the cart for checkout.

The logger file is generated as under :

mocha.js | package.json | index1.js | app.js | myebay1.log

```
2016-10-16 10:24:19.253 DEBUG h@h.h Logged into system

2016-10-16 10:35:22.774 DEBUG h@h.h Logged into system
2016-10-16 12:22:51.950 DEBUG disha.sheth@sjsu.edu Logged into system
2016-10-16 12:47:01.137 DEBUG disha.sheth@sjsu.edu Entered into cart
2016-10-16 12:47:20.896 DEBUG disha.sheth@sjsu.edu Entered into product page
2016-10-16 12:47:50.884 DEBUG disha.sheth@sjsu.edu Entered into cart
2016-10-16 12:48:16.963 DEBUG disha.sheth@sjsu.edu Entered into product page
2016-10-16 12:52:52.443 DEBUG disha.sheth@sjsu.edu Entered into product page
2016-10-16 15:14:47.189 DEBUG disha.sheth@sjsu.edu Searched for product %coin% an
2016-10-16 15:15:23.838 DEBUG disha.sheth@sjsu.edu requested for cart
2016-10-16 15:15:24.635 DEBUG disha.sheth@sjsu.edu Entered into product page
```

- **DATABASE CONNECTION POOLING :**

Database connection pooling is implemented for reducing the response time of the server. In this, a pool of open connections is created. Hence, for every new request, connection is to be just fetched and not created for every connection request. This reduces the response time of the server. On completion, the connection is returned to the pool from where it can be assigned for another request. A queue of pending requests is made which records the requests waiting for connection if the pool is already assigned and no connection is free/available.

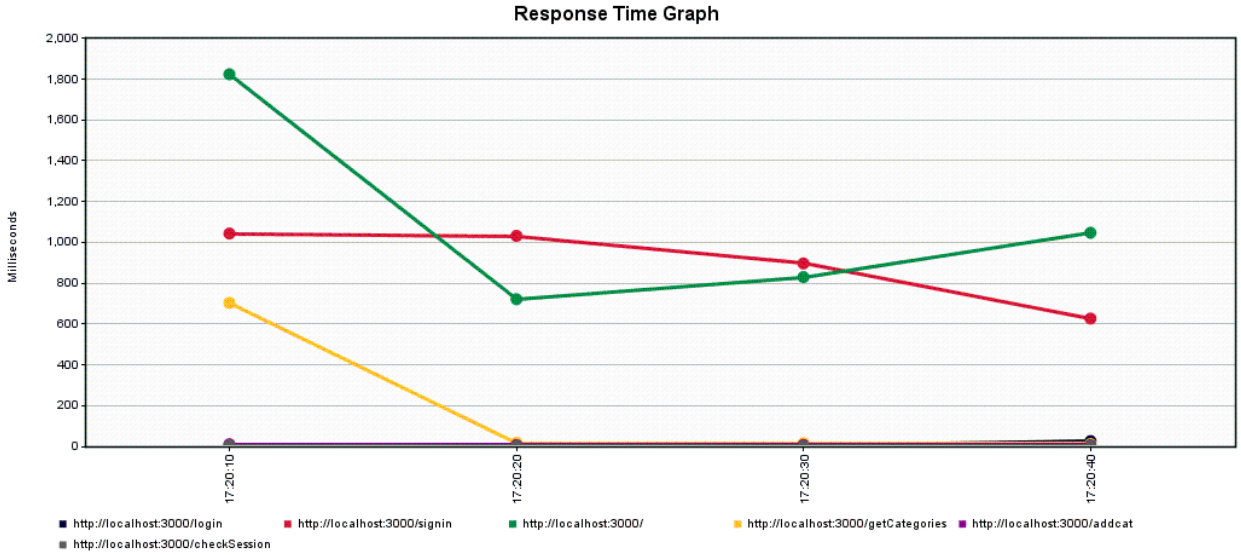
```
var MongoClient = require('mongodb').MongoClient;
var db;
var connected = false;
var ejs= require('ejs');//importing module ejs
var pool = [],free = [], req = [];
var current_connection;
var mongoURL = "mongodb://localhost:27017/myEbay";

/**

* Connects to the MongoDB Database with the provided URL

*/

exports.connect = function(url, callback){
var _db= p.get(url);
db = _db;
connected = true;
console.log(connected +" is connected?");
callback(db);
p.release(db);
};

exports.collection = function(name){
if (!connected) {
throw new Error('Must connect to Mongo before calling "collection"');
}

var coll=db.collection(name);
return coll;
};

function getConnection(pool){
MongoClient.connect(mongoURL, function(err, _db){
if (err) { throw new Error('Could not connect: '+err); }
db = _db;
connected = true;
pool.push(db);
});

}
```

```
}

function Pool()
{
for(var i=0; i < 500; ++i)
{
getConnection(pool);
free.push(i);
}
}

Pool.prototype.get = function(request_name)
{
if(free.length > 0)
{
var db = pool[free.length-1];
free.pop();
current_db = db;
return current_db;
}
else
{
req.push(request_name);
return null;
}
}

Pool.prototype.release = function(number)
{
free.push(number);
if(req.length > 0)
{
Pool.get(req[0]);
req.slice(0,1);
}
};
var p=new Pool();
```

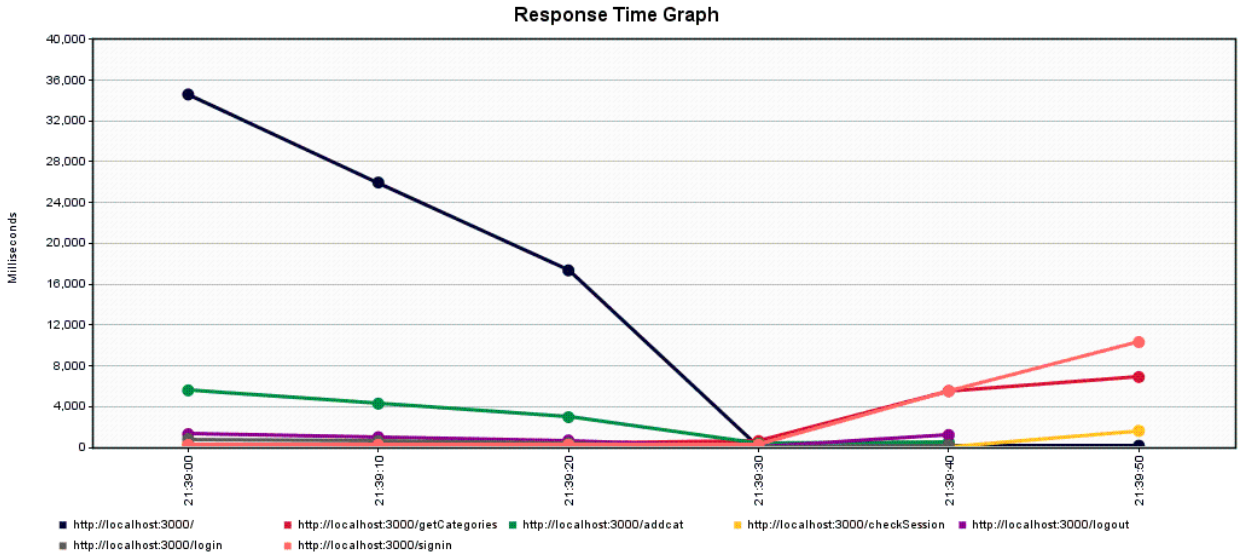
- **JMeter testing :**

1. **100 calls**

Before connection pooling :

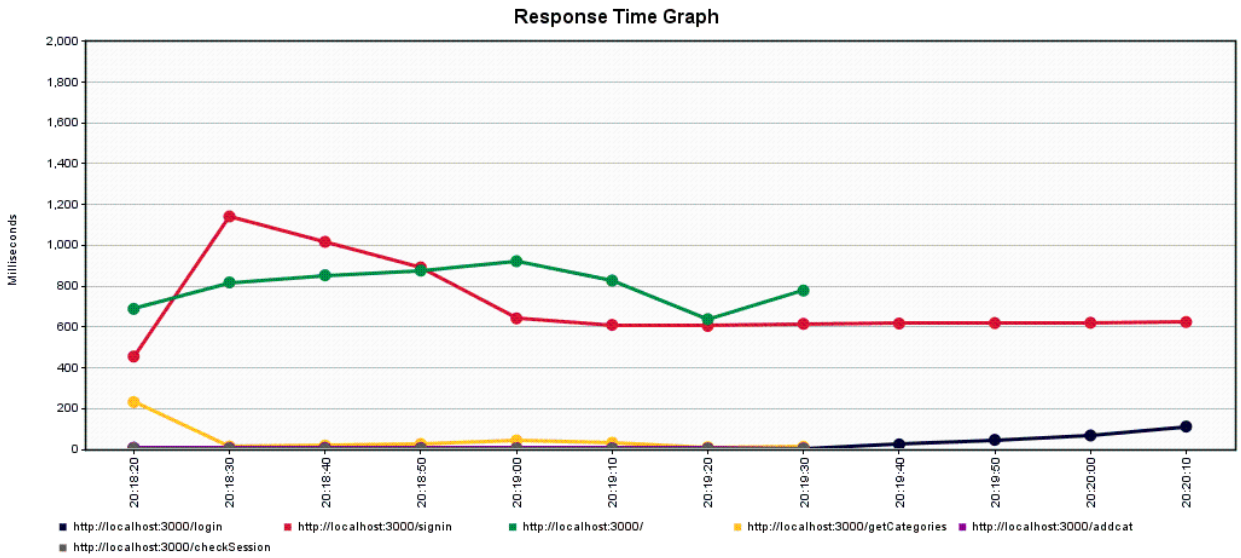


After connection pooling :

