

## LAB 2

# CMPE 273 – Enterprise Distributed Systems

---

Prince Jacob Chandy (010807225)

github link: <https://github.com/princejacobchandy/CMPE273-LAB2>

Username: prince.jacobchandy@sjsu.edu

Password: cmpe273labs

## I. Introduction (Goals and purpose of your system)

This assignment covers developing REST services using Node.js (Express JS), HTML5 and Angular JS, with **MongoDB** service used as the server side database.

The main focus is to build a message-oriented system using **RabbitMQ** which implements Advanced Message Queuing Protocol.

Sessions and **session information are stored in MongoDB** and **passwords are authenticated using passportjs** (passport.authenticate)

Mock eBay web application is developed which replicates the actual eBay application. The developed application offers the following services:

login page: provides existing user with the option to sign in and new users with the option to register

home page: where user can view, the advertisings posted by other users and initiate to buy or bid a product of his/her liking

profile page: where user can view, and update his profile information and create new advertisements (normal selling or bidding)

profile page includes other important information such as sale history, purchase history and bid history.

shopping cart page: which displays the items added by the user

checkout page: which provides user option to add/update payment/credit card information and shipping address.

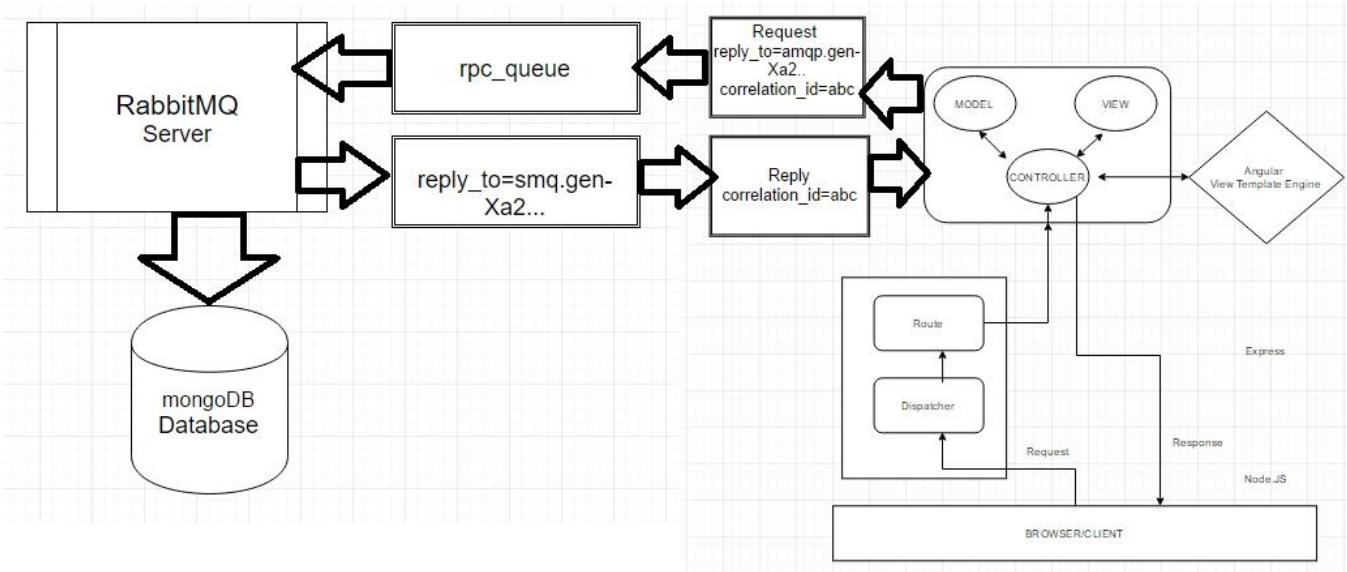
user can confirm buying of items in his shopping cart via checkout option

connection pooling for mongoDB connections

validations for all inputs provided by the user

selling, bidding and purchase

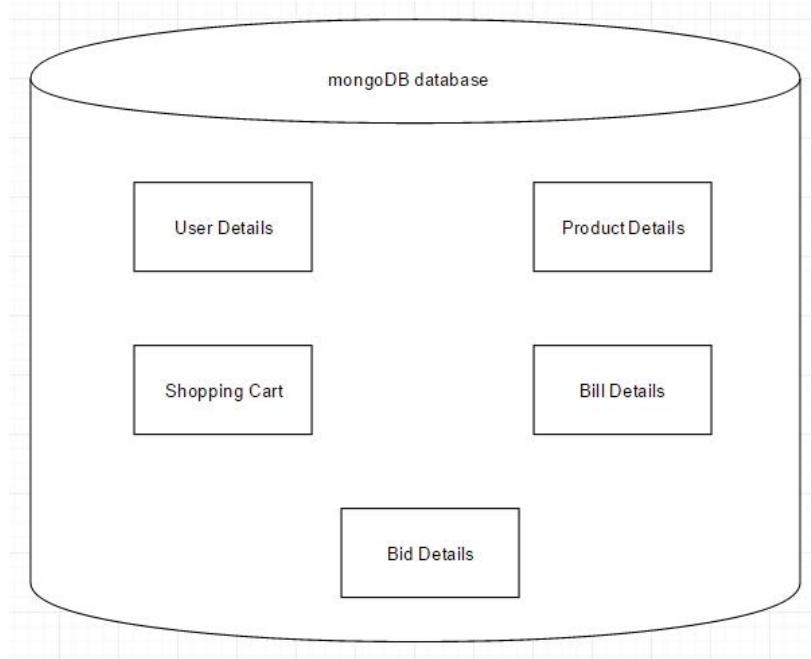
## II. System Design



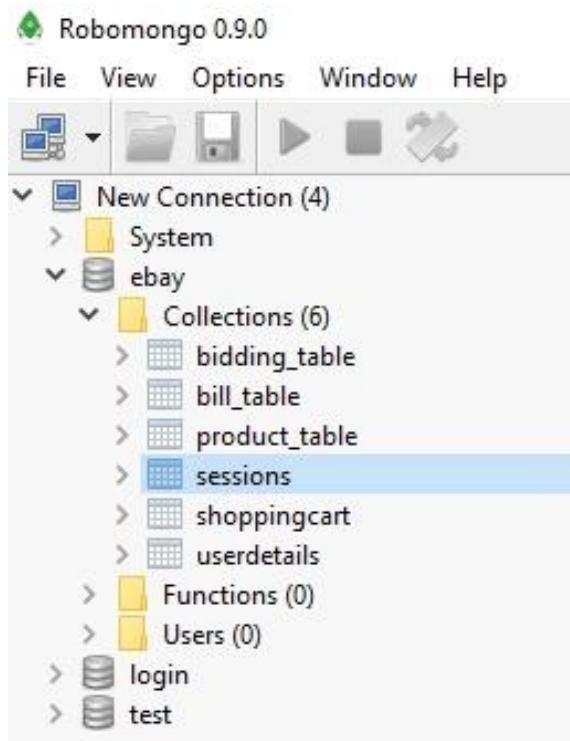
The view (HTML5) is controlled by AngularJS through Express Controller

Server side services are implemented by Node.JS

Server establishes connection via a connection pool and talks to mongoDB database module



mongoDB Database design with the various collections (1)



mongoDB Database design with the various collections (2)

```

Command Prompt - mongo
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.

C:\Users\Prince Jacob Chandy>mongo
MongoDB shell version: 3.2.10
connecting to: test
> show dbs;
ebay  0.000GB
local  0.000GB
login  0.000GB
test   0.000GB
> use ebay;
switched to db ebay
> show collections
bidding_table
bill_table
product_table
sessions
shoppingcart
userdetails
>

```

### mongoDB Database design with the various collections (3)

Key	Value	Type
ObjectID("581fb3bebafbf92ddc2b8850")	{ 10 fields }	Object
_id	ObjectID("581fb3bebafbf92ddc2b8850")	ObjectID
product_id	581fa57749052c2988de96f4	String
product_name	Roadmaster Bikerboy Roadbike	String
product_desc	26 Road Bike for sale	String
product_qty	1	Int32
product_seller	riotfrock@gmail.com	String
bid_price	90	Int32
actual_price	80	Int32
bid_endtime	2016-11-10T21:49:43.574Z	String
ObjectID("581fb477baebafbf92ddc2b8853")	{ 10 fields }	Object
_id	ObjectID("581fb477baebafbf92ddc2b8853")	ObjectID
product_id	581fa50e49052c2988de96f2	String
product_name	Schwinn Mountain Bike	String
product_desc	26 Mountain Bike for sale	String
product_qty	1	Int32
product_seller	riotfrock@gmail.com	String
bid_price	160	Int32
actual_price	150	Int32
bid_endtime	2016-11-10T21:47:58.043Z	String

Bid details collection

**Robomongo 0.9.0**

File View Options Window Help

New Connection (4)

System ebay

Collections (6)

- bidding\_table
- bill\_table
- product\_table
- sessions
- shoppingcart
- userdetails

Functions (0)

Users (0)

login test

db.getCollection('sessions').find({}) db.getCollection('product\_table').find({})

db.getCollection('product\_table').find({})

product\_table 0.001 sec.

Key	Value	Type
(1) ObjectID("581f9ac03ec04a131883b330")	{ 9 fields }	Object
_id	ObjectID("581f9ac03ec04a131883b330")	ObjectId
seller_handle	null	Null
seller_uname	prince	String
product_name	Roadmaster Granite Peak Bike	String
product_description	26inch Steel Frame - Roadmaster Granite Peak Trails Black or Blue	String
product_qty	9	Int32
product_price	100	Int32
seller_email	princeofalltimes@gmail.com	String
product_type	1	Int32
(2) ObjectID("581f9c9d4db0684ccc3bb04f")	{ 9 fields }	Object
(3) ObjectID("581f9c1dbb0684ccc3bb050")	{ 9 fields }	Object
(4) ObjectID("581f9cf0fb0684ccc3bb051")	{ 9 fields }	Object
(5) ObjectID("581f9cf0fb0684ccc3bb052")	{ 9 fields }	Object
(6) ObjectID("581f9cd0fb0684ccc3bb053")	{ 9 fields }	Object
(7) ObjectID("581f9c3dbb0684ccc3bb054")	{ 9 fields }	Object
(8) ObjectID("581fa0e23ec04a131883b331")	{ 9 fields }	Object
(9) ObjectID("581fa2b2b09052c2988de96ea")	{ 9 fields }	Object
(10) ObjectID("581fa2b7949052c2988de96eb")	{ 9 fields }	Object
(11) ObjectID("581fa2c49052c2988de96ec")	{ 11 fields }	Object
(12) ObjectID("581fa50e49052c2988de96f2")	{ 11 fields }	Object
(13) ObjectID("581fa53349052c2988de96f3")	{ 9 fields }	Object
(14) ObjectID("581fa57749052c2988de96f4")	{ 11 fields }	Object
_id	ObjectID("581fa57749052c2988de96f4")	ObjectId
seller_handle	null	Null
seller_uname	riotofrock	String
product_name	Roadmaster Bikerboy Roadbike	String
product_description	26 Road Bike for sale	String
product_qty	20	Int32
product_price	80	Int32
seller_email	riotofrock@gmail.com	String
product_type	2	Int32
bid_start	2016-11-06T21:49:43.574Z	String
bid_end	2016-11-10T21:49:43.574Z	String

## Product details collection

**Robomongo 0.9.0**

File View Options Window Help

New Connection (4)

System ebay

Collections (6)

- bidding\_table
- bill\_table
- product\_table
- sessions
- shoppingcart
- userdetails

Functions (0)

Users (0)

login test

db.getCollection('bill\_table').find({})

db.getCollection('bill\_table').find({})

bill\_table 0 sec.

Key	Value	Type
(1) ObjectID("581fa4bb49052c2988de96ef")	{ 11 fields }	Object
_id	ObjectID("581fa4bb49052c2988de96ef")	ObjectId
product_id	581f9ac03ec04a131883b330	String
product_name	Roadmaster Granite Peak Bike	String
product_qty	1	Int32
product_price	100	Int32
buyer_name	riotofrock@gmail.com	String
seller_name	princeofalltimes@gmail.com	String
totalamt	100	Int32
totalitems	1	Int32
buyer_address	Kanjikuzhy, Kottayam, Kerala, India, 123456	String
bill_payment	*****3456	String
(2) ObjectID("581fa4dd49052c2988de96f1")	{ 11 fields }	Object
_id	ObjectID("581fa4dd49052c2988de96f1")	ObjectId
product_id	581fa3749052c2988de96b	String
product_name	Razer Electra Headphones	String
product_qty	1	Int32
product_price	44	Int32
buyer_name	riotofrock@gmail.com	String
seller_name	princeofalltimes@gmail.com	String
totalamt	44	Int32
totalitems	1	Int32
buyer_address	Kanjikuzhy, Kottayam, Kerala, India, 123456	String
bill_payment	*****3456	String
(3) ObjectID("581fad249052c2988de96f8")	{ 11 fields }	Object
(4) ObjectID("581fb462bafb92ddc2b882")	{ 11 fields }	Object

## Bill details collection

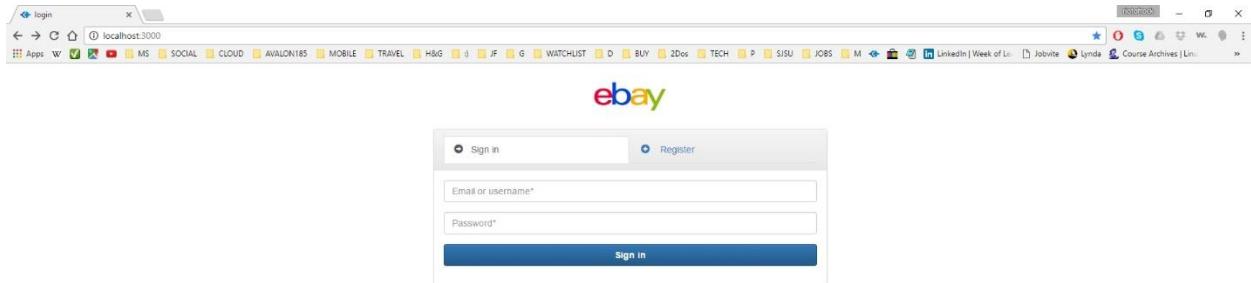
The screenshot shows the Robomongo interface with a connection to the 'ebay' database. The 'shoppingcart' collection is selected. A query `db.getCollection('shoppingcart').find({})` is run, returning two documents. The first document has an \_id of '581fd538725bb81b04209965'. It contains fields like buyer\_uname ('princeofalltimes@gmail.com'), product\_id ('5819cfdf6b0684cc3bb051'), product\_qty ('1'), product\_name ('Apple iPhone 7 Plus'), product\_desc ('64GB - Factory GSM Unlocked Smartphone Multi Colors'), product\_price ('800'), seller\_uname ('apple\_usa'), seller\_email ('apple@usa.com'), and product\_totalqty ('997'). The second document has an \_id of '581fd53f725bb81b04209966'. It contains fields like buyer\_uname ('princeofalltimes@gmail.com'), product\_id ('581fa53349052c2988de96f3'), product\_qty ('2'), product\_name ('Roadmaster Adventures 700 Bike'), product\_desc ('Mountain Bike 26inch - Best Performance'), product\_price ('200'), seller\_uname ('riotofrock'), seller\_email ('riotofrock@gmail.com'), and product\_totalqty ('10').

## Shopping cart collection

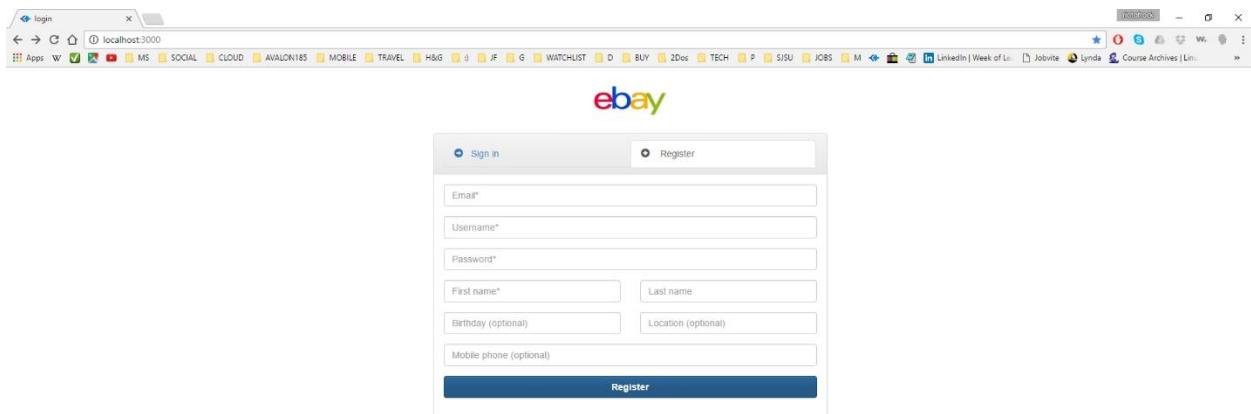
The screenshot shows the Robomongo interface with a connection to the 'ebay' database. The 'userdetails' collection is selected. A query `db.getCollection('userdetails').find({})` is run, returning two documents. The first document has an \_id of '581f98923ec04a131883b32d'. It contains fields like uname ('prince'), pword ('05346af66ab32c43e781bf1be00d9b8e'), fname ('Prince'), lname ('Jacob Chandy'), email ('princeofalltimes@gmail.com'), bday ('1991-01-05'), location ('San Jose'), contact ('6692927279'), lastlogin ('Sun Nov 06 2016 13:49:58 GMT-0800 (Pacific Standard Time)'), ccard\_info ('\*\*\*\*\*6543'), addressline1 ('1334 The Alameda'), addressline2 ('185'), addresscity ('SAN JOSE'), addressstate ('California'), and addresspin ('95126'). The second document has an \_id of '581fa44449052c2988de96ed'. It contains fields like uname ('riotofrock'), pword ('25bc938dfa9685e69f5c2a8af3574679'), fname ('Daby'), lname ('Shelly'), email ('riotofrock@gmail.com'), bday ('2016-11-06'), location ('Kottayam'), contact ('9611745326'), lastlogin ('Sun Nov 06 2016 13:44:46 GMT-0800 (Pacific Standard Time)'), ccard\_info ('\*\*\*\*\*3456'), addressline1 ('Kanikuzhy'), addressline2 ('Kottayam'), addresscity ('Kerala'), addressstate ('India'), and addresspin ('123456').

## User details collection

### III. RESULTS



**Image description:** login page (sign in options)



**Image description:** login page (sign up/register options)

The screenshot shows a web browser window with the URL `localhost:3000` in the address bar. The page is titled "ebay" and displays a registration form. The form fields are as follows:

- Email: `princeofalltimes@gmail.com`
- Username: `prince`
- Password: `*****`
- First Name: `Prince`
- Last Name: `Jacob Chandy`
- Date of Birth: `1991-01-05`
- City: `San Jose`
- Phone Number: `6692927279`

A green success message at the bottom of the form area reads: "Registration done! Sign in with your new credentials!" Below this message is a blue "Register" button.

**Image description:** user registration success

The screenshot shows a web browser window with the URL `localhost:3000` in the address bar. The page is titled "ebay" and displays a registration form. The form fields are identical to the successful registration above, except for the email field which contains `princeofalltimes@gmail.com`. A red error message at the bottom of the form area reads: "Username/Email already in use!" Below this message is a blue "Register" button.

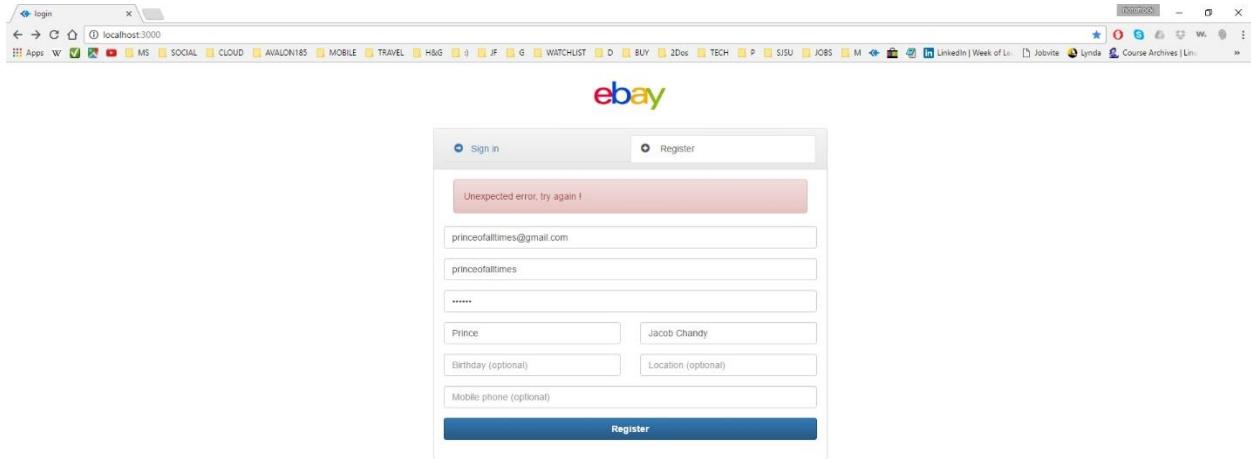
**Image description:** user registration error – username/email already in use

The screenshot shows a web browser window with the URL `localhost:3000`. The page is titled "ebay" and features a registration form. The form includes fields for Email\*, Username\*, Password\*, First name\*, Last name, Birthday (optional), Location (optional), and Mobile phone (optional). A red error message at the bottom states: "Type in the required(\*) fields and try again!". A blue "Register" button is at the bottom right.

**Image description:** user registration error – required fields should be filled (required fields are marked by \*)

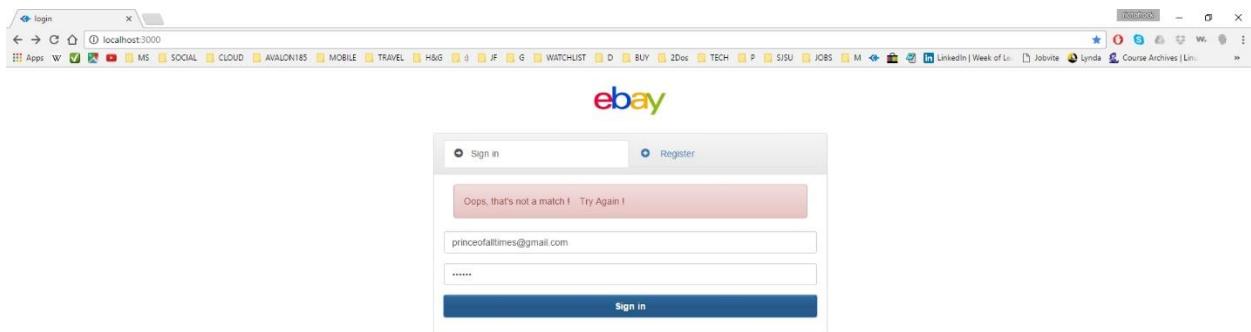
The screenshot shows a web browser window with the URL `localhost:3000`. The page is titled "ebay" and features a registration form. The form includes fields for Email\*, Username\*, Password\*, First name\*, Last name, Birthday (optional), Location (optional), and Mobile phone (optional). A red error message at the top states: "Server not responding at the time, try again!". The input fields contain sample data: Email is "princeoftimetimes@gmail.com", Username is "princeoftimetimes", Password is "\*\*\*\*\*", First name is "Prince", Last name is "Jacob Chandy", and the other optional fields are empty. A blue "Register" button is at the bottom right.

**Image description:** user registration error – server is not responding



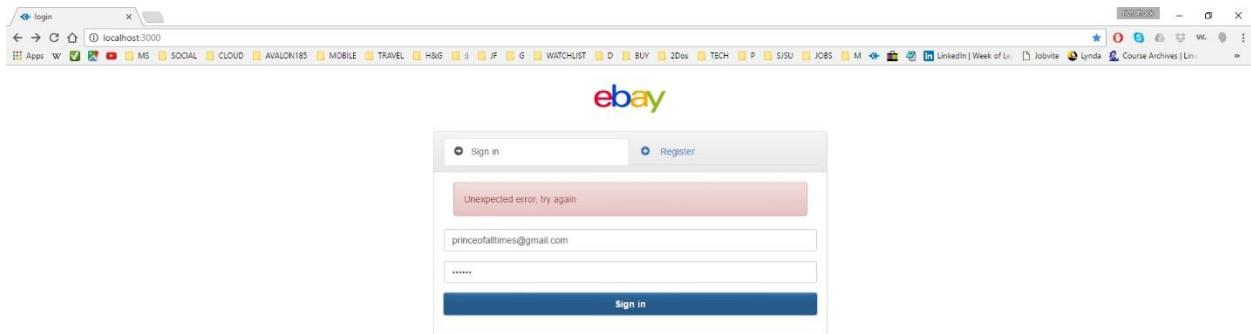
The screenshot shows a web browser window with the eBay logo at the top. Below it is a registration form. At the top of the form, there is a red error message box containing the text "Unexpected error, try again!". The registration fields include: Email (princeofalltimes@gmail.com), Username (princeofalltimes), Password (\*\*\*\*\*), First Name (Prince), Last Name (Jacob Chandy), Birthday (optional), Location (optional), and Mobile phone (optional). A blue "Register" button is at the bottom.

**Image description:** user registration error – in case of any other error during registration

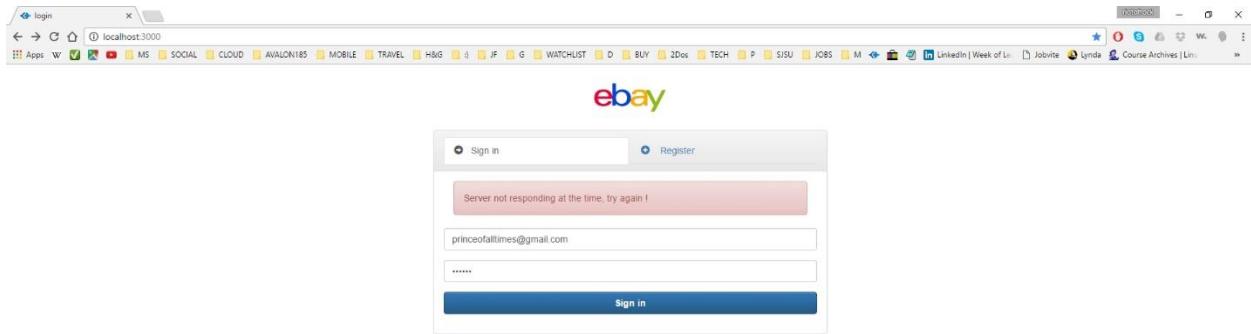


The screenshot shows a web browser window with the eBay logo at the top. Below it is a login form. At the top of the form, there is a red error message box containing the text "Oops, that's not a match! Try Again!". The login fields include: Email (princeofalltimes@gmail.com) and Password (\*\*\*\*\*). A blue "Sign in" button is at the bottom.

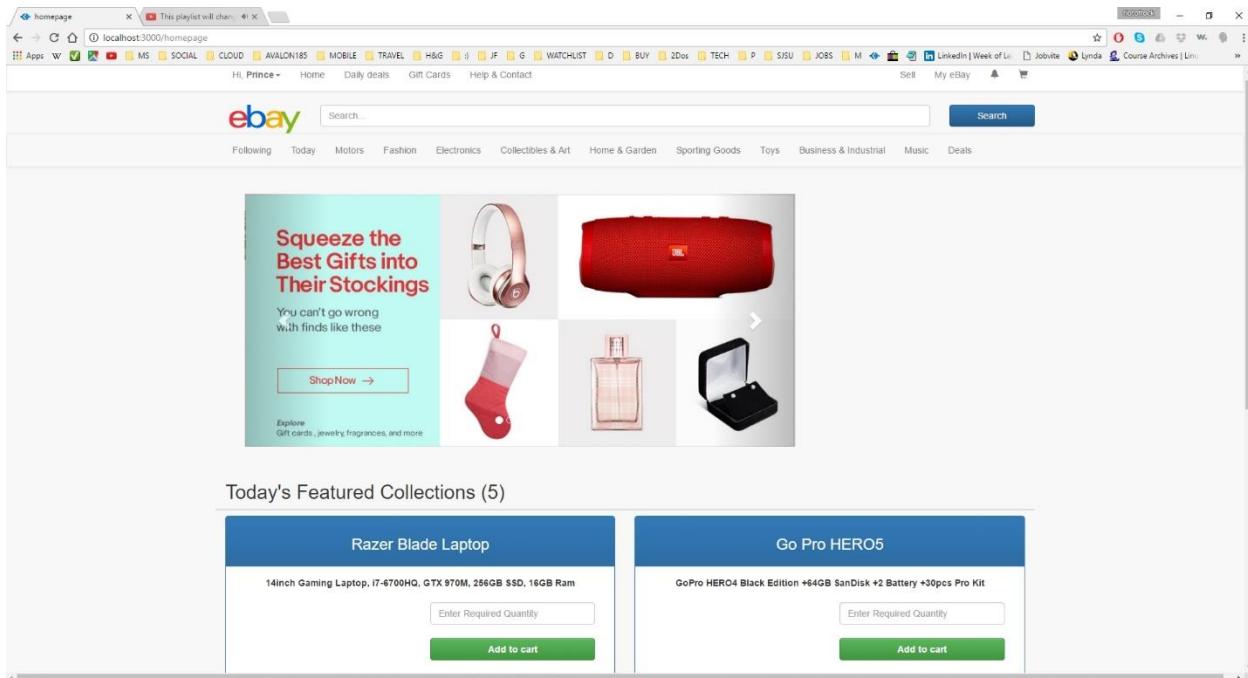
**Image description:** user login error – username and password is not matching (or not found in database)



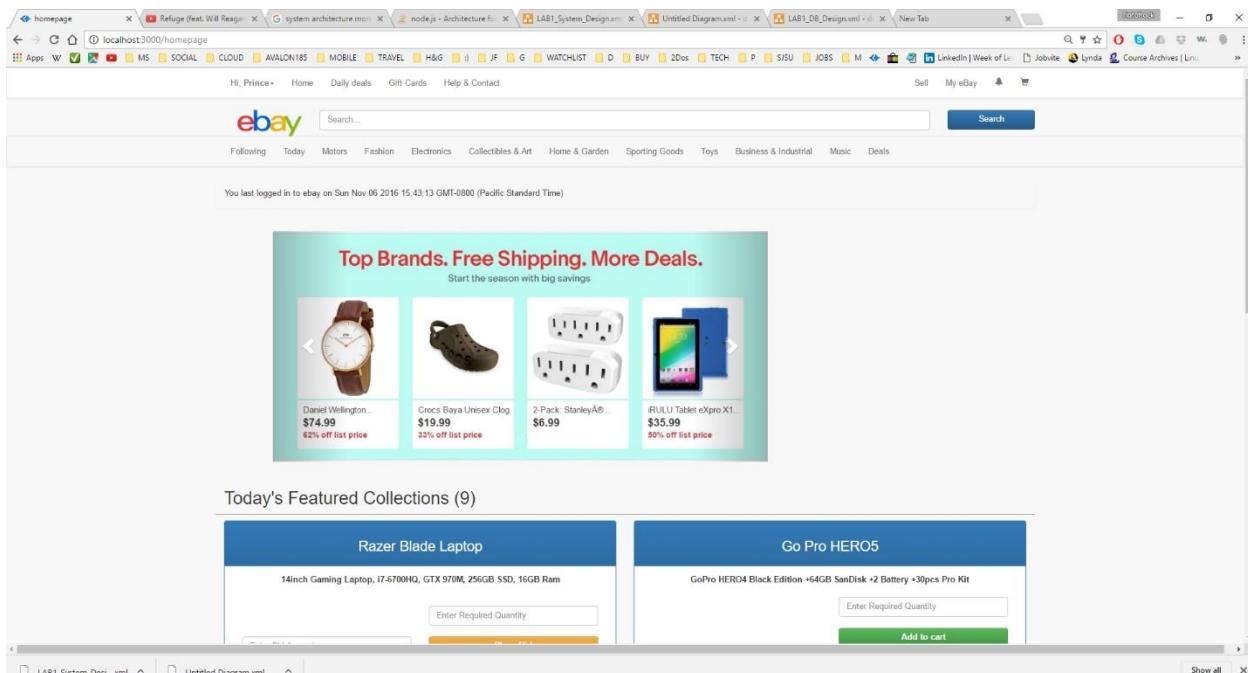
**Image description:** user login error – server is not responding/server down



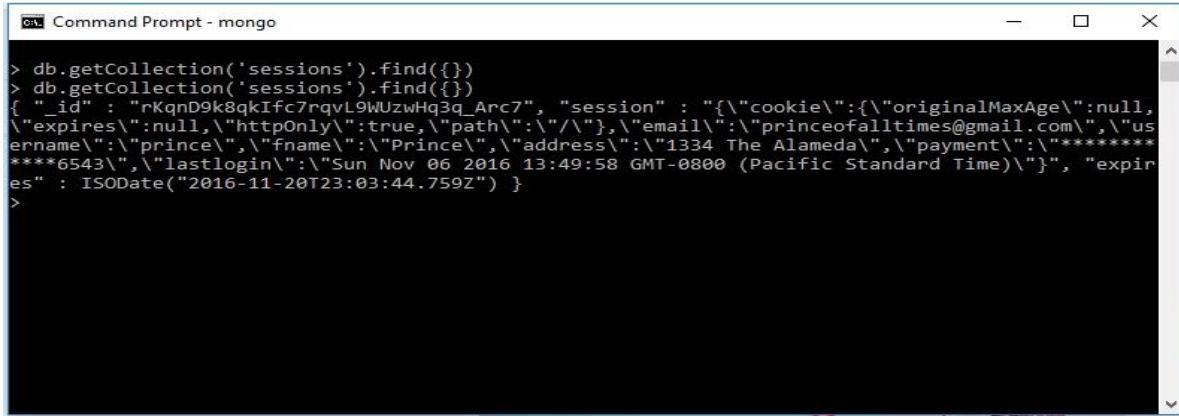
**Image description:** user login error – in case of any other error while signing in