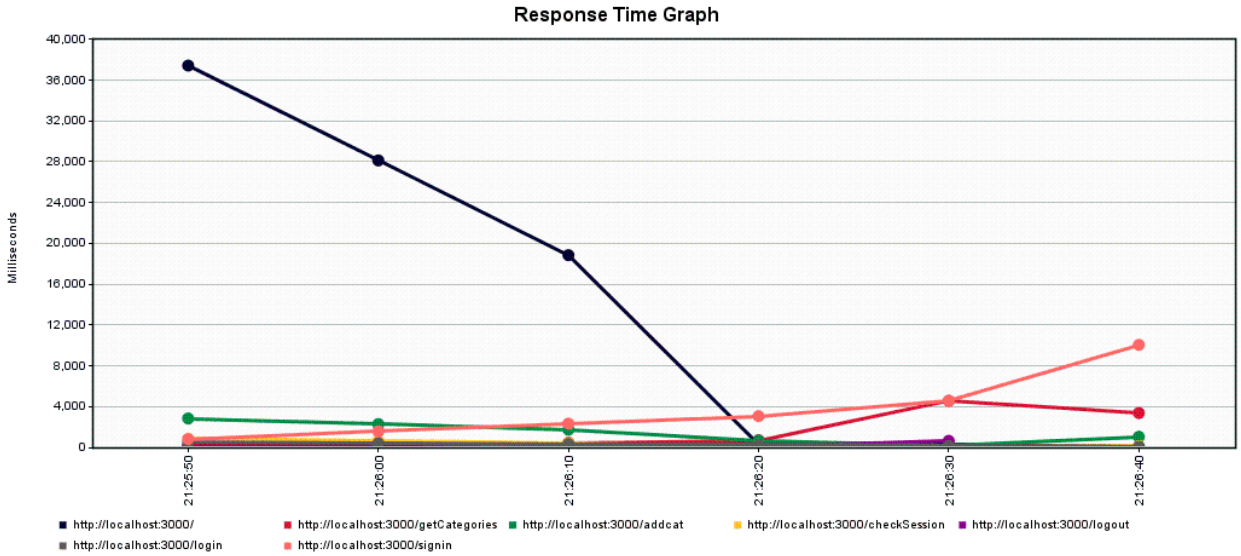


2. **200 calls**

Before connection pooling :

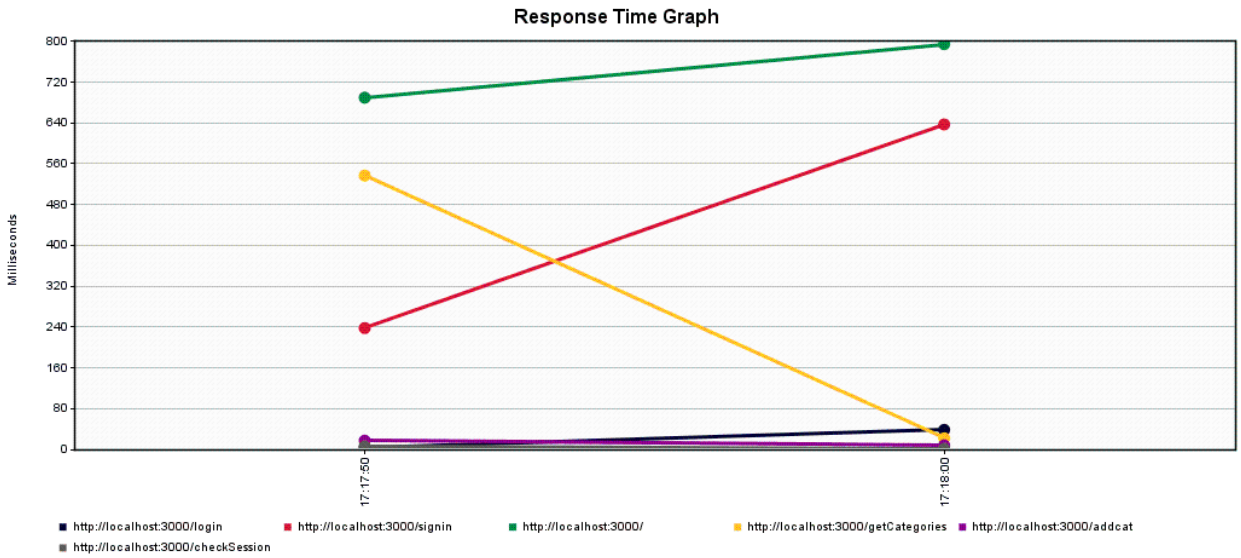


After connection pooling :

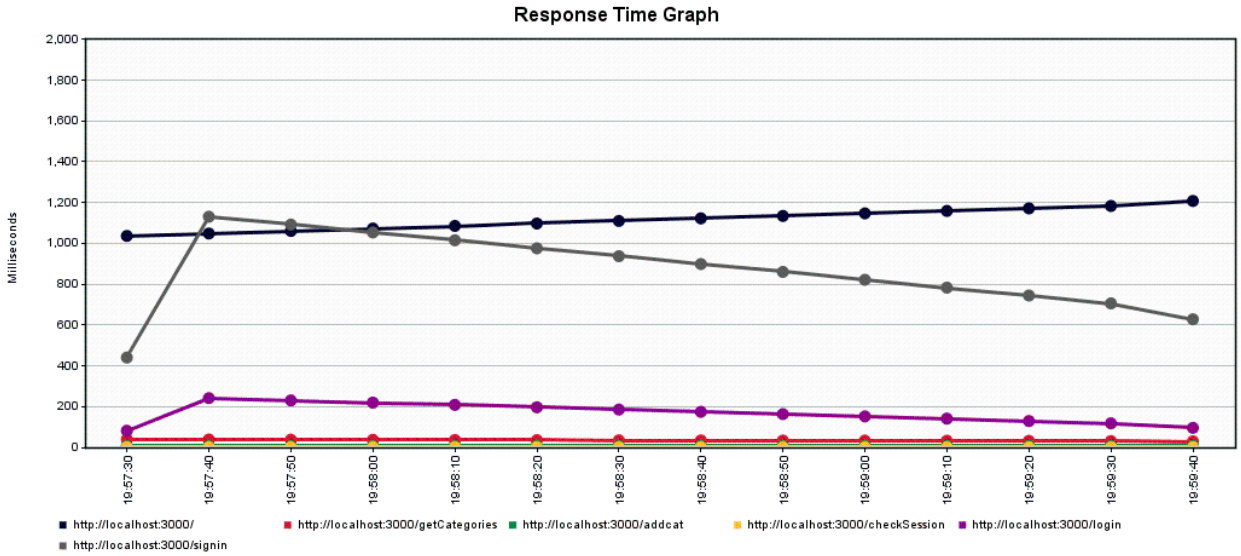


3. 300 calls

Before connection pooling :

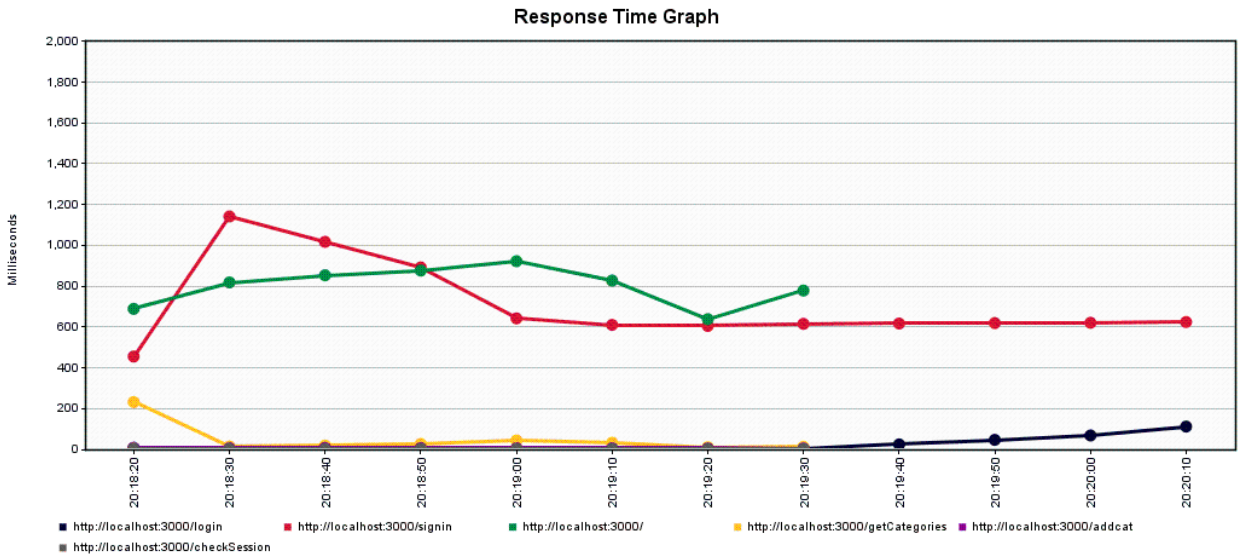


After connection pooling :