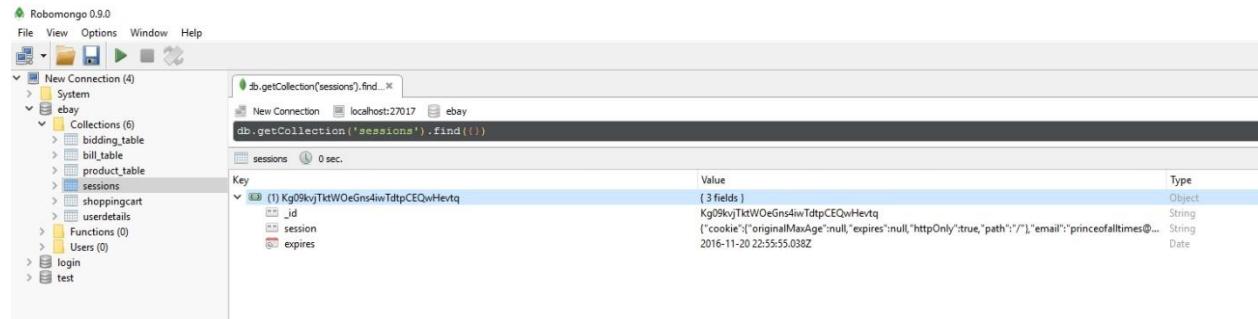
**Image description:** home page - for first time login



**Image description:** home page - for successive login (with last time logged in information)



```
Command Prompt - mongo
> db.getCollection('sessions').find({})
> db.getCollection('sessions').find({})
{
  "_id" : "rKqnD9k8qkIfc7rqvL9WUzwHq3q_Arc7",
  "session" : {
    "cookie" : {
      "originalMaxAge" : null,
      "expires" : null,
      "httpOnly" : true,
      "path" : "/",
      "email" : "princeofalltimes@gmail.com",
      "username" : "prince",
      "fname" : "Prince",
      "address" : "1334 The Alameda",
      "payment" : "*****6543",
      "lastlogin" : "Sun Nov 06 2016 13:49:58 GMT-0800 (Pacific Standard Time)"
    },
    "expires" : ISODate("2016-11-20T23:03:44.759Z")
  }
}
```

**Image description:** session information stored in mongoDB (1)


Robomongo 0.9.0

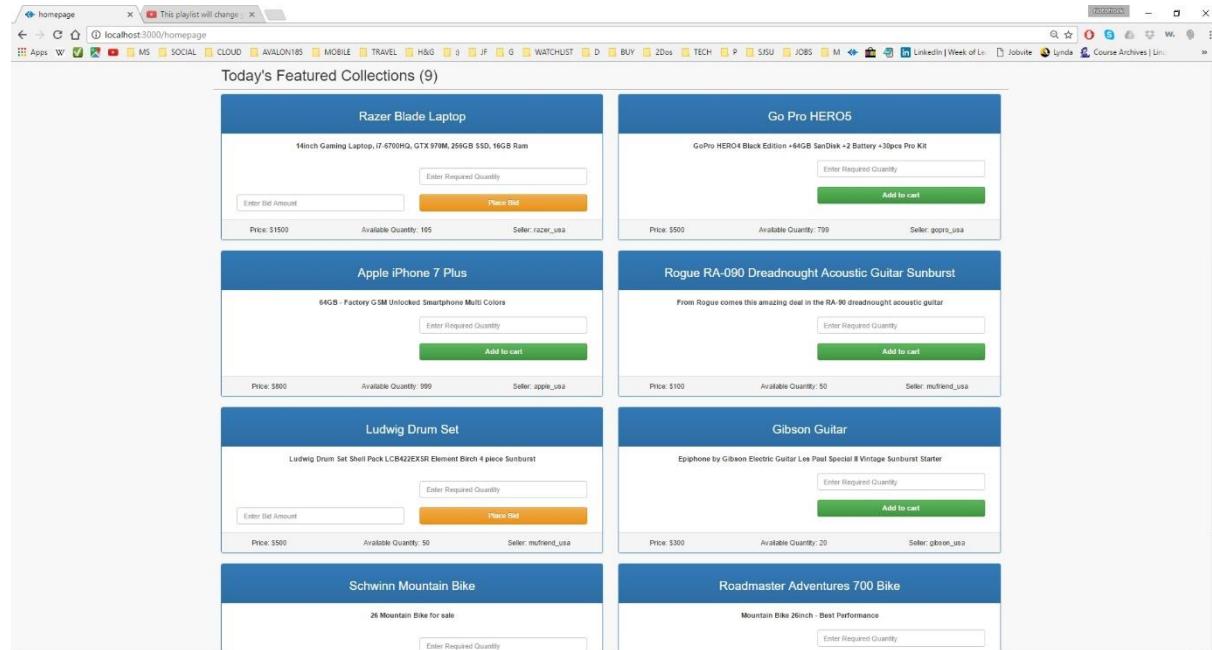
File View Options Window Help

New Connection (4) | System | ebay | Collections (6) | sessions | Key | Value | Type

- bidding\_table
- bill\_table
- product\_table
- sessions
- shoppingcart
- userdetails

db.getCollection('sessions').find(...)

Key	Value	Type
(1) Kg09kjTktWOeGns4iwTdtpCEQwHevtq	<code>{ _id : Kg09kjTktWOeGns4iwTdtpCEQwHevtq, session : { cookie : { originalMaxAge : null, expires : null, httpOnly : true, path : '/' }, email : 'princeofalltimes@gmail.com' }, expires : ISODate('2016-11-20 22:55:55.038Z' ) }</code>	Object

**Image description:** session information stored in mongoDB (2)


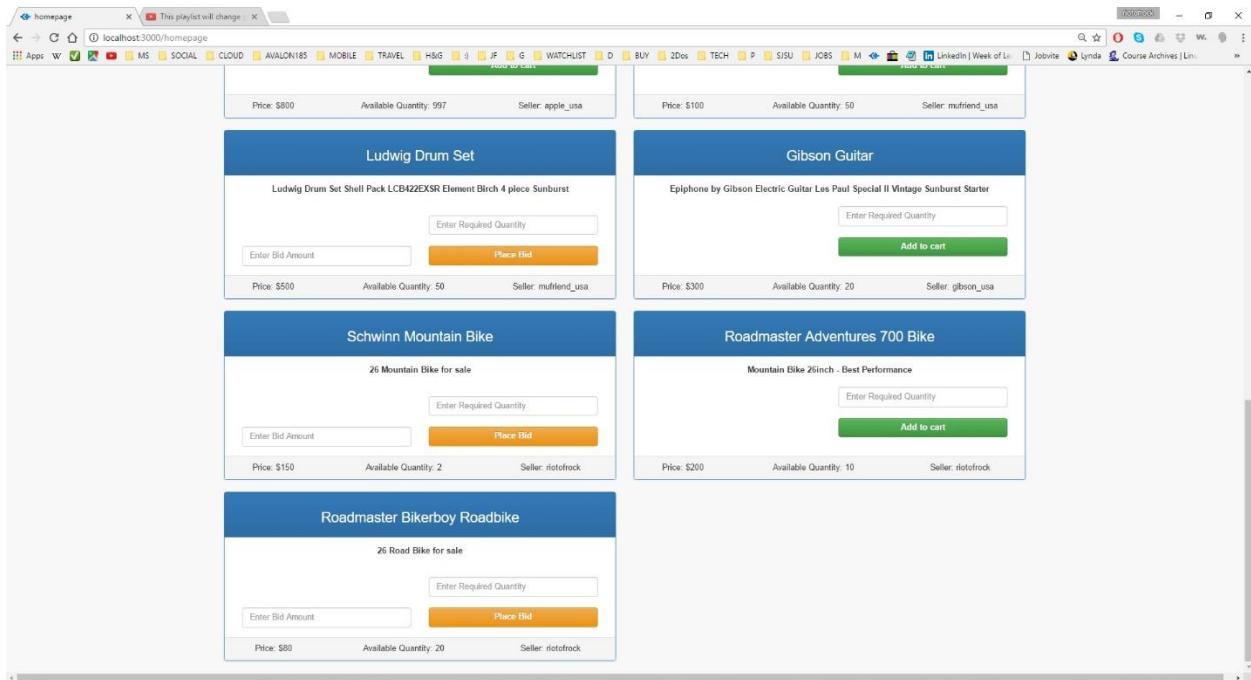
homepage This playlist will change

localhost:3000/homepage

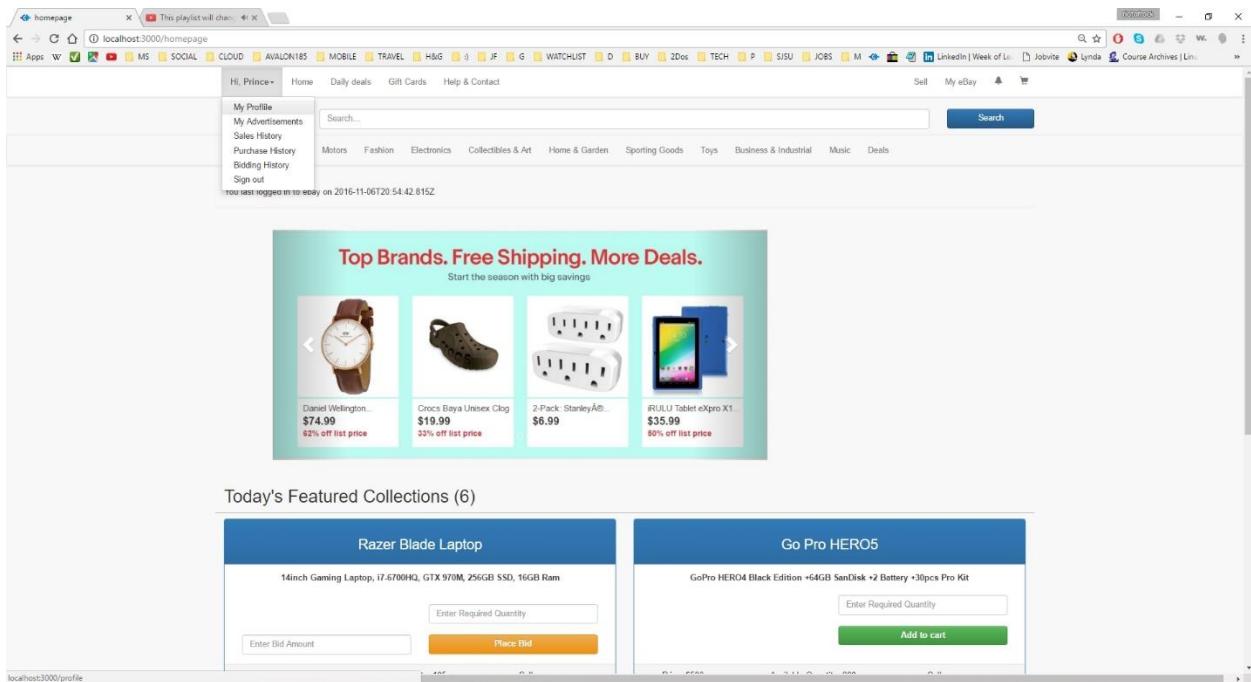
Today's Featured Collections (9)

<b>Razer Blade Laptop</b> 14inch Gaming Laptop, i7-6700HQ, GTX 970M, 256GB SSD, 16GB Ram Price: \$1500 Available Quantity: 195 Seller: razer_usa	<b>Go Pro HERO5</b> GoPro HERO4 Black Edition +64GB SanDisk +2 Battery +30pc Pro Kit Price: \$500 Available Quantity: 799 Seller: goptech_usa
<b>Apple iPhone 7 Plus</b> 64GB - Factory GSM Unlocked Smartphone Multi Colors Price: \$600 Available Quantity: 999 Seller: apple_usa	<b>Rogue RA-090 Dreadnought Acoustic Guitar Sunburst</b> From Rogue comes this amazing deal in the RA-90 dreadnought acoustic guitar Price: \$100 Available Quantity: 50 Seller: mufriend_usa
<b>Ludwig Drum Set</b> Ludwig Drum Set Shell Pack LC542EXSR Element Birch 4 piece Sunburst Price: \$500 Available Quantity: 50 Seller: mufriend_usa	<b>Gibson Guitar</b> Epiphone by Gibson Electric Guitar Les Paul Special II Vintage Sunburst Starter Price: \$300 Available Quantity: 20 Seller: gibson_usa
<b>Schwinn Mountain Bike</b> 26 Mountain Bike for sale Enter Required Quantity	<b>Roadmaster Adventures 700 Bike</b> Mountain Bike 26inch - Best Performance Enter Required Quantity

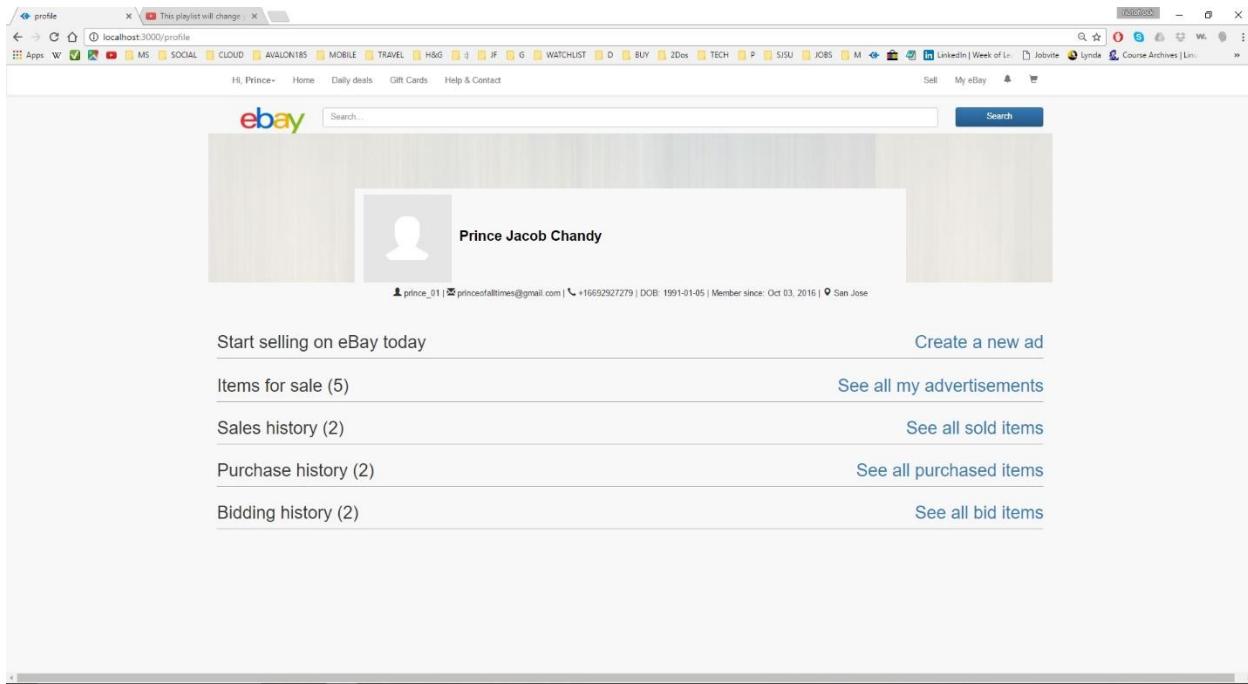
**Image description:** home page – advertisements posted by other users are listed (1); products sold out (available quantity=0) and products whose bid end time is over won't be listed



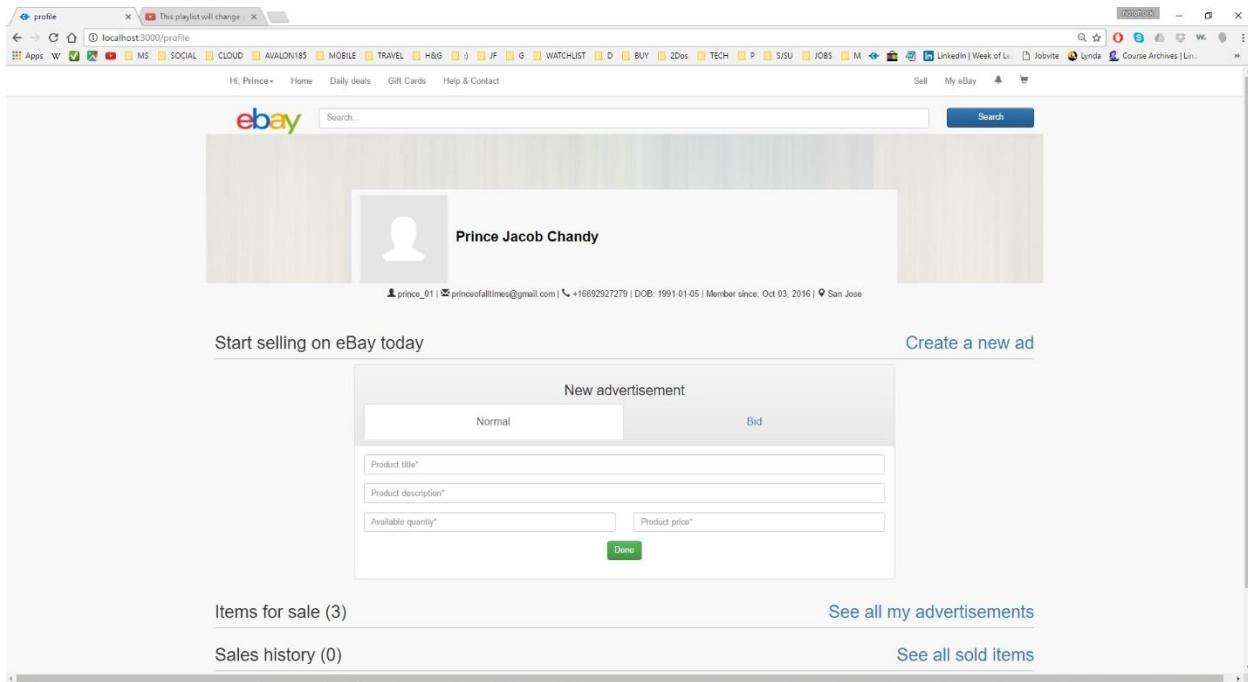
**Image description:** home page – advertisements posted by other users are listed (2); products sold out (available quantity=0) and products whose bid end time is over won't be listed



**Image description:** home page – drop down options listed in the main header (functionalities)



**Image description:** profile page – displays profile information and purchase/sale history; also allows user to create his own advertisements (normal and bid)



**Image description:** profile page – create a new normal advertisement (1) - required fields are marked by \* and an ad can be created only if all the required information is filled

The screenshot shows a browser window with the URL `localhost:3000/profile`. The eBay logo is at the top left, followed by a search bar and a 'Search' button. Below the search bar is a user profile section for 'Prince Jacob Chandy'. The profile includes a placeholder profile picture, the name 'Prince Jacob Chandy', and a member since date of 'Oct 03, 2016'. Below the profile is a message 'Start selling on eBay today' and a 'Create a new ad' button. A central modal dialog is open titled 'New advertisement' with two tabs: 'Normal' (selected) and 'Bid'. The 'Normal' tab contains fields for 'Product title\*' (Razer Electra Headphones), 'Used Razer Electra Headphones (1 year old)', 'Available quantity\*' (1), and 'Bidding price\*' (44). A 'Done' button is at the bottom right of the modal. At the bottom of the page, there are links for 'Items for sale (3)', 'See all my advertisements', 'Sales history (0)', and 'See all sold items'.