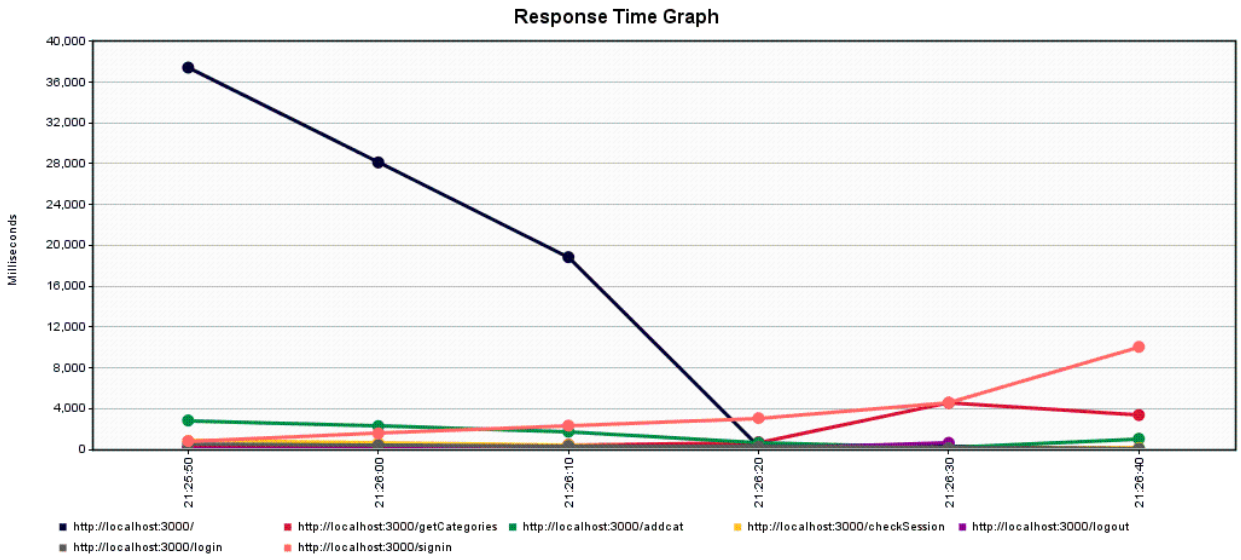


4. **400 calls**

   Before connection pooling :

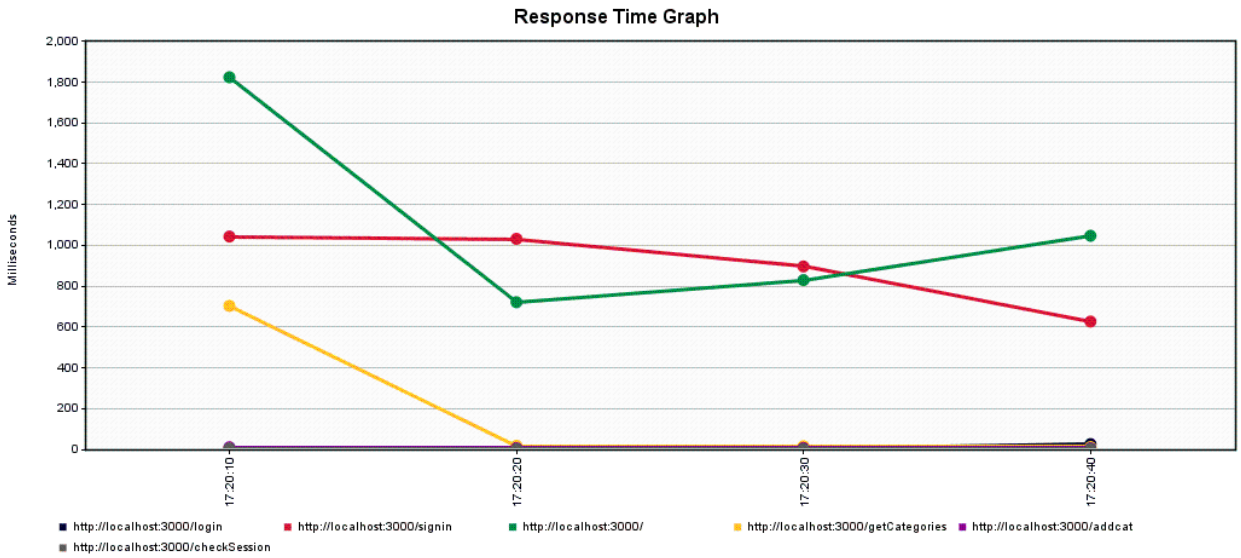


After connection pooling :

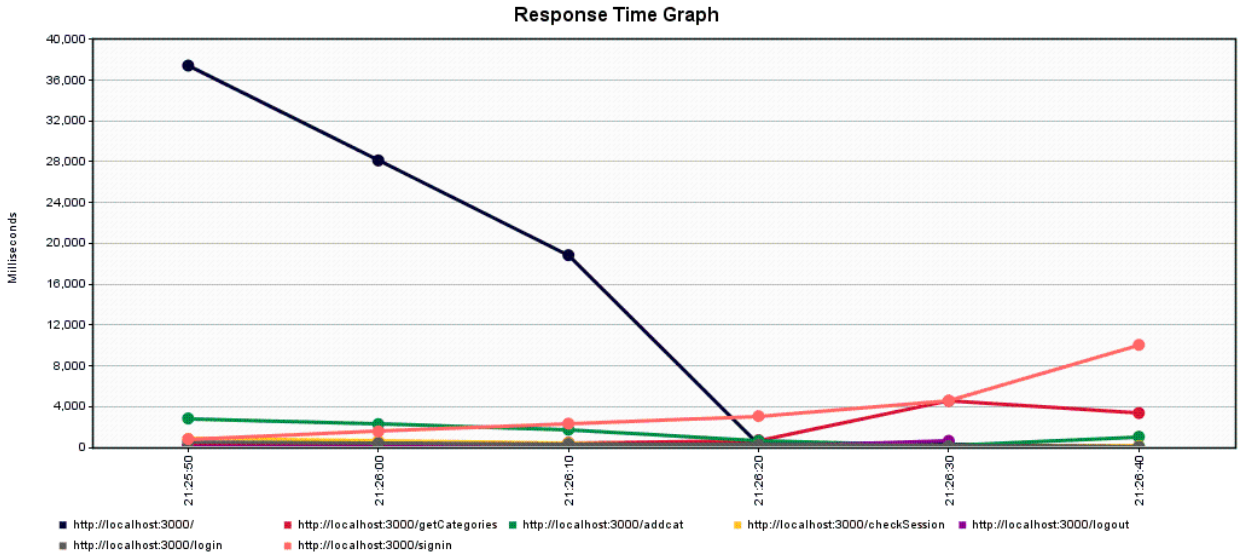


5. 500 calls

Before connection pooling :

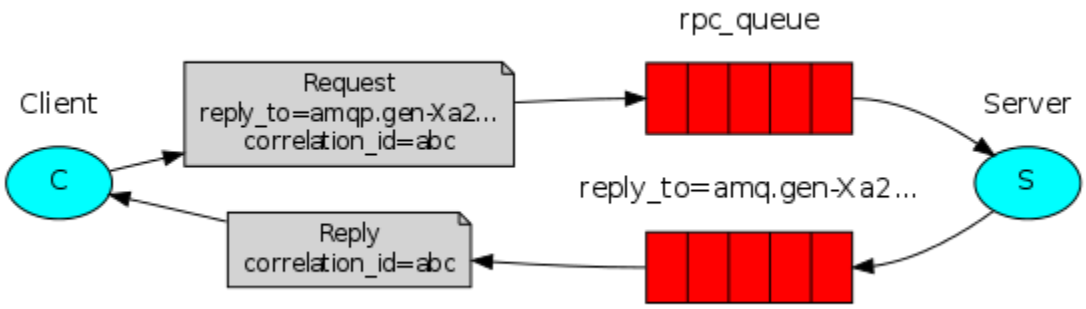


After connection pooling :



RABBITMQ ARCHITECTURE:

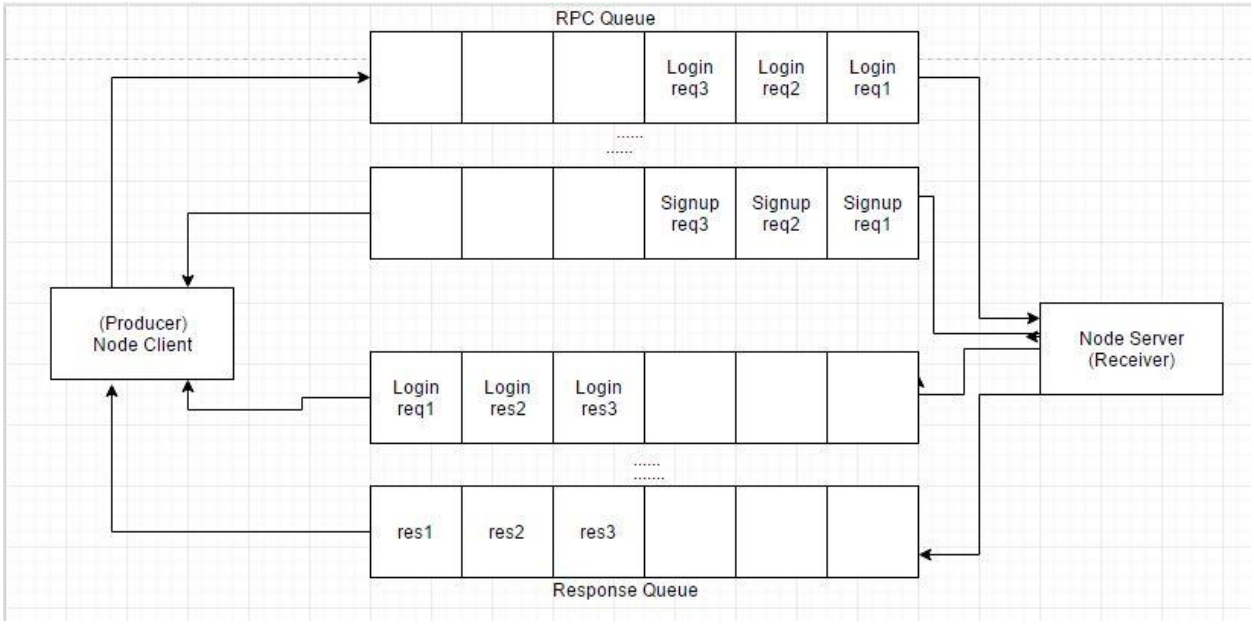
A client sends a request message and a server replies with a response message. In order to receive a response we need to send a 'callback' queue address with the request.



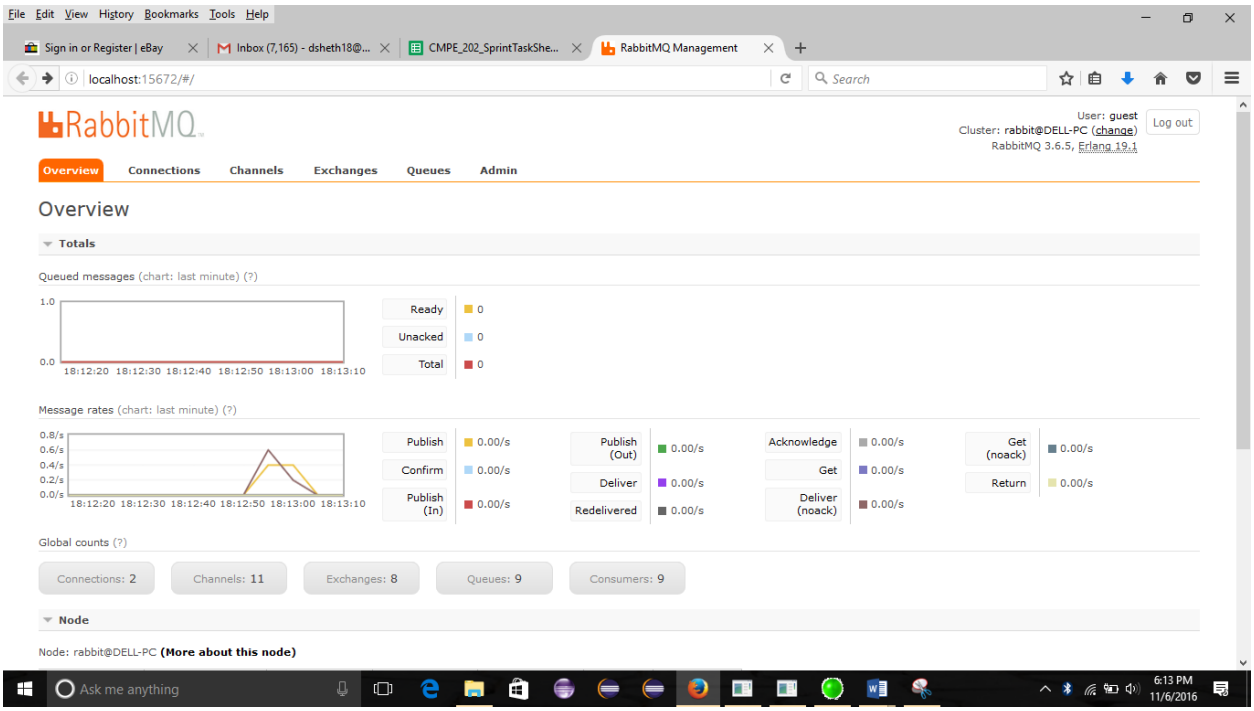
The following queues are created in our application of eBay marketplace.

Overview		
Name	Features	**State**
addtocart	AD	idle
amq.gen-e-TPnJaoZmxduuQa9EbfRQ	Excl AD	idle
cart	AD	idle
checkout	AD	idle
itemdetails	AD	idle
login_queue	AD	idle
register	AD	idle
search	AD	idle
sell	AD	idle

The following architecture has been implemented in the application for rabbitMQ:



When a message is sent over to the server, the same is reflected on rabbitMQ management console. Example of the same is as below :



PART 2:

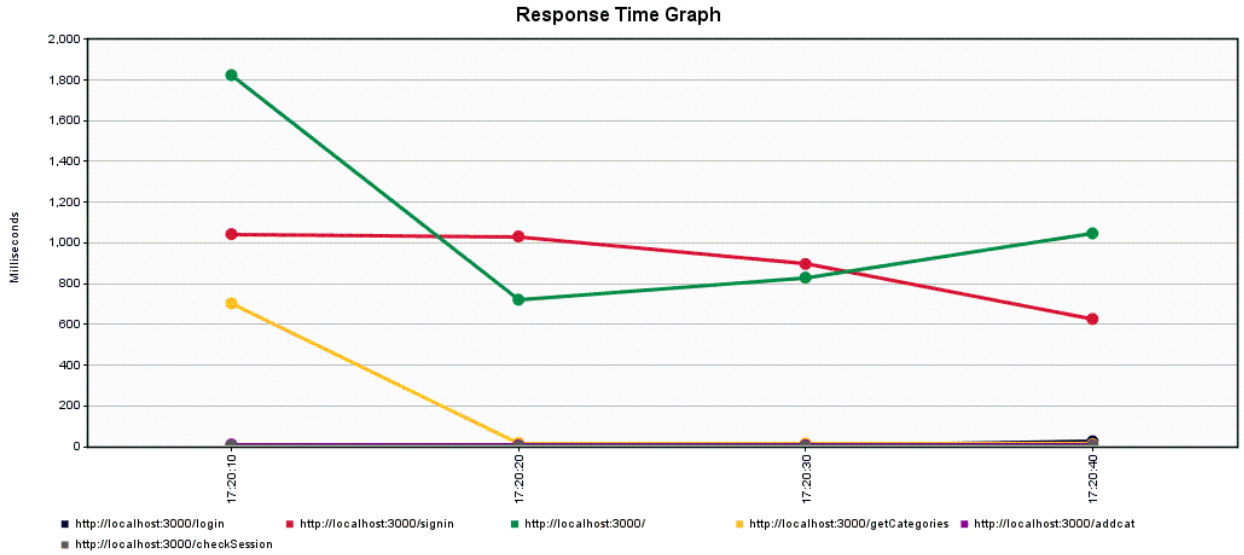
1. **Explain what performance change RabbitMQ provides? Elaborate on the results of throughput with and without using RabbitMQ. If you find any increase/decrease in the throughput, explain the reason for the same.**

   Decoupling of server side into dedicated queues will enhance the performance and reduce the response time of the web-application. This in turn will improve the throughput of the application. The other reason to use message queues is scalability. It significantly improves the scalability of application. Multiple requests can be catered simultaneously using various dedicated queues. So response latency decreases. The graphical results shows the comparison between two cases.