**Image description:** profile page – create a new normal advertisement (2)

This screenshot is identical to the one above, showing the same eBay profile page and the 'New advertisement' dialog. The difference is in the 'Normal' tab of the dialog, which now has four required fields marked with asterisks (\*): 'Product title\*', 'Product description\*', 'Available quantity\*', and 'Bidding price\*'. The 'Done' button remains at the bottom right. The rest of the page, including the sidebar links and the bottom navigation, is identical to the first screenshot.

**Image description:** profile page – create a new bidding advertisement (1) - required fields are marked by \* and an ad can be created only if all the required information is filled

The screenshot shows a browser window with the URL `localhost:3000/profile`. The page displays a user profile for "Prince Jacob Chandy". Below the profile picture, there is a message: "This playlist will change". The main content area features a search bar with "ebay" typed in. A large button labeled "Start selling on eBay today" is on the left, and a link "Create a new ad" is on the right. In the center, a modal dialog titled "New advertisement" is open. It has two tabs: "Normal" (selected) and "Bid". Under "Normal", there are input fields for "Takamine Guitar" and "Takamine Jasmine Guitar", both set to quantity 2 at a price of \$100. A dropdown menu shows quantity 4 is selected. A "Done" button is at the bottom right of the dialog. Below the dialog, a link "Items for sale (4)" is on the left and "See all my advertisements" is on the right.

**Image description:** profile page – create a new bidding advertisement (2)

The screenshot shows the same eBay profile page as the previous one, but now it lists five items for sale under the heading "Items for sale (5)". The items are displayed in a grid:

- Roadmaster Granite Peak Bike**: 26inch Steel Frame - Roadmaster Granite Peak Trails Black or Blue. Price/Item: \$100, Quantity: 10, Seller: prince.
- Acoustic Guitar - Used**: Granada Guitar 6 string. Price/Item: \$100, Quantity: 1, Seller: prince.
- Acer Laptop**: Acer Aspire E 17. Price/Item: \$900, Quantity: 1, Seller: prince.
- Razer Electra Headphones**: Used Razer Electra Headphones (1 year old). Price/Item: \$44, Quantity: 1, Seller: prince.
- Takamine Guitar**: Takamine Jasmine Guitar. Price/Item: \$100, Quantity: 2, Seller: prince.

**Image description:** profile page – see all advertisements created by the user

The screenshot shows a web browser window with a grid of six advertisement cards. Each card has a title, a brief description, a quantity input field, a 'Place Bid' button, and an 'Add to cart' button.

- Gibson Guitar:** Epiphone by Gibson Electric Guitar Les Paul Special II Vintage Sunburst Starter. Price: \$100, Available Quantity: 50, Seller: mufriend\_usa.
- Acoustic Guitar - Used:** Granada Guitar 6 string. Price: \$500, Available Quantity: 50, Seller: mufriend\_usa.
- Acer Laptop:** Acer Aspire E 17. Price: \$300, Available Quantity: 20, Seller: gibson\_usa.
- Razer Electra Headphones:** Used Razer Electra Headphones (1 year old). Price: \$100, Available Quantity: 1, Seller: prince.
- Takamine Guitar:** Takamine Jasmine Guitar. Price: \$100, Available Quantity: 2, Seller: prince.
- Gibson Guitar (repeated):** Epiphone by Gibson Electric Guitar Les Paul Special II Vintage Sunburst Starter. Price: \$100, Available Quantity: 50, Seller: mufriend\_usa.

**Image description:** home page – newly created advertisements created as seen in another user's homepage

The screenshot shows a web browser window displaying a user profile. The profile belongs to 'Prince Jacob Chandy'. The page includes sections for selling, advertising, and viewing sales history.

**Sales History:**

- Roadmaster Granite Peak Bike:** Bill ID: 581fa4ab409c2c2988de96ef, Total Price: \$100, Total Items: 1, Buyer Info: riotofrock@gmail.com. Price/Item: \$100, Quantity: 1, Paid using: \*\*\*\*\*3456.
- Razer Electra Headphones:** Bill ID: 581fa4ab409c2c2988de96f1, Total Price: \$44, Total Items: 1, Buyer Info: riotofrock@gmail.com. Price/Item: \$44, Quantity: 1, Paid using: \*\*\*\*\*3456.

**Purchase History:** See all purchased items

**Bidding History:** See all bid items

**Image description:** profile page – sales history of current user's products with buyer details and billing information

This screenshot shows a user's eBay profile page. At the top, there is a navigation bar with links like 'profile', 'localhost:3000/profile', 'Hi, Prince', 'Home', 'Daily deals', 'Gift Cards', 'Help & Contact', 'Sell', 'My eBay', and a search bar. Below the navigation is the eBay logo and a search bar.

The main content area displays the user's profile picture and name, 'Prince Jacob Chandy'. Below this, it shows the user's member information: 'prince\_01 | princeofalltimes@gmail.com | +16892927279 | DOB: 1991-01-05 | Member since: Oct 03, 2016 | San Jose'.

There are several sections for managing items:

- Start selling on eBay today**
- Create a new ad**
- Items for sale (5)**
- Sales history (2)**
- Purchase history (2)**
- Bidding history (2)**
- See all my advertisements**
- See all sold items**
- See all purchased items**
- See all bid items**

Two purchase history items are listed in a grid:

Product Name	Bill ID	Total Price	Quantity	Paid using
Go Pro HERO5	581fad249052c2988de96f0	\$500	1	*****6543
Apple iPhone 7 Plus	581fb462ba8bf2ddc2b4892	\$1600	1	*****6543

**Image description:** profile page – purchase history of current user with seller details and billing information

This screenshot shows a user's eBay profile page, similar to the one above but with different purchase history items.

The main content area displays the user's profile picture and name, 'Prince Jacob Chandy'. Below this, it shows the user's member information: 'prince\_01 | princeofalltimes@gmail.com | +16892927279 | DOB: 1991-01-05 | Member since: Oct 03, 2016 | San Jose'.

There are several sections for managing items:

- Start selling on eBay today**
- Create a new ad**
- Items for sale (5)**
- Sales history (2)**
- Purchase history (2)**
- Bidding history (2)**
- See all my advertisements**
- See all sold items**
- See all purchased items**
- See all bid items**

Two bidding history items are listed in a grid:

Product Name	26 Road Bike for sale	26 Mountain Bike for sale
Roadmaster Bikerboy Roadbike	Bid ends at: 2016-11-10T21:49:43.574Z Bid Amount: \$90	Bid ends at: 2016-11-10T21:47:58.043Z Bid Amount: \$160
Schwinn Mountain Bike	Quantity: 1 Seller: riotfrock@gmail.com	Quantity: 1 Seller: riotfrock@gmail.com

**Image description:** profile page – bidding history of current user with seller details and bidding information; list all the products bid by the user which is active

This screenshot shows a user's profile page with sections for items for sale, sales history, purchase history, and bidding history.

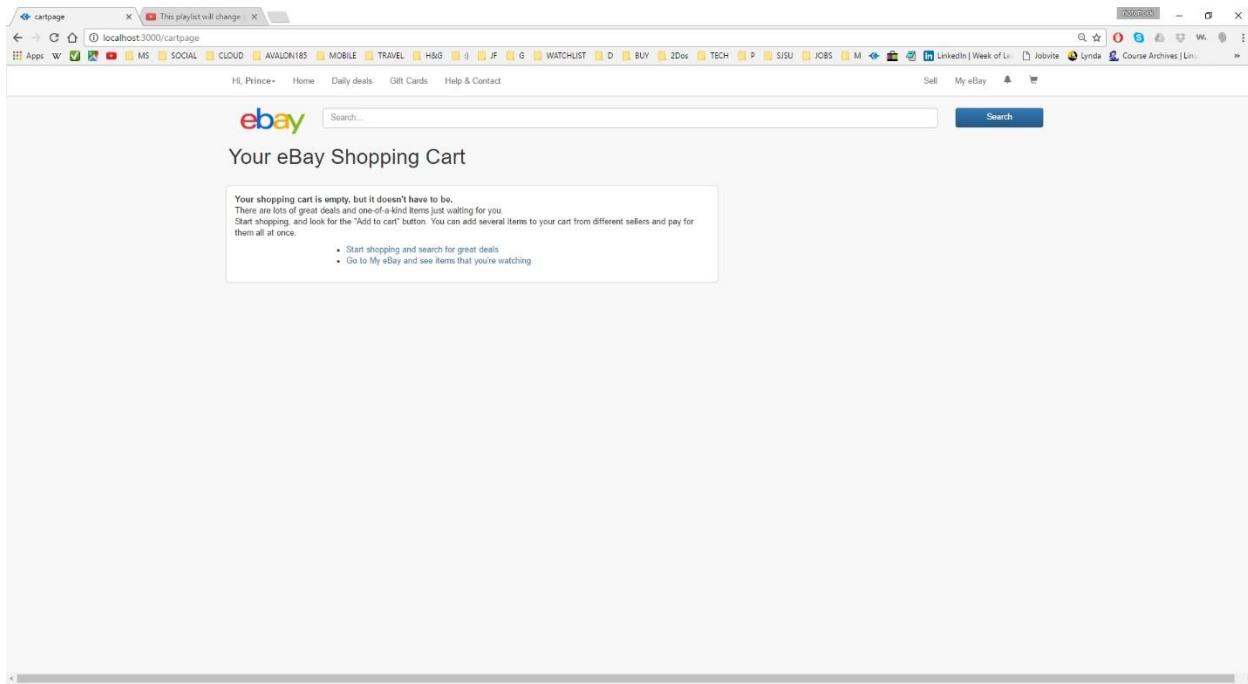
- Items for sale (5)**: Shows two items: "Roadmaster Granite Peak Bike" and "Razer Electra Headphones".
- Sales history (2)**: Shows two sales: "Roadmaster Granite Peak Bike" and "Razer Electra Headphones".
- Purchase history (2)**: Shows two purchases: "Go Pro HERO5" and "Apple iPhone 7 Plus".
- Bidding history (2)**: Shows two bids: "Roadmaster Bikerboy Roadbike" and "Schwinn Mountain Bike".

**Image description:** profile page – sales history, purchase history, bidding history

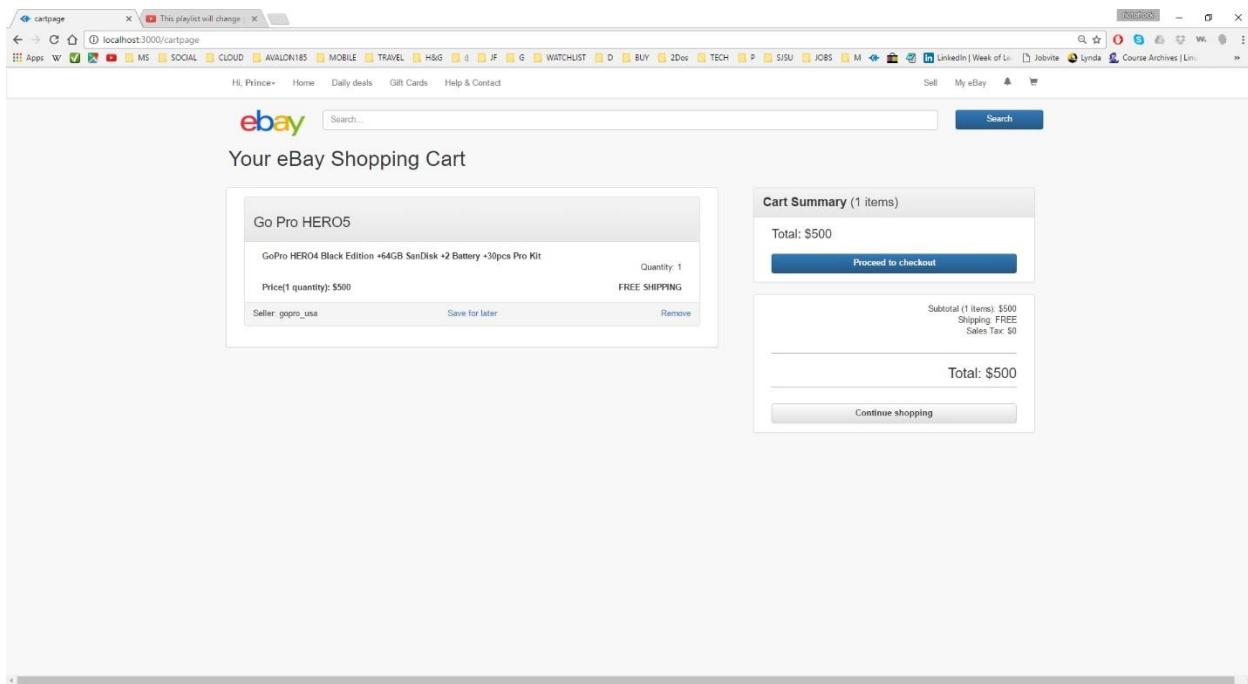
This screenshot shows a home page with a sidebar and a main content area.

- Today's Featured Collections (9)**: Shows a collection of items including headphones, a speaker, and a perfume bottle.
- Razer Blade Laptop**: Details: 14inch Gaming Laptop, i7-6700HQ, GTX 970M, 256GB SSD, 16GB Ram. Price: \$1500. Available Quantity: 105. Seller: razer\_usa.
- Go Pro HERO5**: Details: GoPro HERO4 Black Edition +64GB SanDisk +2 Battery +30pcs Pro Kit. Price: \$500. Available Quantity: 800. Seller: gopro\_usa.
- Apple iPhone 7 Plus**
- Rogue RA-090 Dreadnought Acoustic Guitar Sunburst**

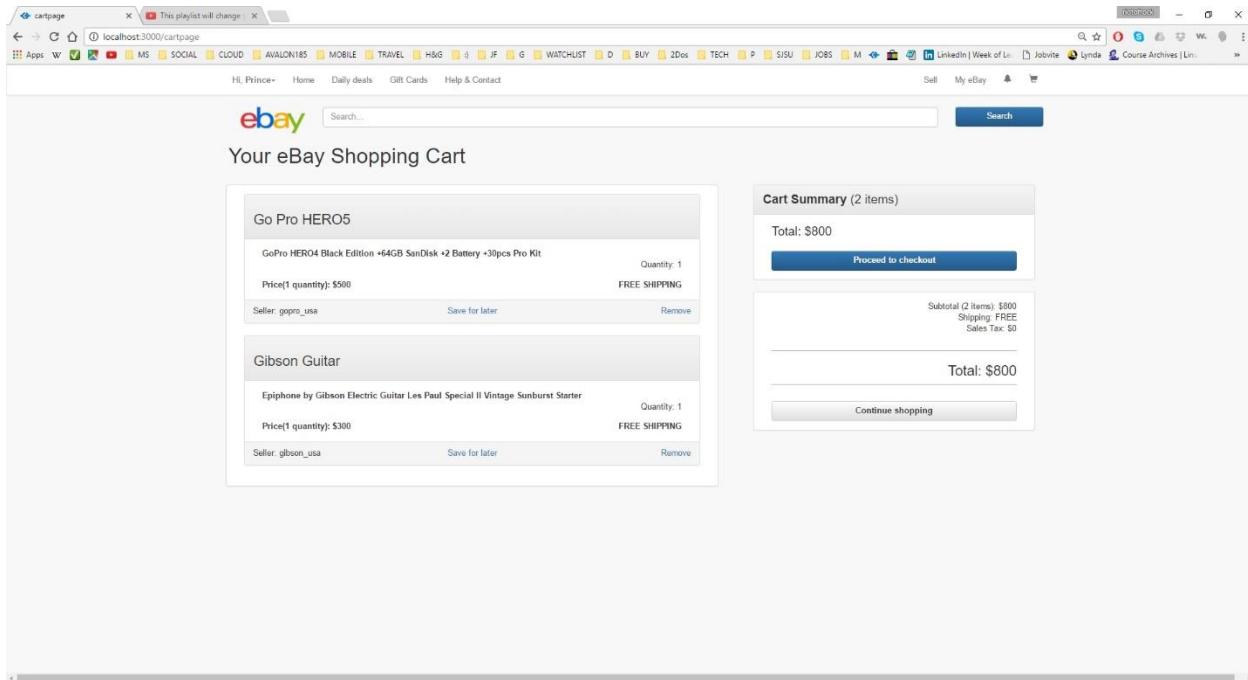
**Image description:** home page – buying a normal product by entering required quantity and clicking add to cart; add to cart button will proceed to cart only if the entered quantity is less than available quantity



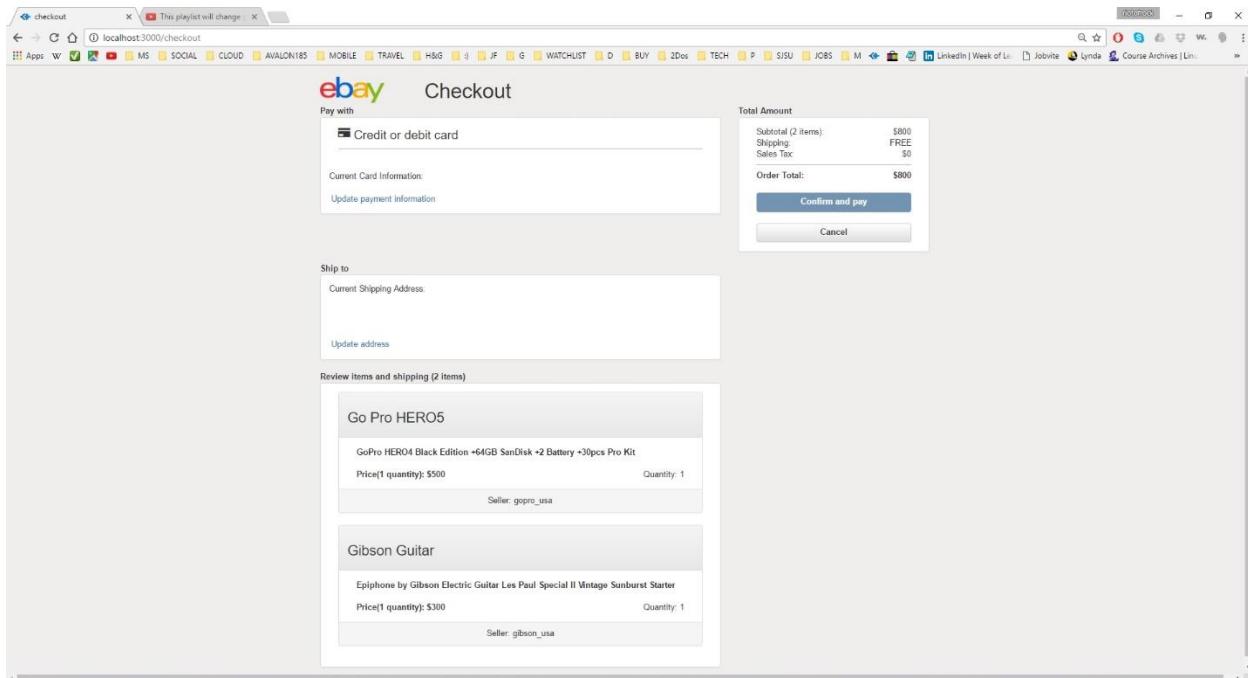
**Image description:** shopping cart page – when empty; Items in shopping cart when added will be added to session as well as database so that user gets his shopping cart information when he returns later after logout



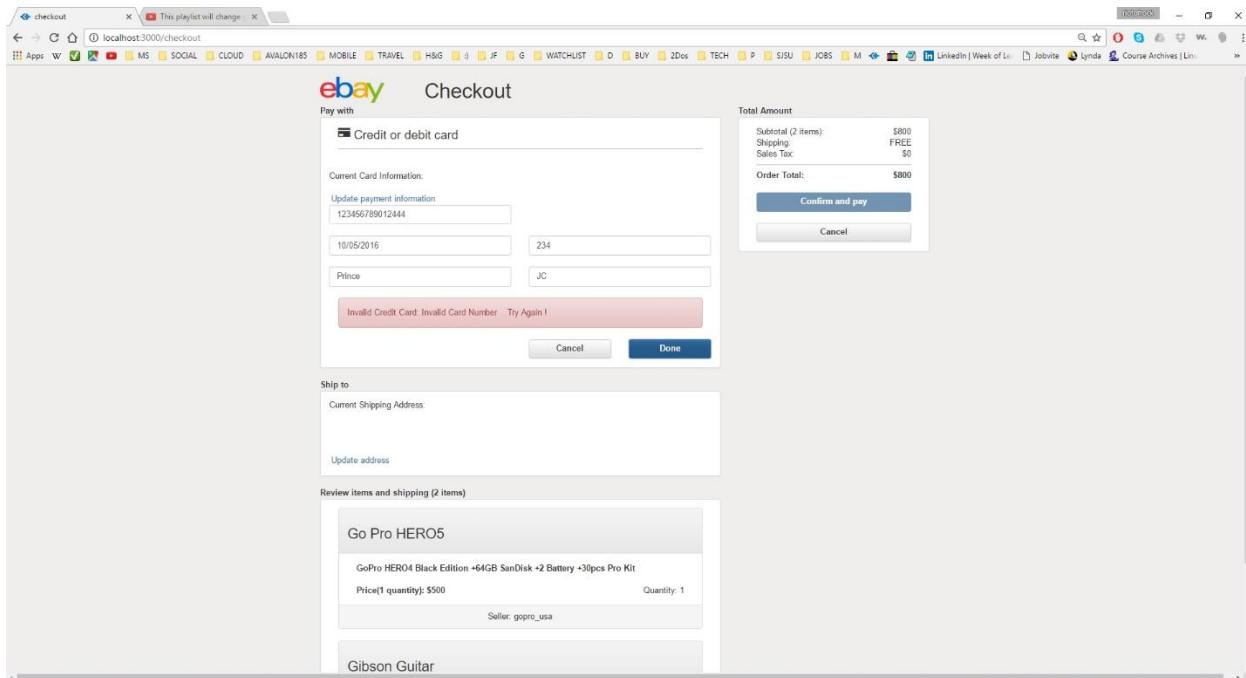
**Image description:** shopping cart page - all the items added by the user from homepage is listed here. Pressing add to cart from home page routes to shopping cart page. Can access this page also by clicking at the cart icon at the extreme top left



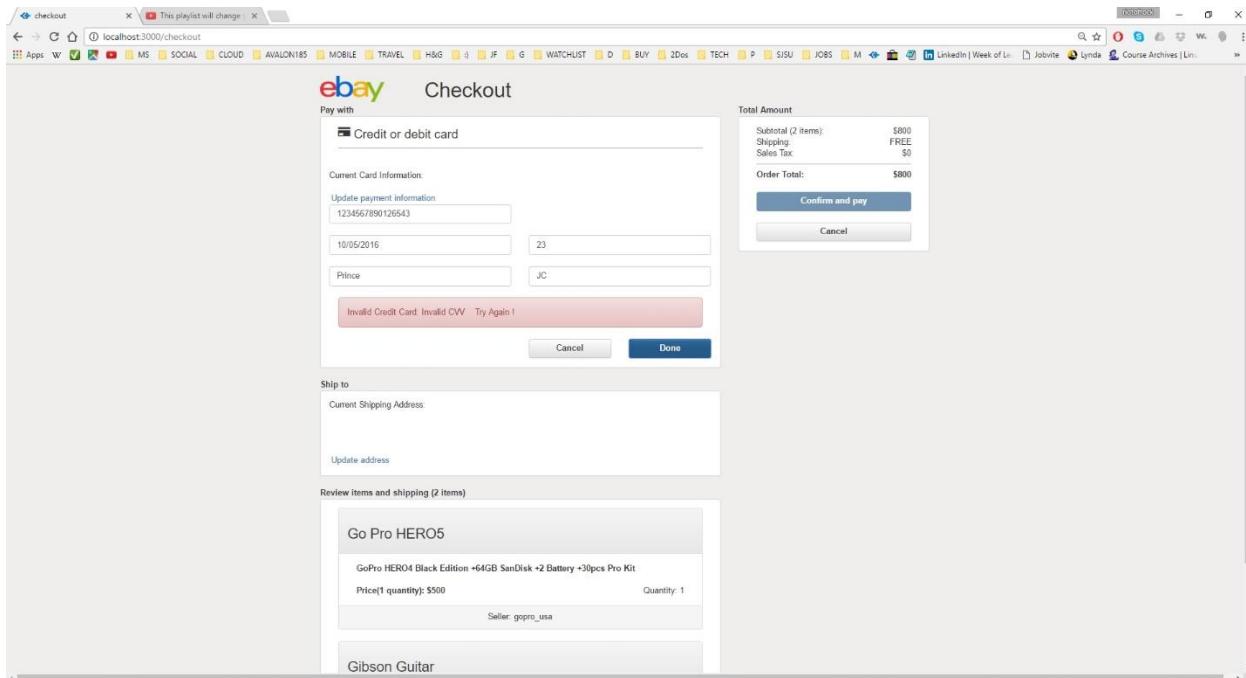
**Image description:** shopping cart page – with multiple items and total amount, items details; Note that delete option is provided for all items to remove them from the cart. Continue shopping routes back to the homepage



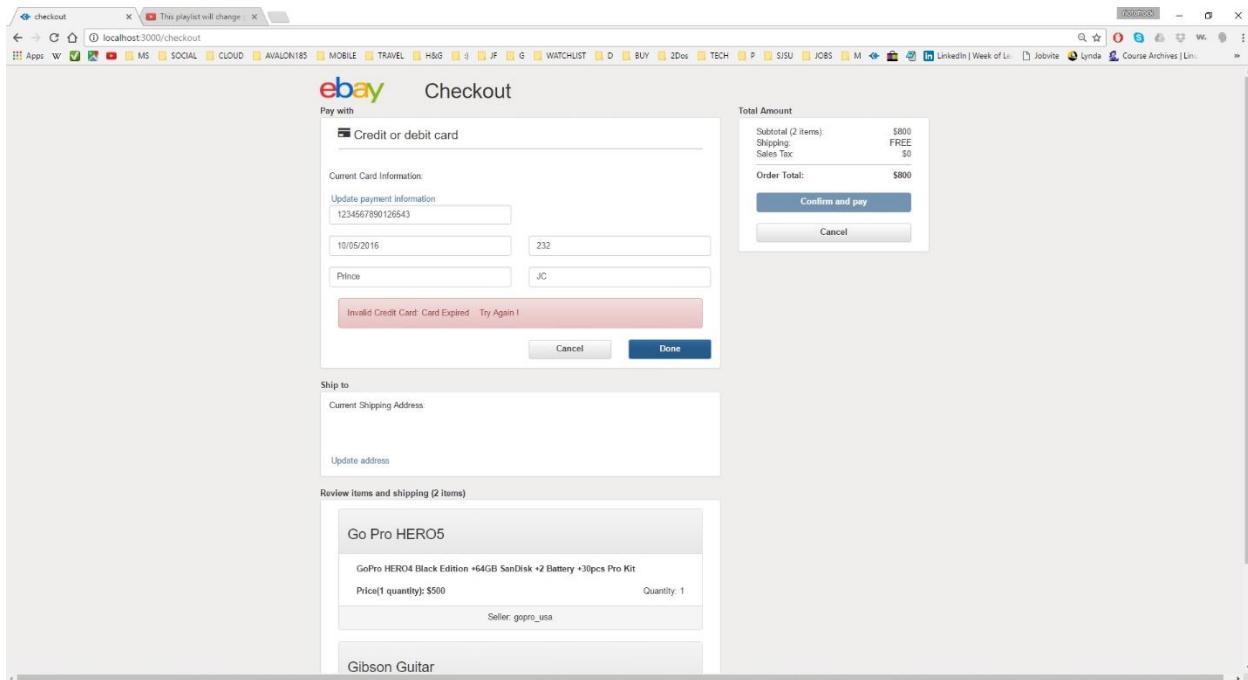
**Image description:** checkout page – user can provide (add/update) payment information and shipping address information. Confirm and pay button will be active only after entering valid creditcard information and shipping address



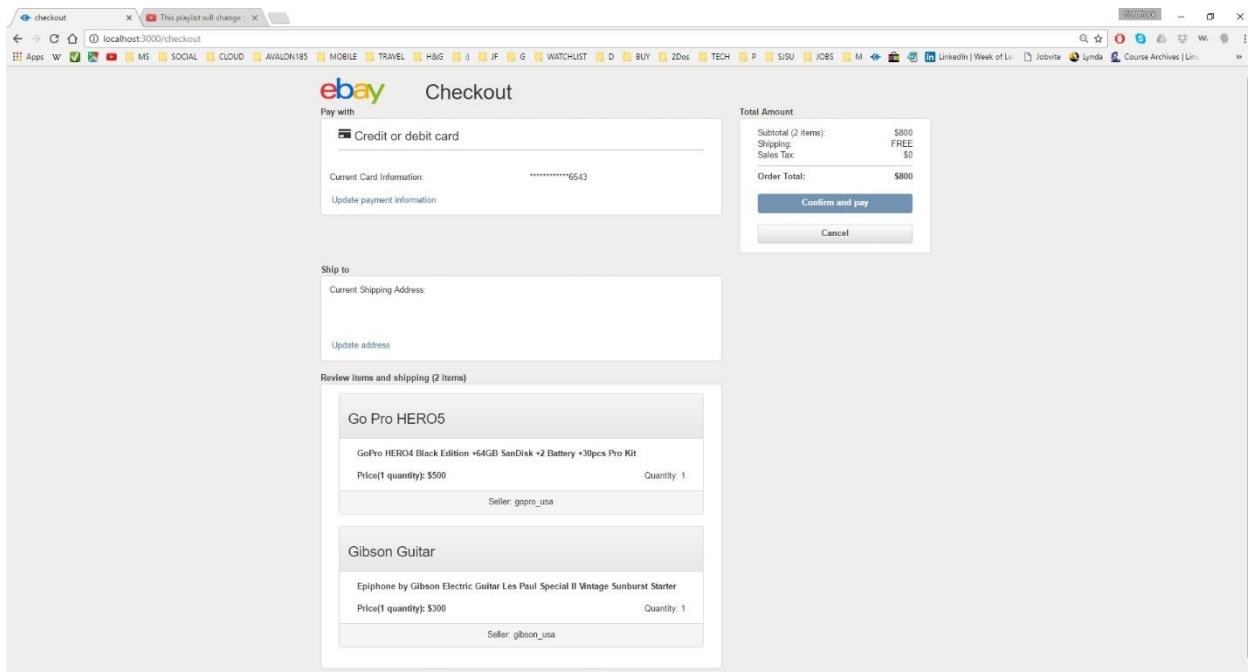
**Image description:** checkout page – payment/credit card validation (1)



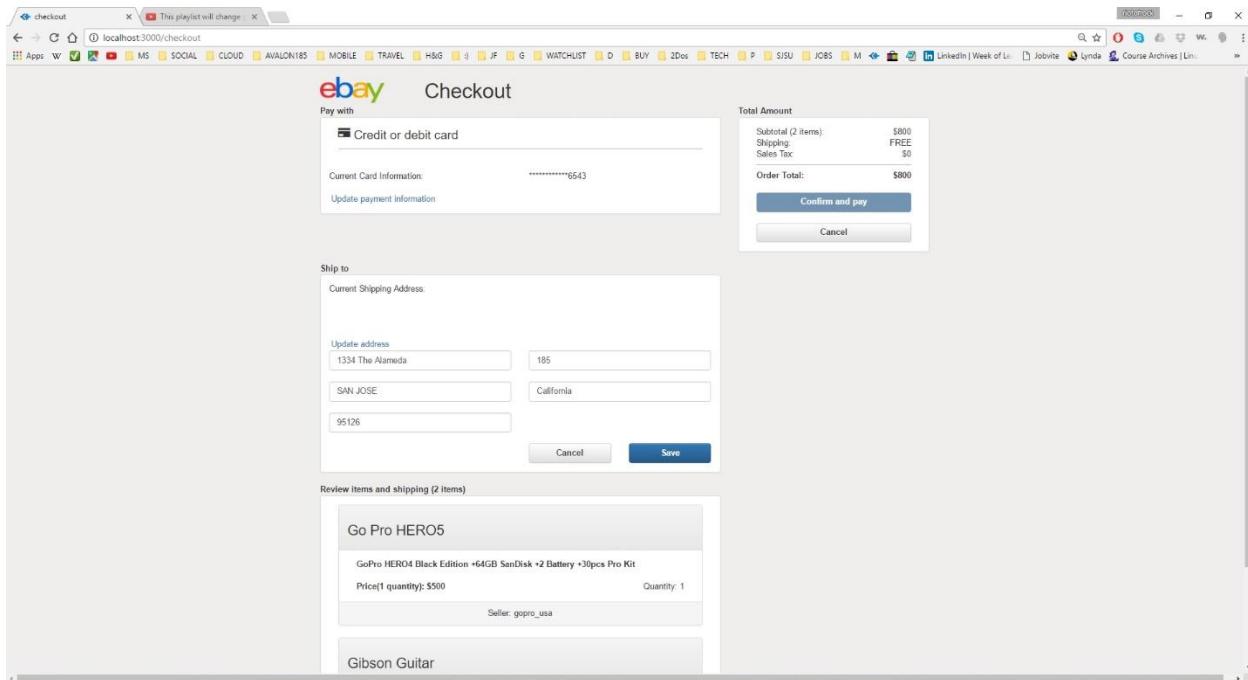
**Image description:** checkout page – payment/credit card validation (2)