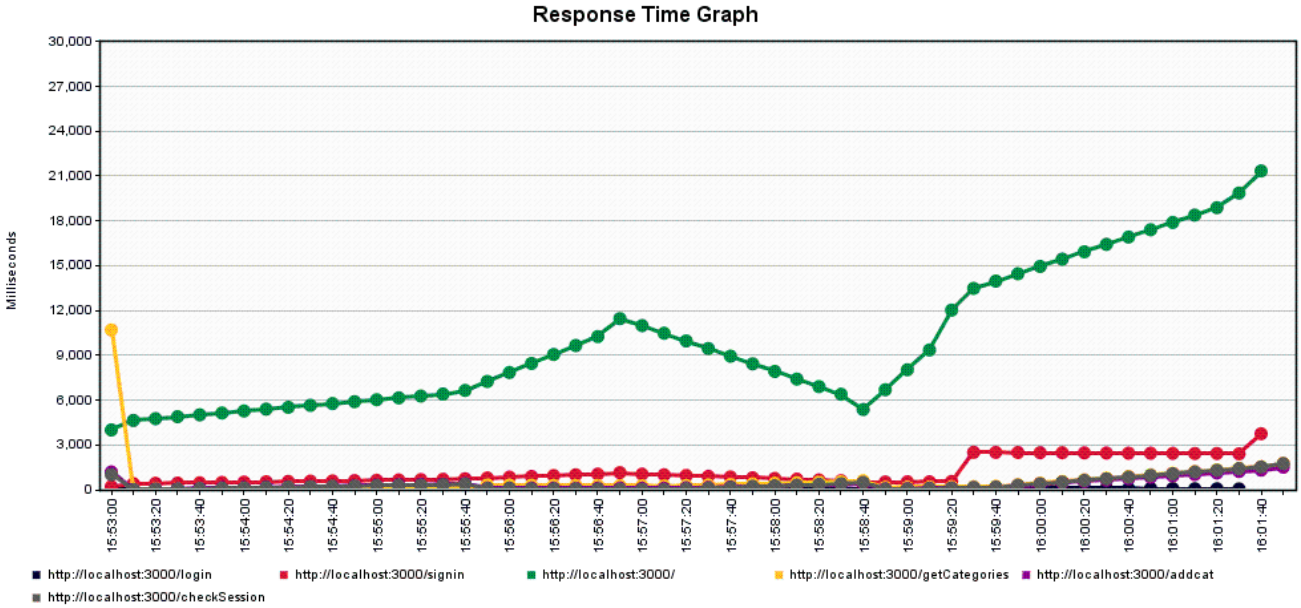
   Graphs showing average response time after using RabbitMQ are as below :

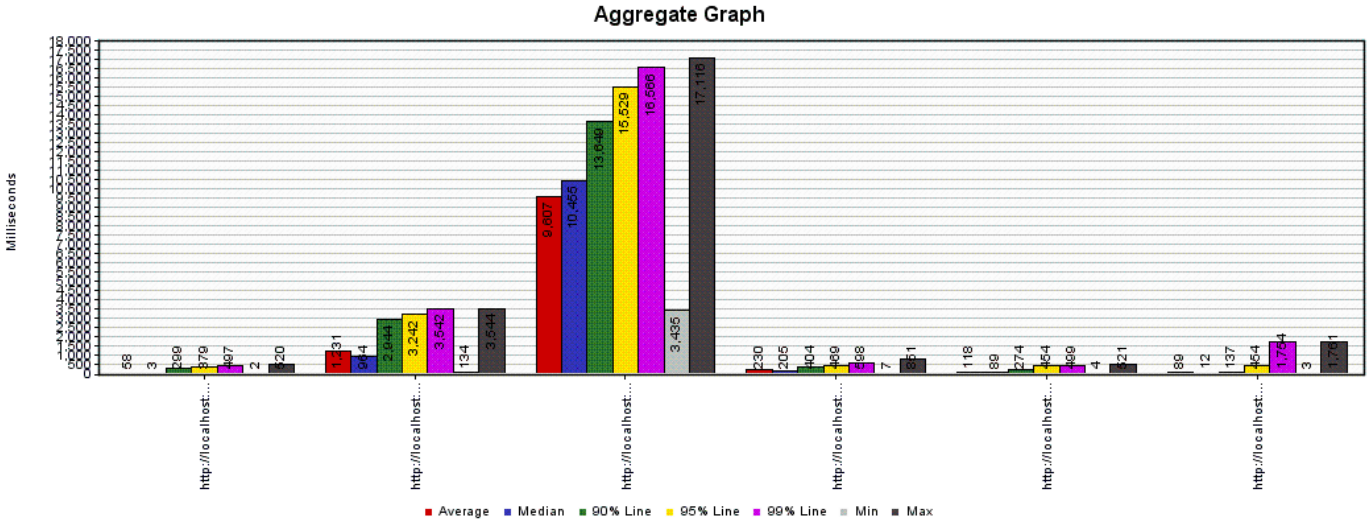
   1) 100 users

   Before message queuing:



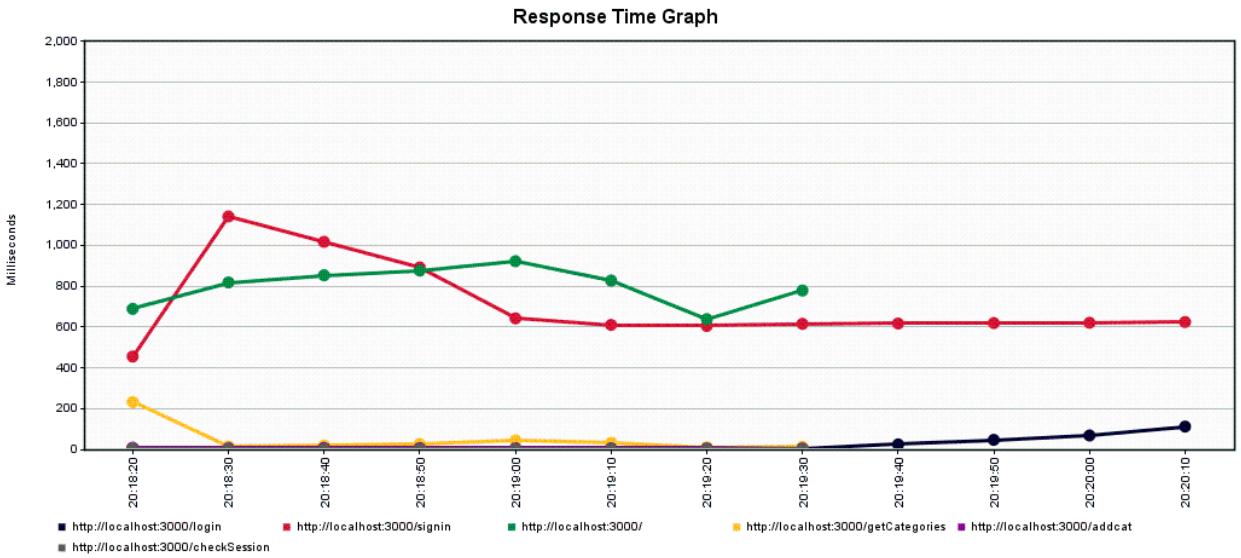
After message queuing:



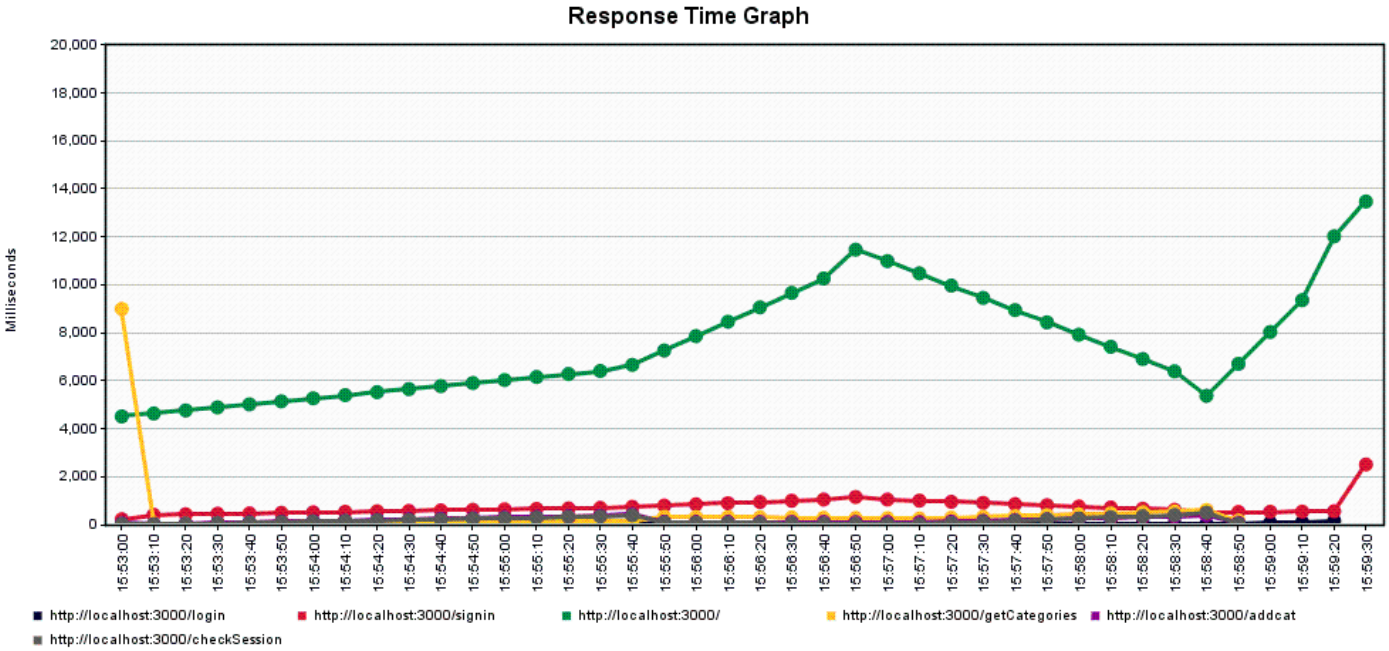


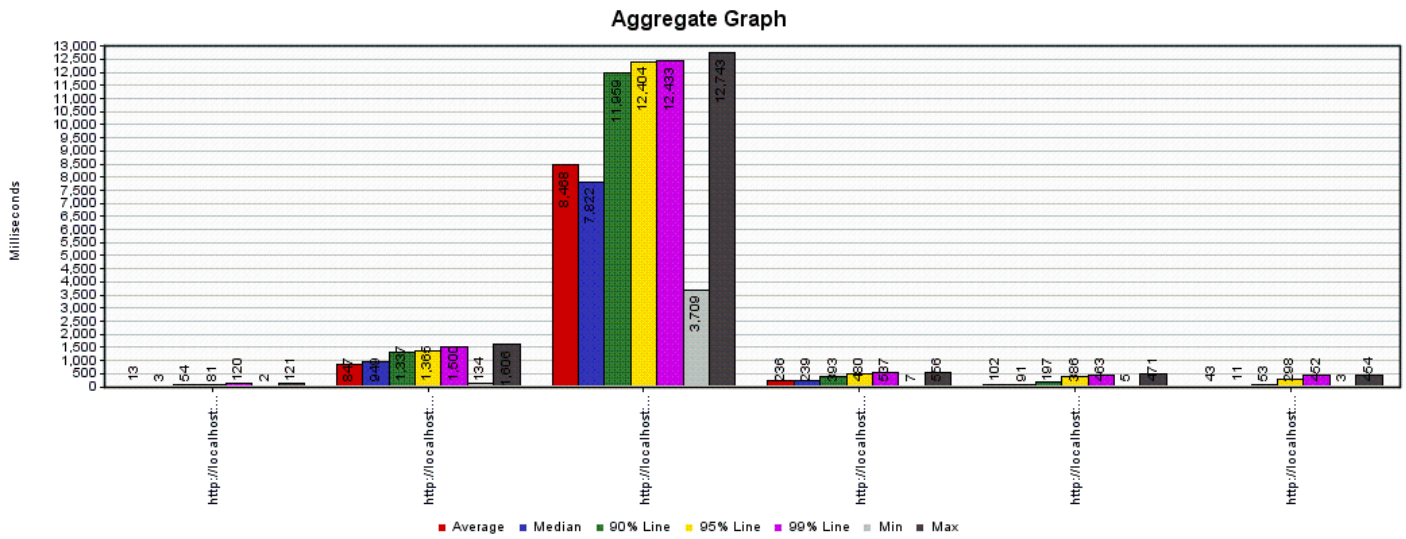
2) 200 users

Before message queuing:



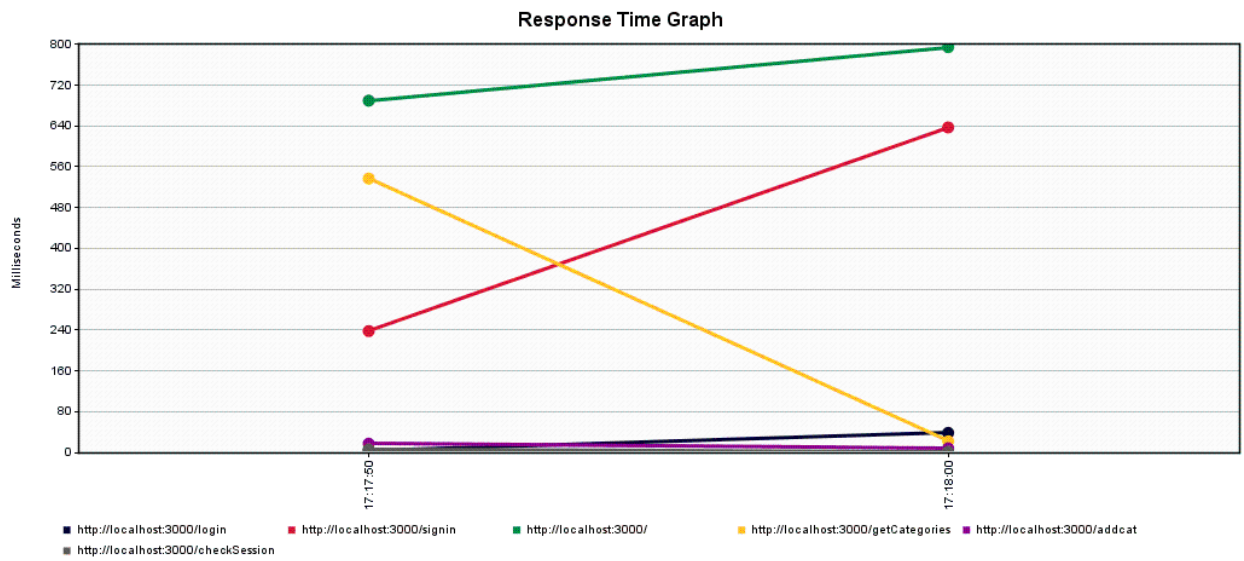
After message queuing:



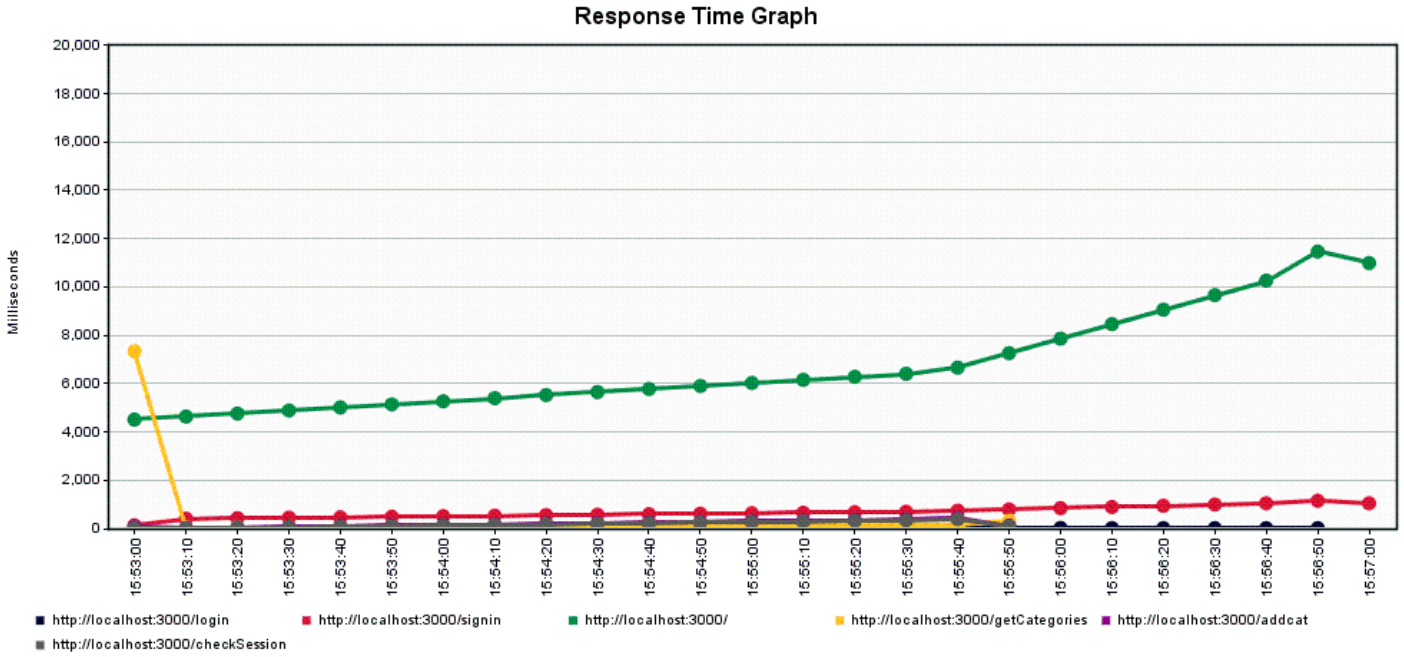


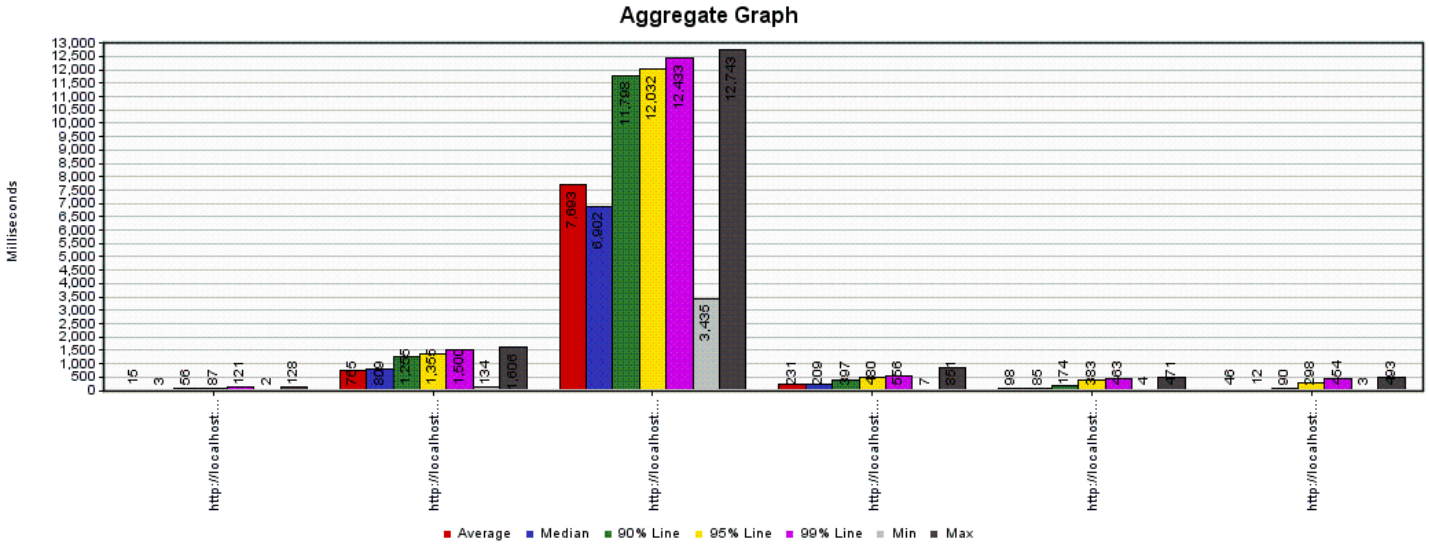
3) 300 users

Before message queuing:



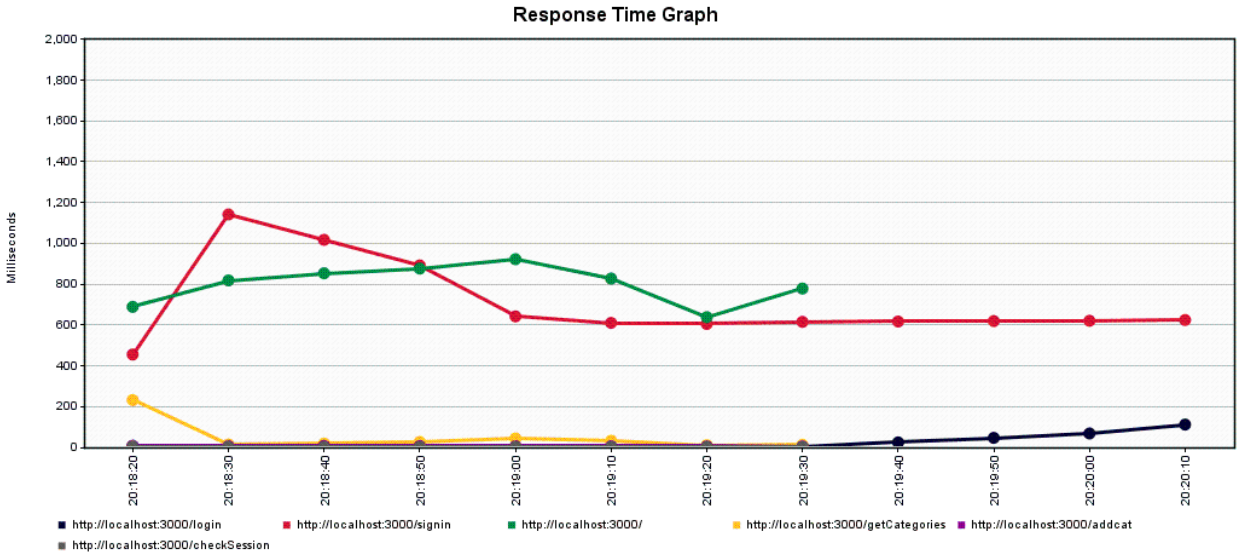
After message queuing:





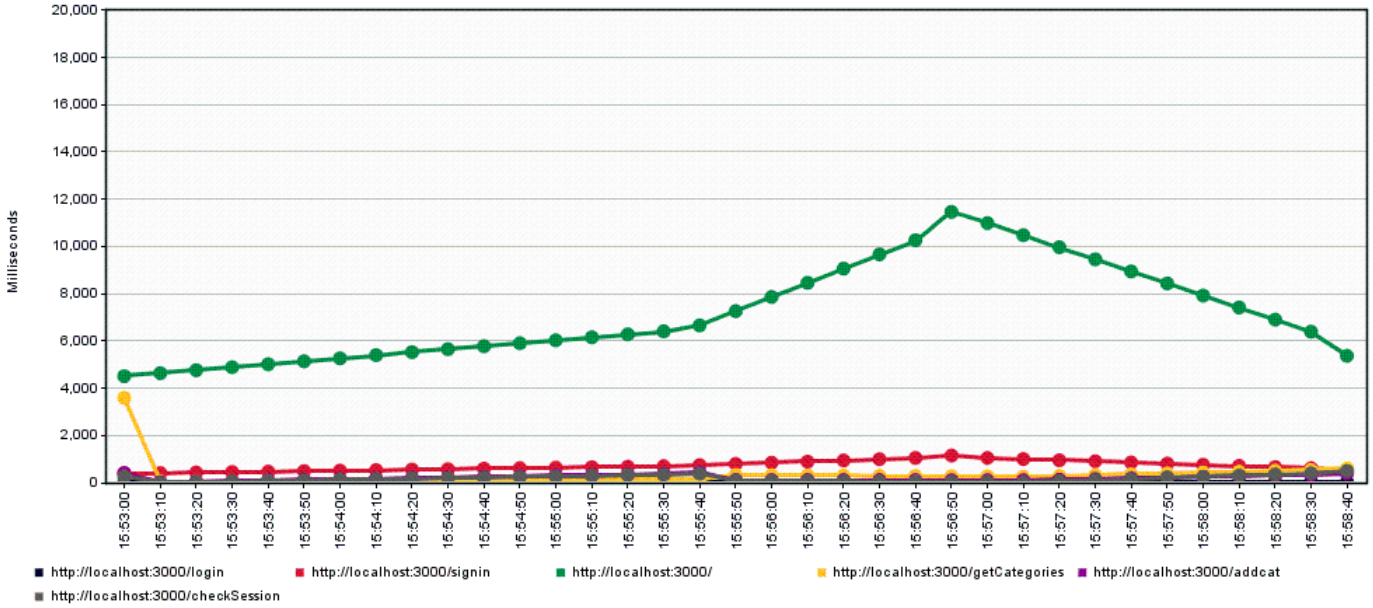
4) 400 users

Before message queuing:

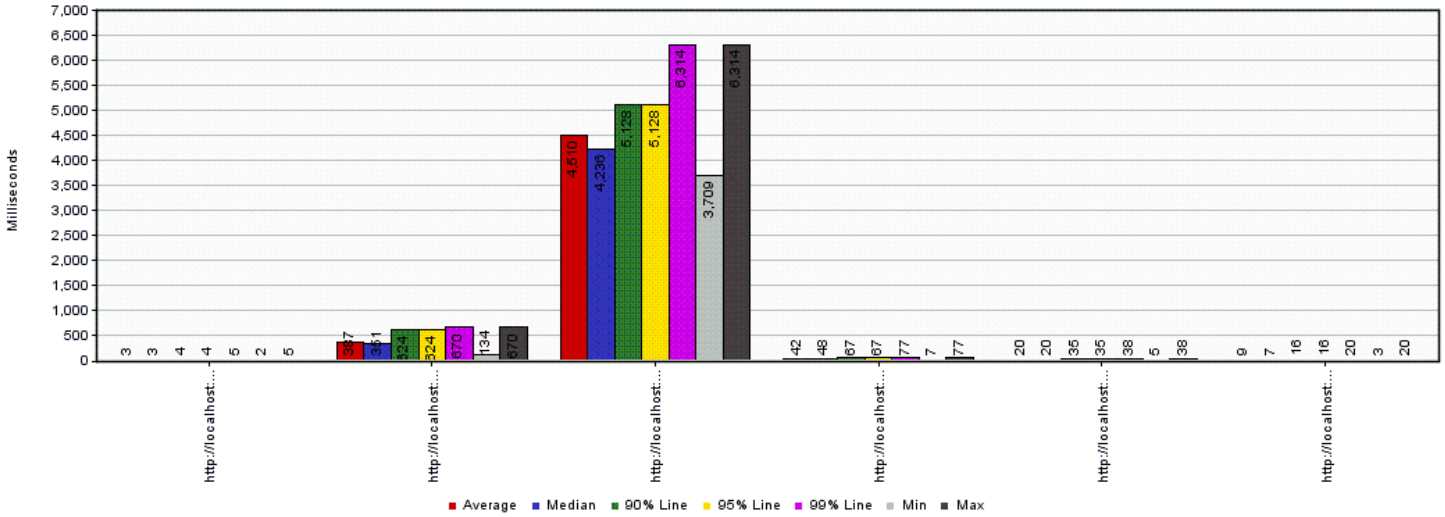


After message queuing:

Response Time Graph

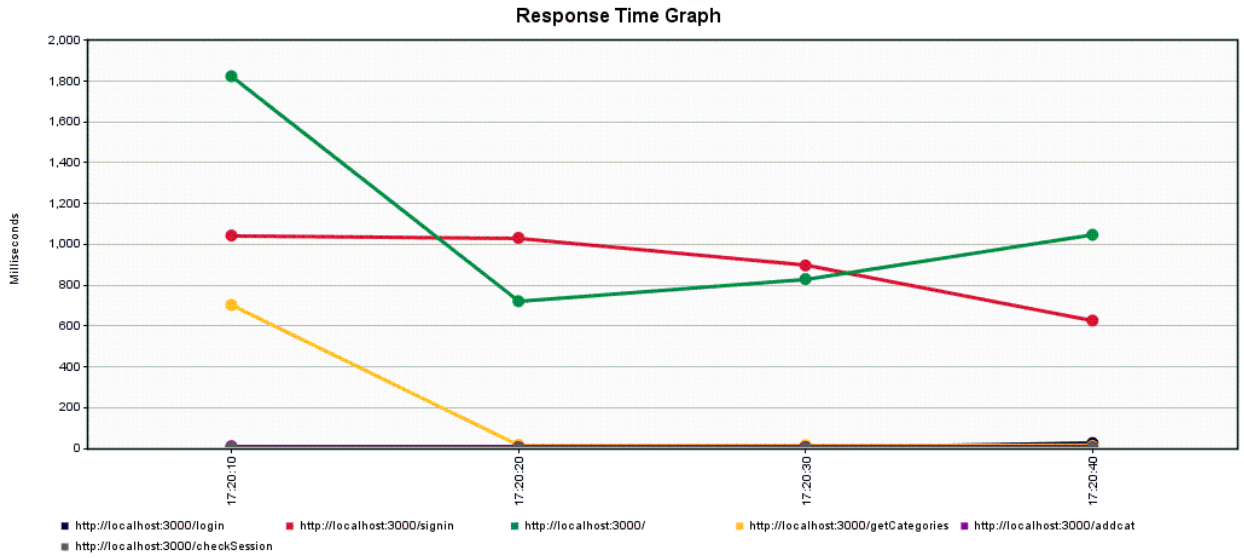


Aggregate Graph



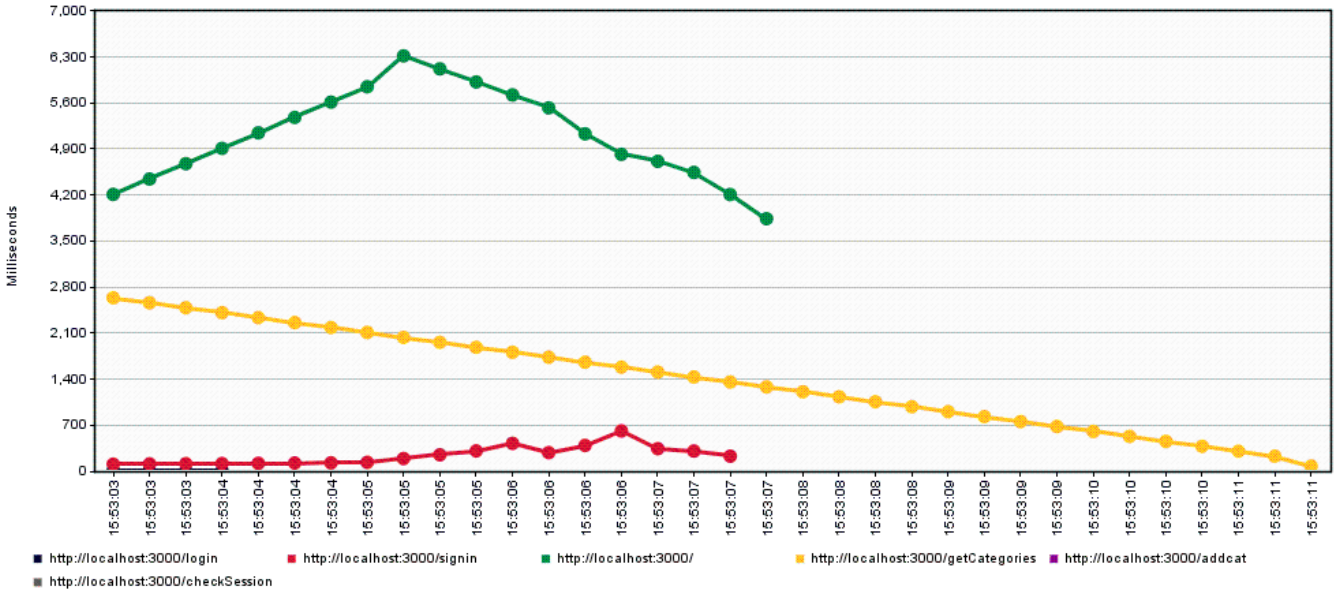
5) 500 users

Before message queuing:

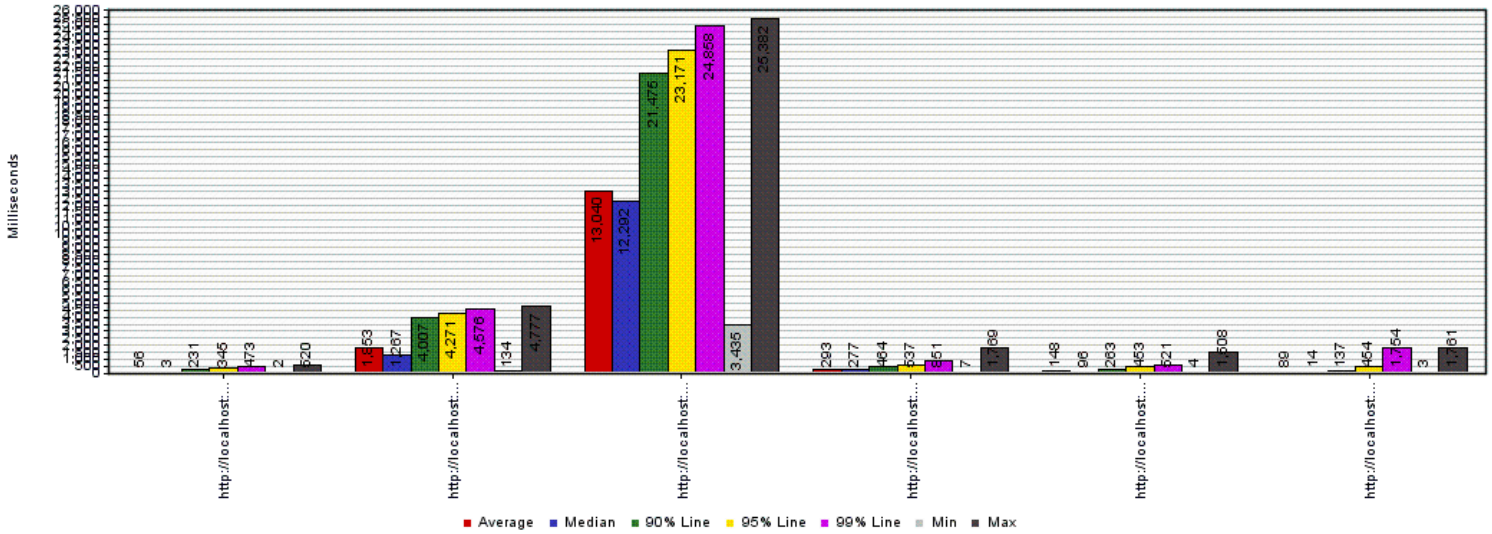


After message queuing:

Response Time Graph



Aggregate Graph



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

PassportJS is the pre-defined module used widely for user authentication. It helps in validating the user and hence prevents intrusion by unauthorized user. All sign-in and profile cookies are strongly encrypted. By developing a strategy, common user ID and passwords can be used across different sites.

The passportJS code is as below.

app.js

```
var passport = require('passport');
..
..
passport.initialize();
..
..
app.post('/signin',passport.authenticate(function(req,res,user)
{
//logic
}
));
```

This file initialises the passport module. When the API call is made is calls the passport.js file. This passport.js file interacts with RabbitMQ Backend that interacts with mongoDB.

The code for *passport.js* is below:

```
module.exports = function(passport) {
  passport.use('login', new LocalStrategy(function(username, password, done) {

   var msg_payload = {
   "username" : username,
   "password" : password
   };


     process.nextTick(function(){
```

```
        mq_client.make_request('login_queue', msg_payload, function(err, results) {
        var user=results.user;
        console.log("logging user....."+user);
        if(err) {
                return done(err);
            }
            console.log("user....."+user)
            if(!user) {
                return done(null, false);
            }

            if(user.password != password) {
                done(null, false);
            }

             console.log(user.username);
            done(null, user);
        });


            });

    }));
}
```

3. **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**

   MySQL (RDBMS) and MongoDB (NoSQL) have their own pros and cons. If given an option to implement both in the application, I would implement MySQL in order to store list of products. This is because of the size and usability of the table. If this is stored in mongoDB data redundancy increases. All the information related to the users is stored in mongoDB. This includes product purchased, products sold, profile, etc. Each document corresponds to a user. This will decrease the retrieval time of the query and improve the performance. RDBMS, if used in this situation, the query becomes very complex due to large number of joins and keys.