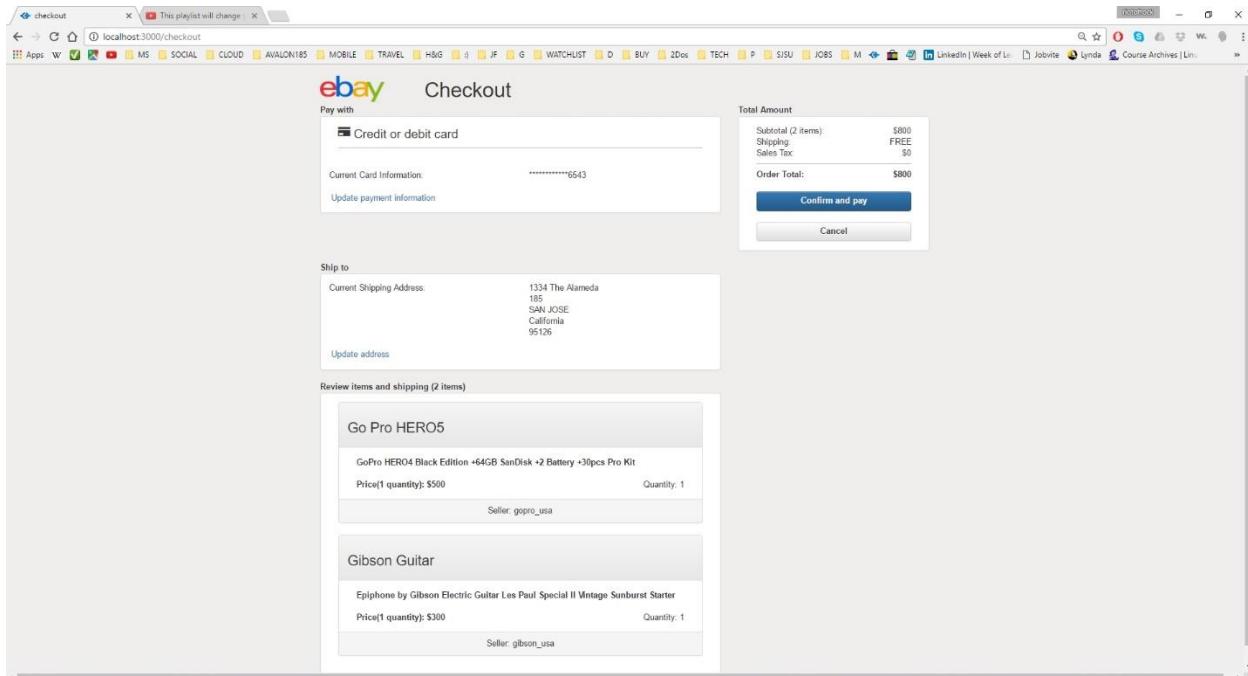
**Image description:** checkout page – payment/credit card validation (3)



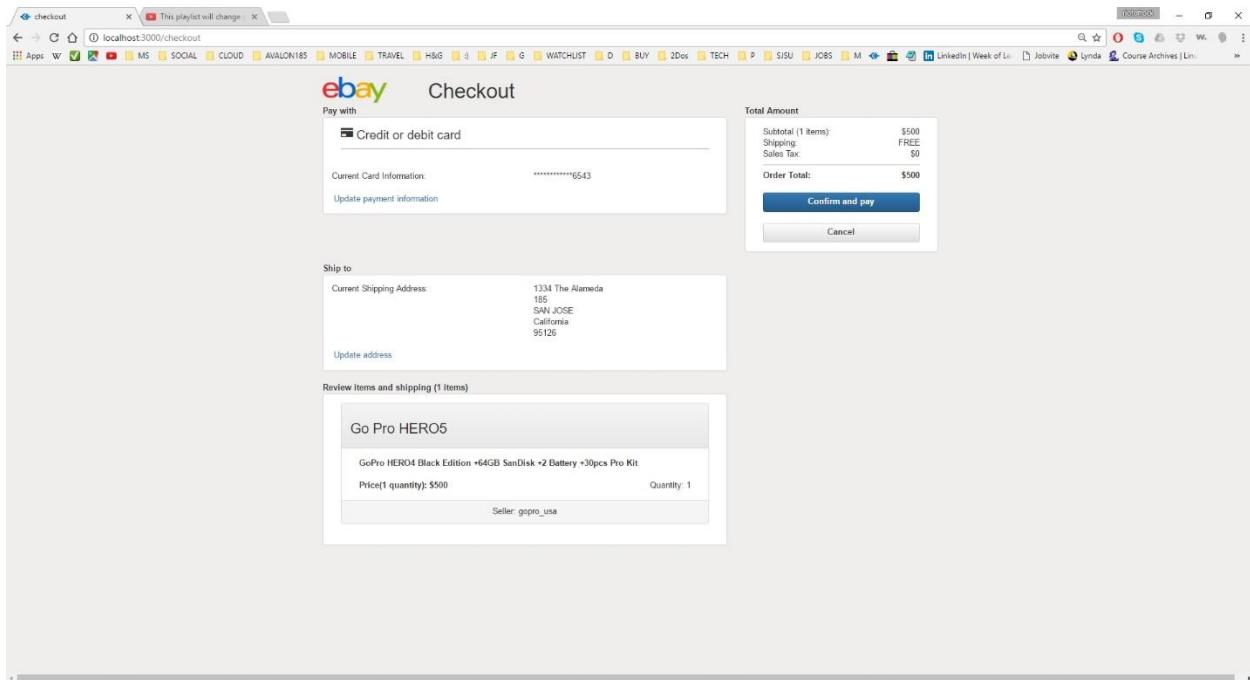
**Image description:** checkout page – payment/credit card validation success and information have been updated in database



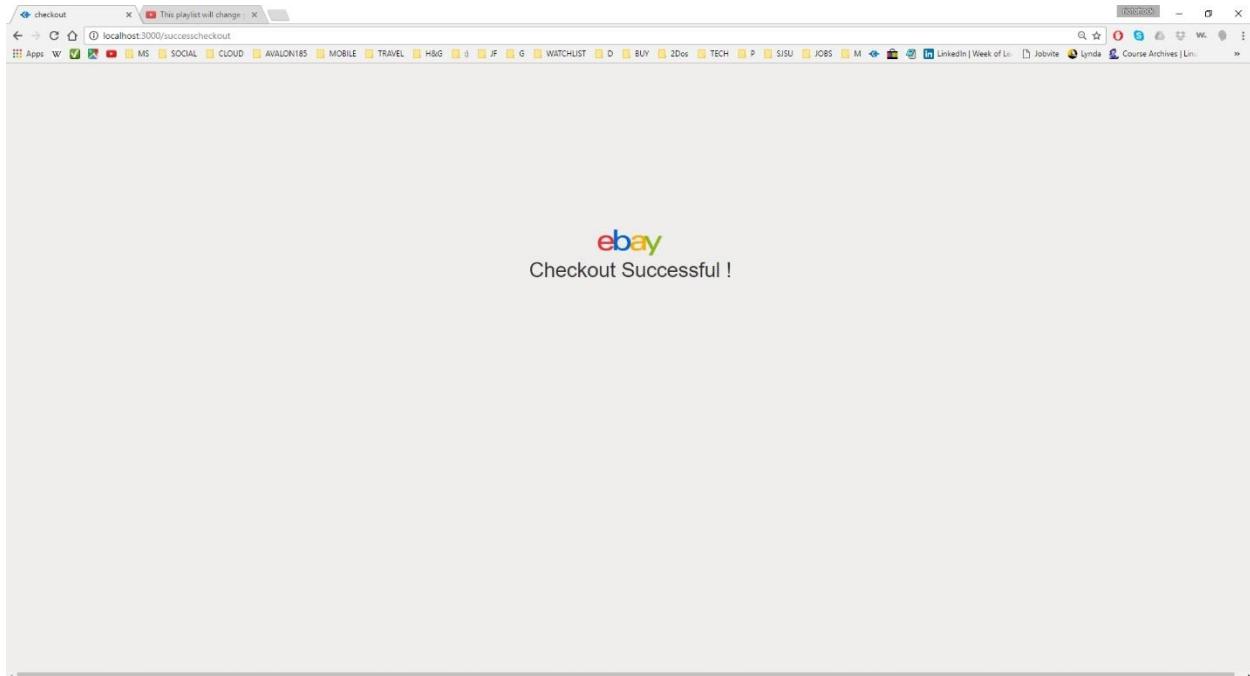
**Image description:** checkout page – shipping address addition/update



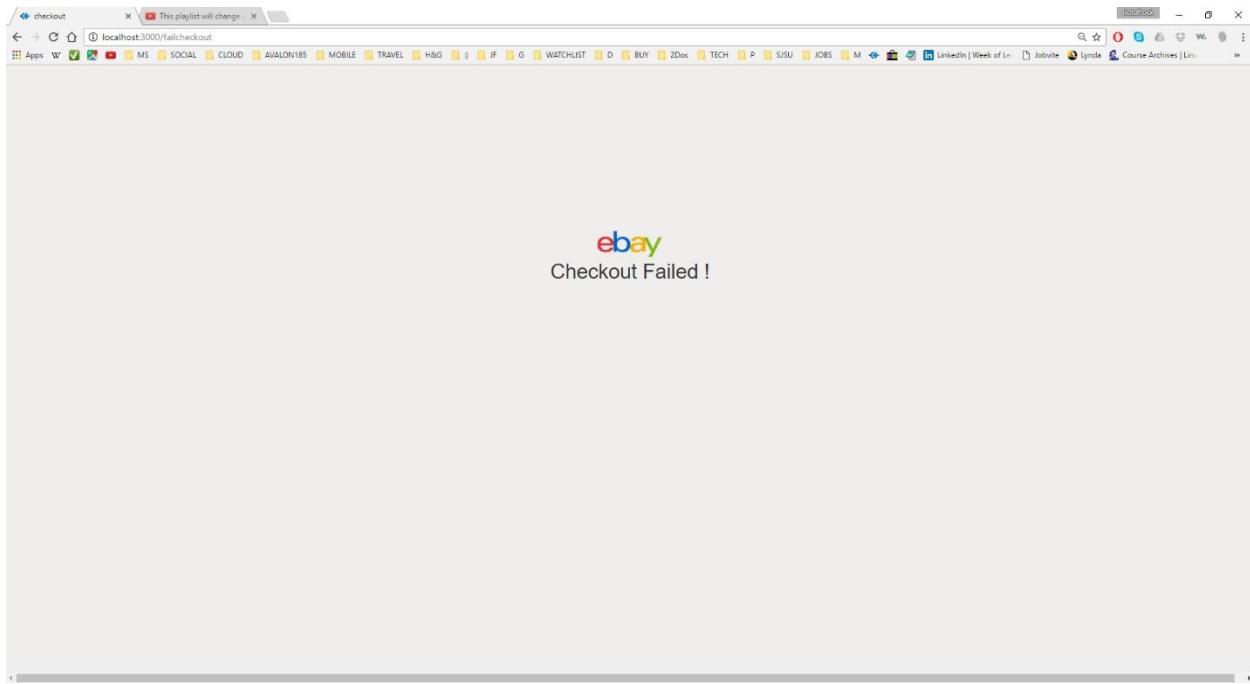
**Image description:** checkout page – confirm and pay button active since now we have valid payment and shipping information; checkout checks one final time in the database where the product and quantity in the cart is available and proceed to successful checkout page if available, or else to fail checkout page



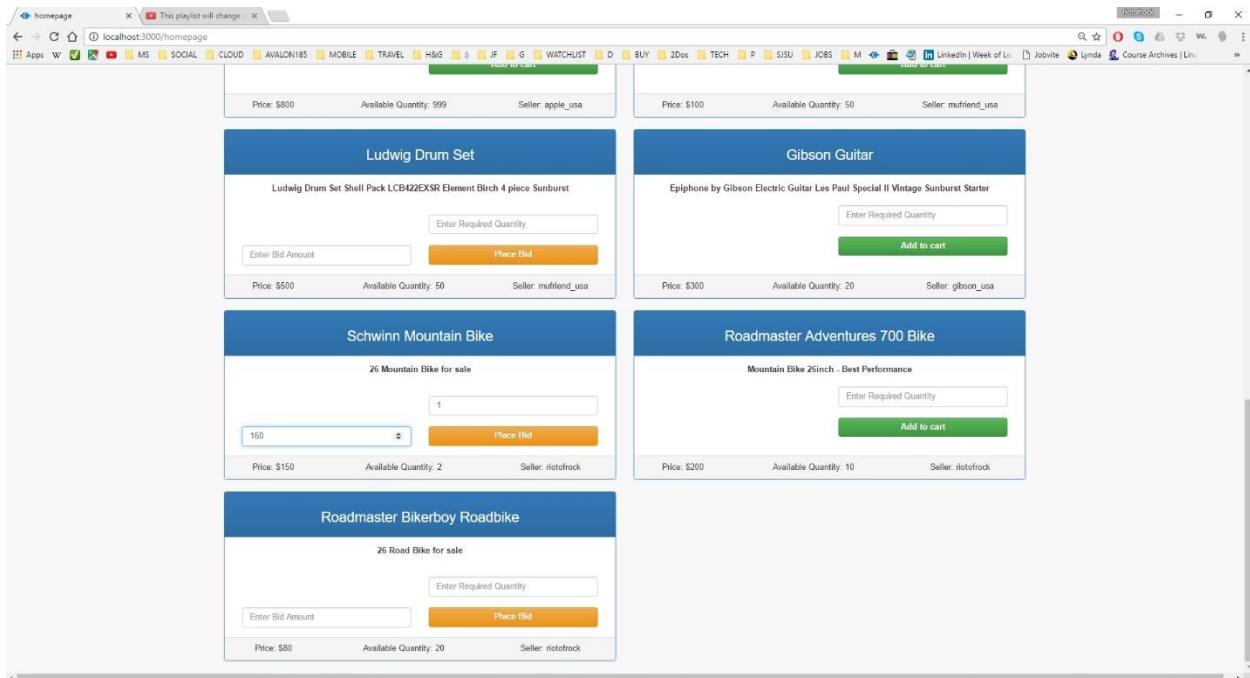
**Image description:** checkout page with single item in cart



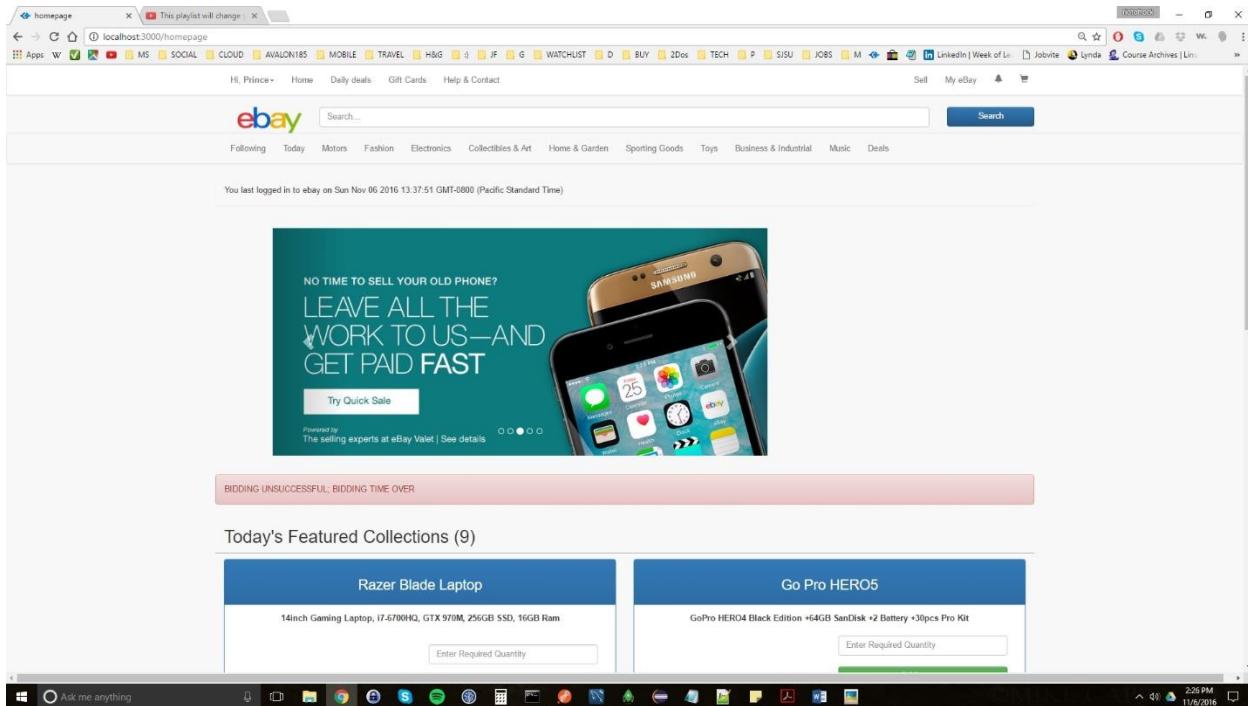
**Image description:** checkout successful page; can return to main page by clicking on ebay image



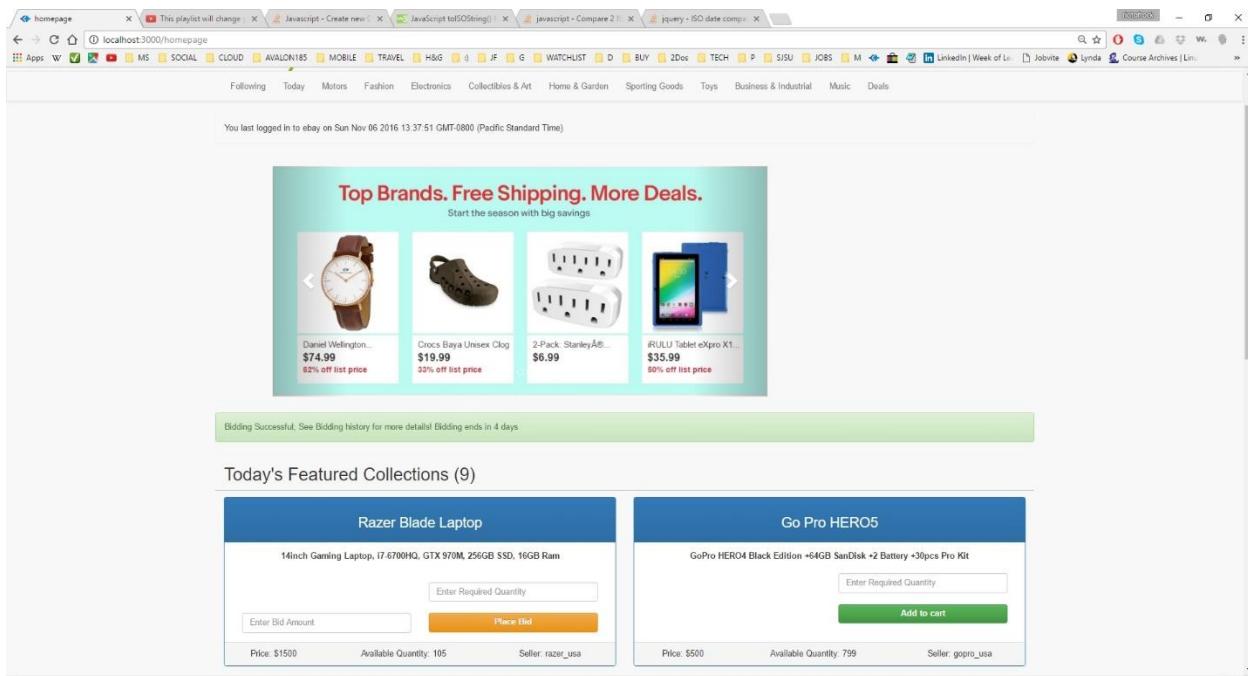
**Image description:** checkout failure page; can return to main page by clicking on ebay image



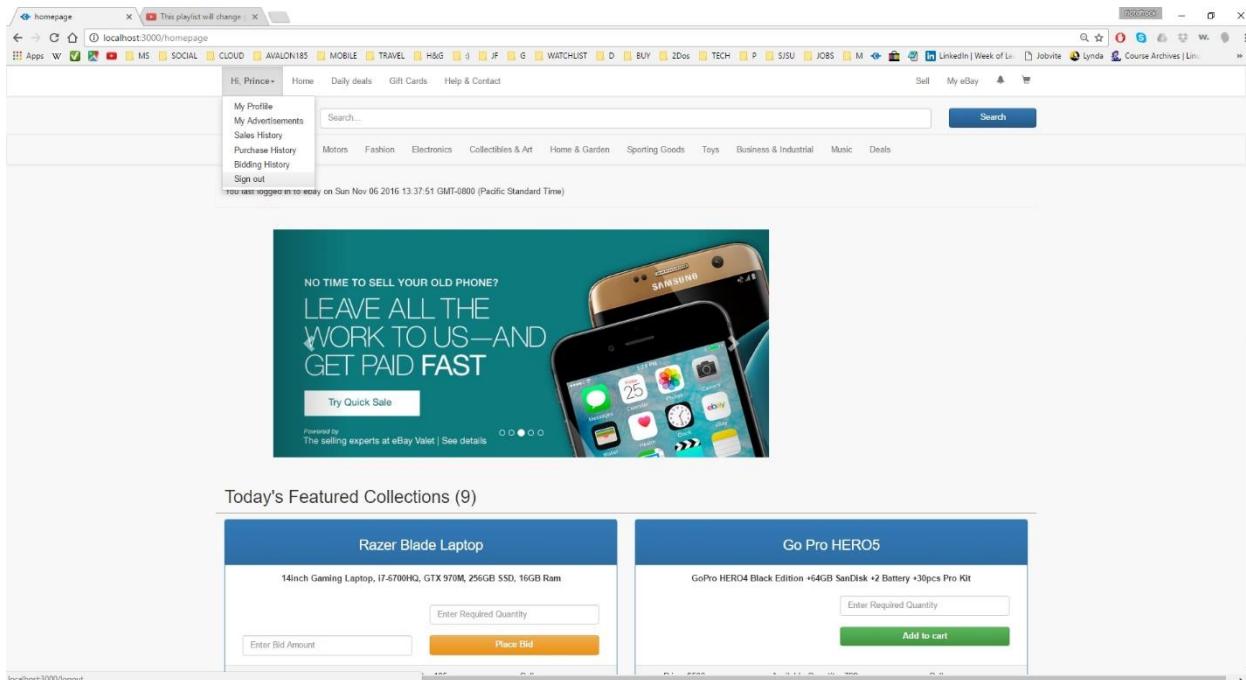
**Image description:** home page – bidding an item; user can bid an item by entering bid amount and the required quantity. Bidding will be allowed if the required quantity is more than the available quantity and the bid amount is more than the current product price displayed.



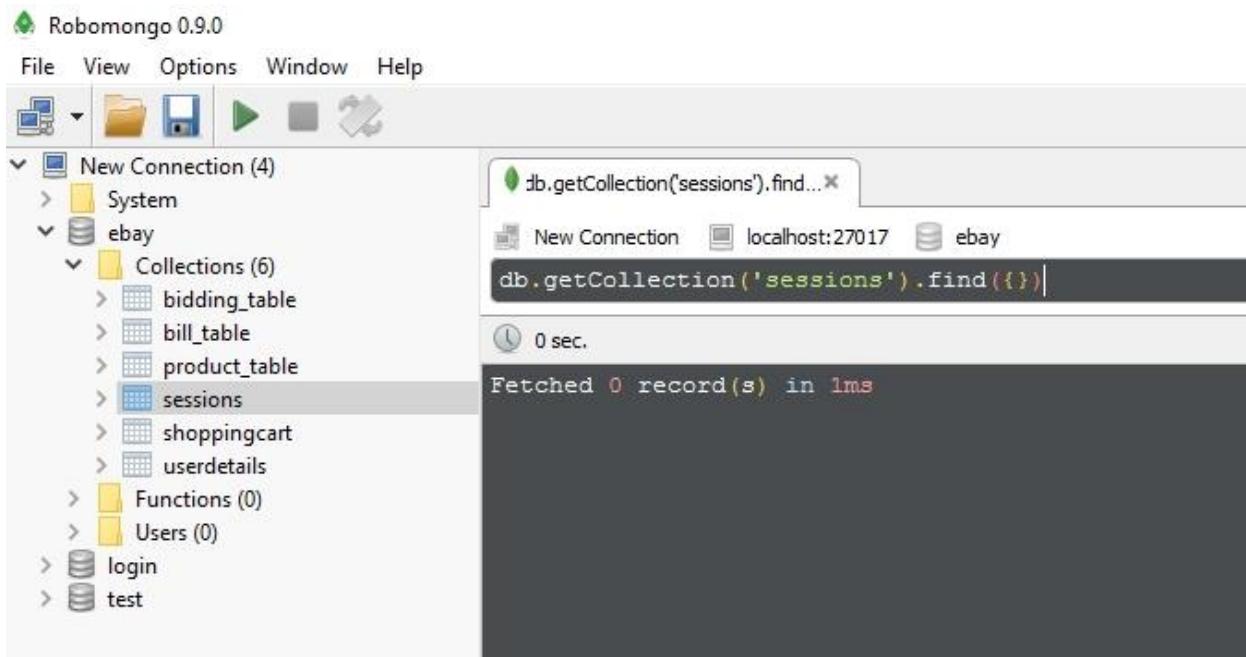
**Image description:** home page – bidding unsuccessful prompt will be shown when user try to bid a product whose bidding period is over



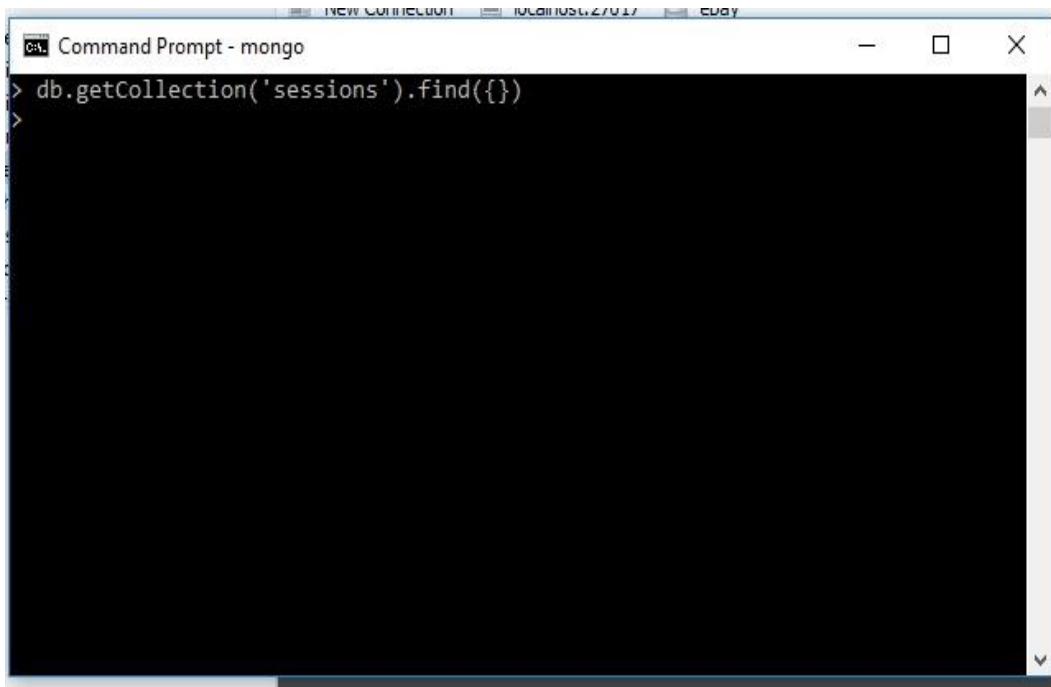
**Image description:** home page – bidding successful prompt will be shown to the user with the bidding timer information (time left till the bidding ends)



**Image description:** home page – user logging out



**Image description:** respective session information will be cleared/destroyed when user log out of the system (1)



```
> db.getCollection('sessions').find({})
```

**Image description:** respective session information will be cleared/destroyed when user log out of the system (2)

```
//WITH CONNECTION POOLING
var MongoClient = require('mongodb').MongoClient;
var db2;
var connected = false;

/**
 * Connects to the MongoDB Database with the provided URL
 */
exports.connect = function(url, callback) {
  var options = {
    db : {
      numberOfRetries : 5
    },
    server : {
      auto_reconnect : true,
      poolSize : 1000,
      socketOptions : {
        connectTimeoutMS : 5000
      }
    },
    replSet : {},
    mongos : {}
  };
  MongoClient.connect(url, options, function(err, _db2) {
    if (err) {
      throw new Error('Could not connect: ' + err);
    }
    db2 = _db2;
    connected = true;
    console.log(connected + " is connected?");
    callback(db2);
  });
};

/**
 * Returns the collection on the selected database
 */
exports.collection = function(name) {
  if (!connected) {
    throw new Error('Must connect to Mongo before calling "collection"');
  }
  return db2.collection(name);
};
```

**Image description:** connection pooling used for mongoDB

**RabbitMQ**

Overview    Connections    Channels    Exchanges    **Queues**    Admin

### Queues

▼ All queues (20)

Pagination

Page **1** of 1 - Filter:   Regex (?)()

Overview			Messages			Message rates		
Name	Features	State	Ready	Unacked	Total	Incoming	deliver / get	ack
addaddressstodb_queue	AD	idle	0	0	0			
addtocart_queue	AD	idle	0	0	0			
amq.gen-wUA6-vx5vdwpceRPb1G1cA	Excl AD	idle	0	0	0	0.00/s	0.00/s	0.00/s
createbidad_queue	AD	idle	0	0	0			
createnormalad_queue	AD	idle	0	0	0			
finalcheckout_queue	AD	idle	0	0	0			
getbiditems_queue	AD	idle	0	0	0			
getmyads_queue	AD	idle	0	0	0			
getprofile_queue	AD	idle	0	0	0			
getpurchaseditems_queue	AD	idle	0	0	0			
getsolditems_queue	AD	idle	0	0	0			
gettheheads_queue	AD	idle	0	0	0	0.00/s	0.00/s	0.00/s
getthecart_queue	AD	idle	0	0	0			
loadaddresspayment_queue	AD	idle	0	0	0			
loadcheckoutpage_queue	AD	idle	0	0	0			
login_queue	AD	idle	0	0	0	0.00/s	0.00/s	0.00/s
placebid_queue	AD	idle	0	0	0			
register_queue	AD	idle	0	0	0			
removetheitem_queue	AD	idle	0	0	0			
validate_queue	AD	idle	0	0	0			

**Image description:** message-queues implemented via RabbitMQ

**RabbitMQ**

Overview    Connections    Channels    **Queues**    Admin

### Queue getheads\_queue

▼ Overview

Queued messages (chart: last minute) (?)

Ready: 0  
Unacked: 0  
Total: 0

Message rates (chart: last minute) (?)

Publish	0.00/s
Confirm	0.00/s
Publish (In)	0.00/s
Publish (Out)	0.00/s
Deliver	0.00/s
Redelivered	0.00/s
Acknowledge	0.00/s
Get	0.00/s
Get (noack)	0.00/s
Deliver (noack)	0.00/s
Return	0.00/s

Details

Features	auto-delete: true	State	idle	Total	Ready	Unacked	In memory	Persistent
Policy		Consumers	1	0	0	0	0	0
		Consumer utilisation (?)	0%	Messages (?)	0B	0B	0B	0B
				Message body bytes (?)	0B	0B	0B	0B
				Process memory (?)	9.1kB			

**Image description:** sample operation – reading advertisements from the database by a user using the getheads\_queue

```

//POST
app.post('/checklogin',function(req, res, next) { //AMQPified
    //console.log("CRAP IS " + req.body.username);
    passport.authenticate('checklogin', function(err, user, info) {
        if(err) {
            return next(err);
        }

        if(!user) {
            console.log("session not initialized");
            console.log("Invalid Login");
            var json_responses = {"statusCode" : 401};
            res.send(json_responses);
        }
    });

    req.logIn(user, {session:false}, function(err) {
        if(err) {
            return next(err);
        }

        req.session.email = user.email;
        req.session.username = user.username;
        req.session.fname = user.fname;
        req.session.handle = user.seller_handle;

        req.session.address=user.addressline1;
        req.session.payment=user.ccard_info;
        req.session.lastlogin=user.lastlogin;

        var login_timestamp=new Date();
        fs.appendFile('public/logs/ebayLogs.txt', login_timestamp + ' : ' + 'User signed in successfully : ' +req.session.email+ '\n', function(err){});

        var json_responses = {"statusCode" : 200};
        res.send(json_responses);
    });
});(req, res, next);
});

```

**Image description:** passportjs authentication implementation snippet (1)

```

/*
var passport = require("passport");
var LocalStrategy = require("passport-local").Strategy;
var mongo = require("./mongo");
var mongoURL = "mongodb://localhost:27017/ebay";
var encryption = require("./encryption");
var mq_client = require('../rpc/client');

module.exports = function(passport) {
    passport.use('checklogin', new LocalStrategy(function(username, password, done) {
        //console.log("username, pwd is : " + username + ', ' + password);
        password = encryption.encrypt(password);

        var msg_payload = {
            "email": username, "pword": password
        };
        mq_client.make_request('login_queue', msg_payload, function(err, results) {
            //console.log(results);
            if (err) {
                throw err;
            } else {
                if (results.code.status == 1) {
                    return done(null, false);

                } else if (results.code.status == 2) {
                    return done(null, false);

                } else if (results.code.status == 3) {
                    //console.log('useremail is: ' + user.email);
                    done(null, false);
                }
                else if (results.code.status == 4) {
                    done(null, results.code.user);
                }
            }
        });
    }));
}

```

**Image description:** passportjs authentication implementation snippet (2)

```
function handle_login_request(msg, callback) {
    var res = {};
    console.log("In login_handle request: msg.username is " + msg.email);

    mongo.connect(mongoURL, function(){
        //console.log('Connected to mongo at: ' + mongoURL);
        var coll = mongo.collection('userdetails');

        /* COMMENT THIS PORTION-UPDATION OF DB WITH TIME BEFORE RUNNING JMETER
        mongo.connect(mongoURL, function(){
            var coll = mongo.collection('userdetails');
            var login_time_stamp=new Date();
            login_time_stamp=login_time_stamp.toString();
            console.log("PRINCE JACOB login_time_stamp is: " + login_time_stamp);
            coll.update({"email": msg.email}, {$set:{'lastlogin':login_time_stamp}},function(err, results){
                //not doing anything here
            });
        */ */

        process.nextTick(function(){
            coll.findOne({email: msg.email, pword: msg.pword}, function(error, user) {

                if(error) {
                    //return done(error);
                    res.code = {status:"1", "user": user};
                    callback(null, res);
                }

                if(!user) {
                    //return done(null, false);
                    res.code = {status:"2", "user": user};
                    callback(null, res);
                }

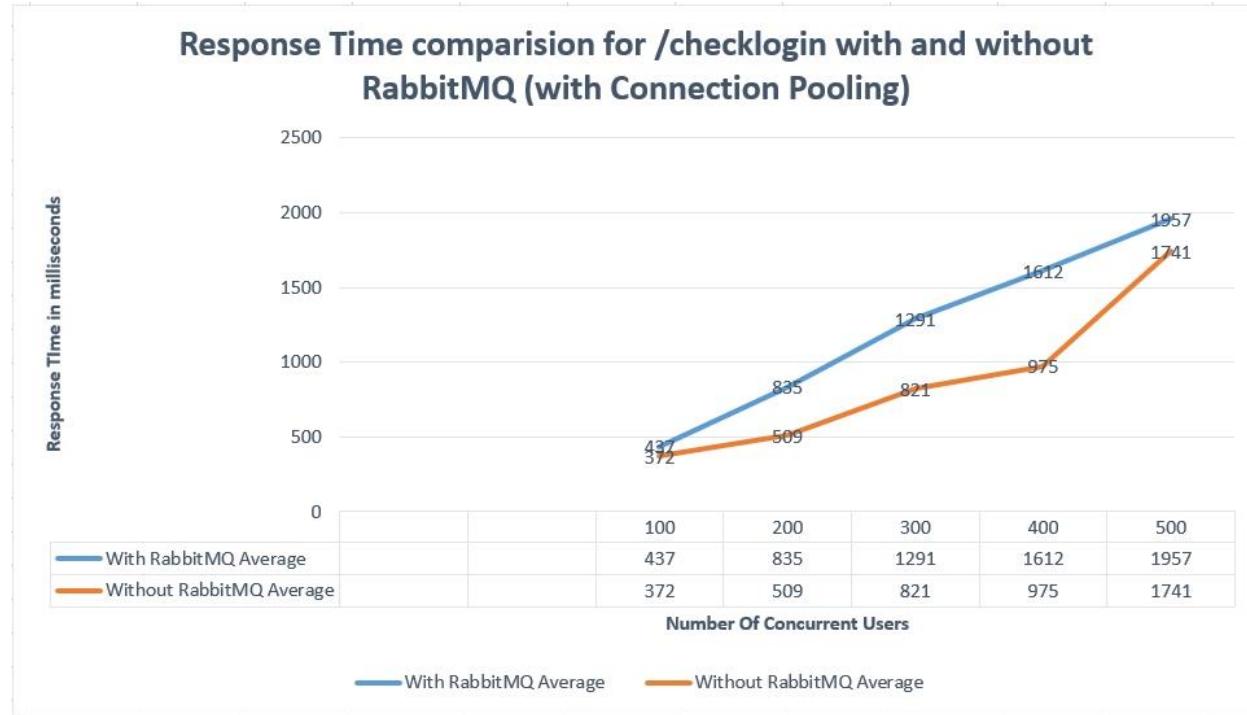
                if(user) {
                    console.log("user exists in db");
                    if(user.pword != msg.pword) {
                        console.log('useremail is: ' + user.email);
                        //done(null, false);
                        res.code = {status:"3", "user": user};
                        callback(null, res);
                    }
                }

                // connection.close();

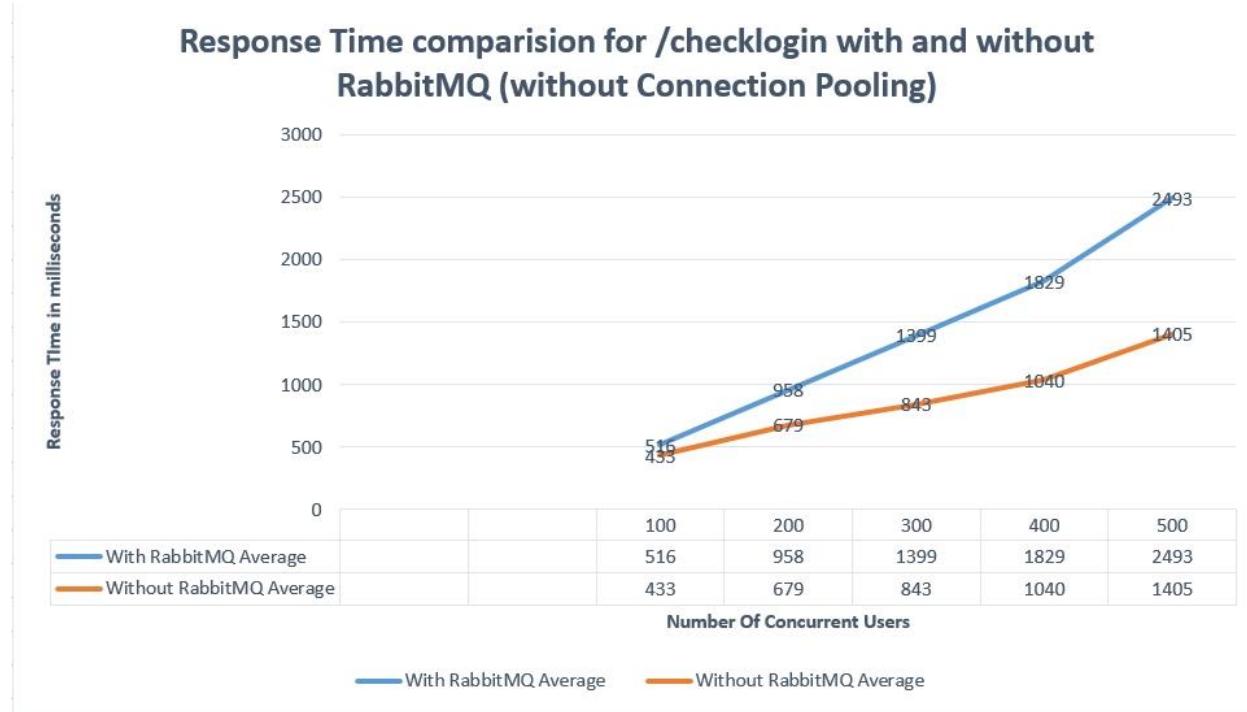
                //done(null, user);
                res.code = {status:"4", "user": user};
                callback(null, res);
            });
        });
    });
}
```

**Image description:** passportjs authentication implementation snippet (3)

## IV. PERFORMANCE GRAPHS & ANALYSIS

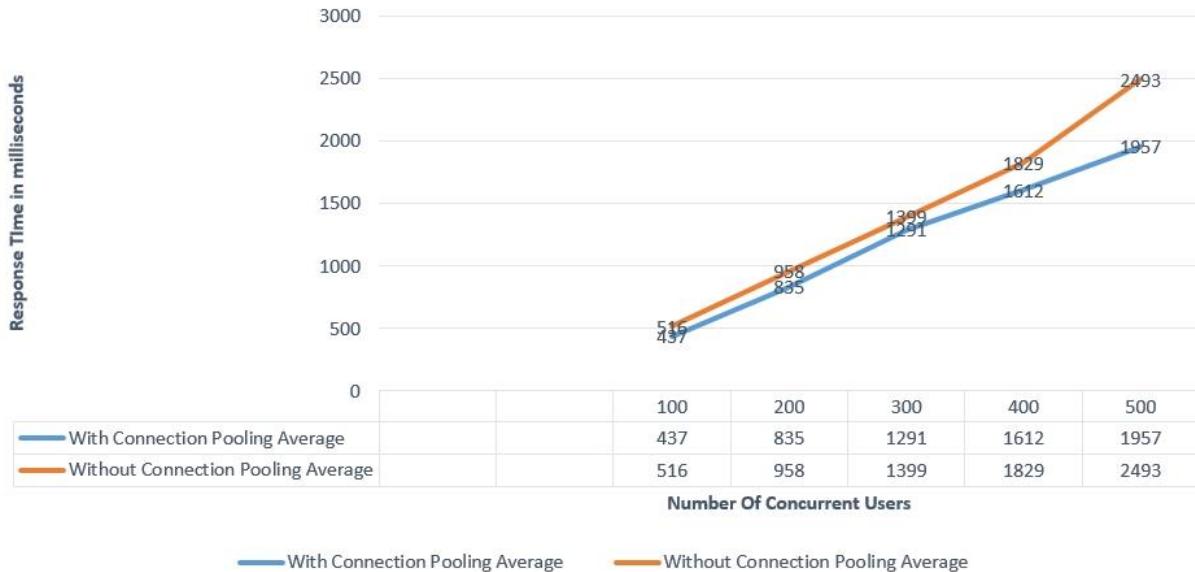


**Image description:** Comparison - number of concurrent users versus average response time in milliseconds for sign- in operation with and without using RabbitMQ (with connection pool)



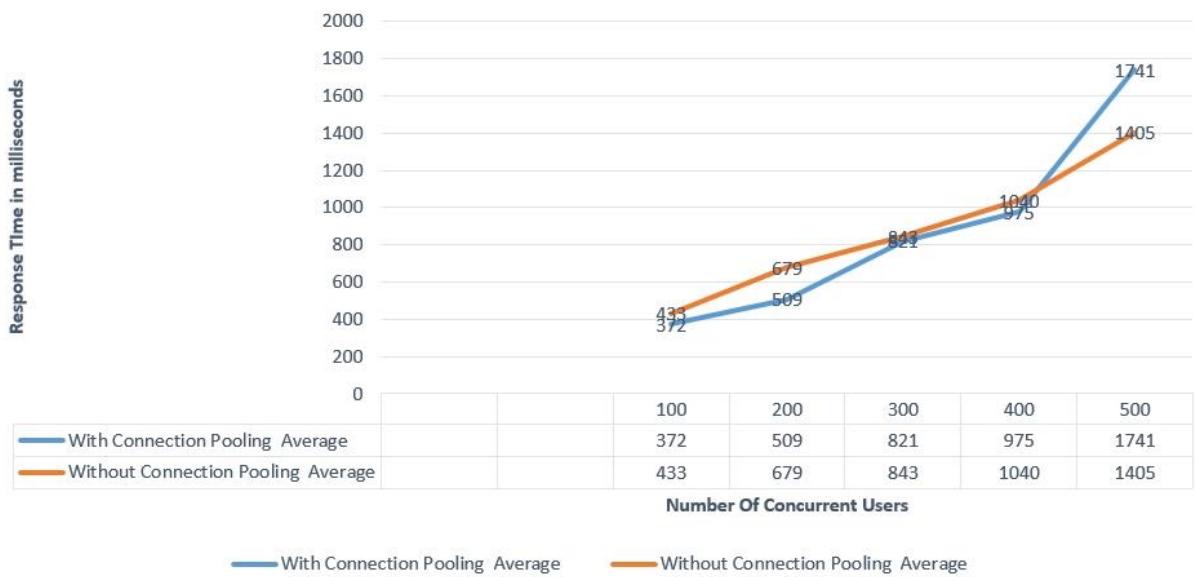
**Image description:** Comparison - number of concurrent users versus average response time in milliseconds for sign- in operation with and without using RabbitMQ (without connection pool)

### Response Time comparision for /checklogin with and without Connection Pooling (with rabbitMQ)

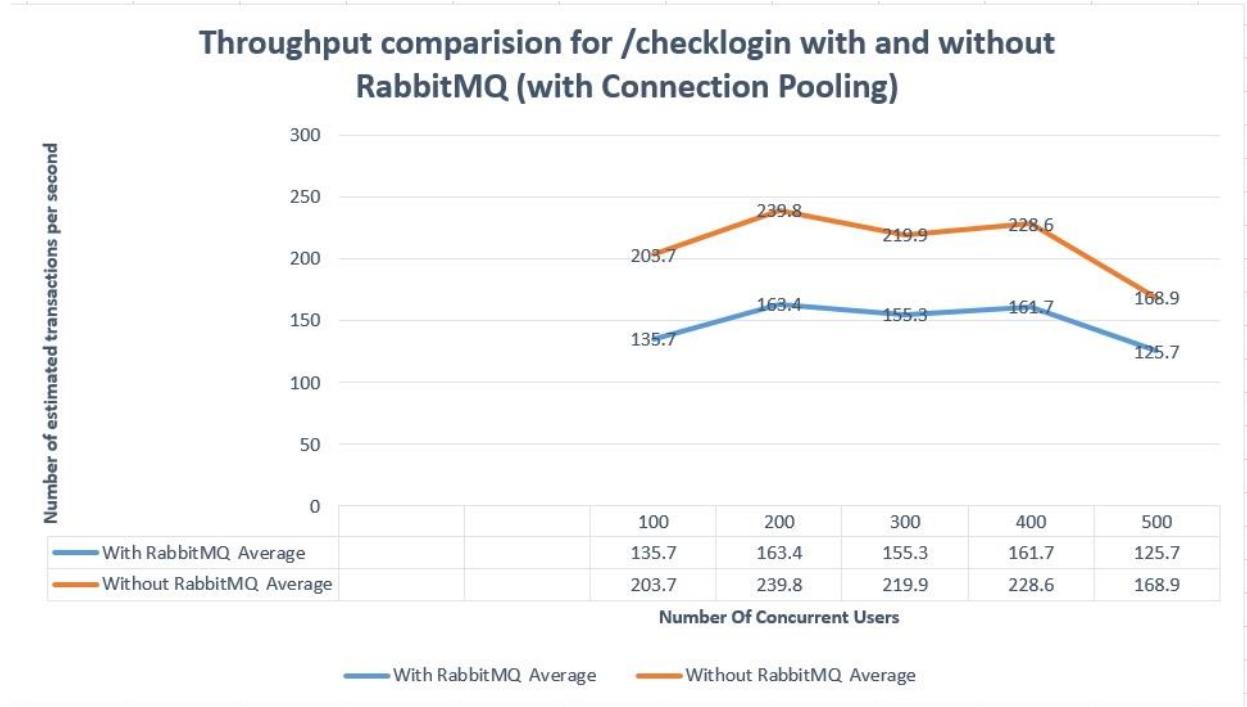


**Image description:** Comparison - number of concurrent users versus average response time in milliseconds for sign- in operation with and without using connection pool (with RabbitMQ)

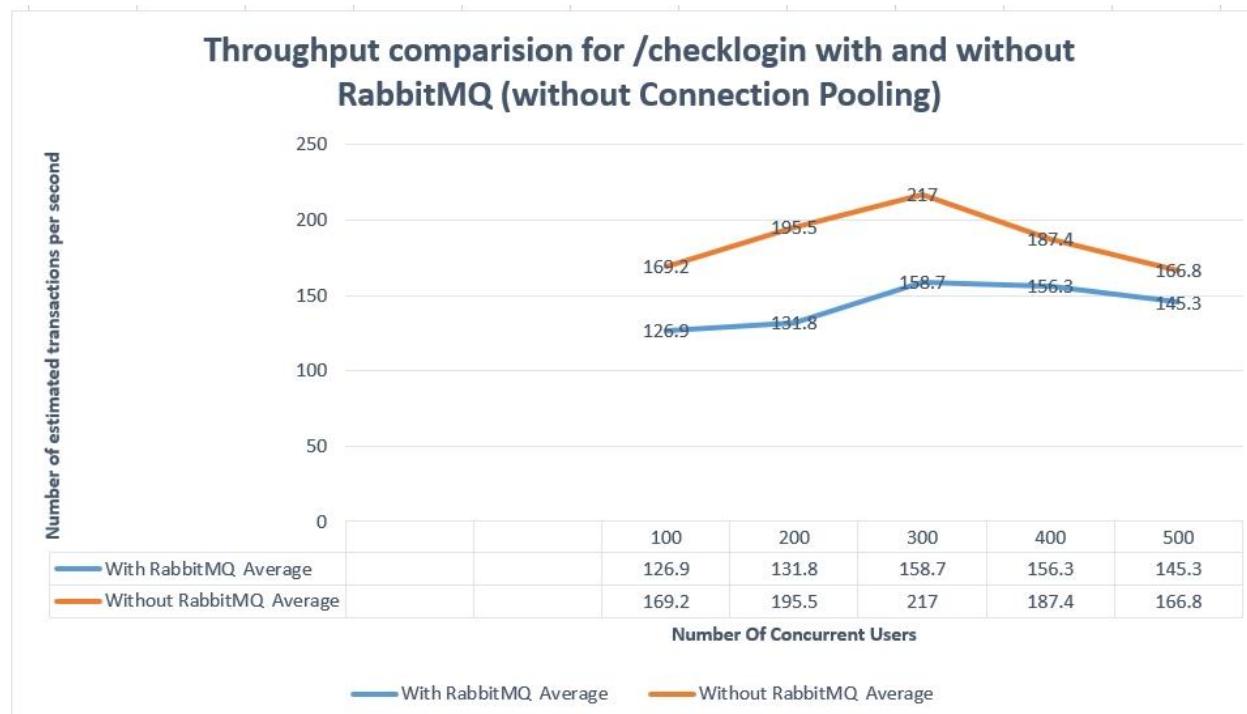
### Response Time comparision for /checklogin with and without Connection Pooling (without RabbitMQ)



**Image description:** Comparison - number of concurrent users versus average response time in milliseconds for sign- in operation with and without using connection pool (without RabbitMQ)



**Image description:** Throughput Comparison - number of concurrent users versus number of estimated transactions per second for sign- in operation with and without using RabbitMQ (with connection pool)

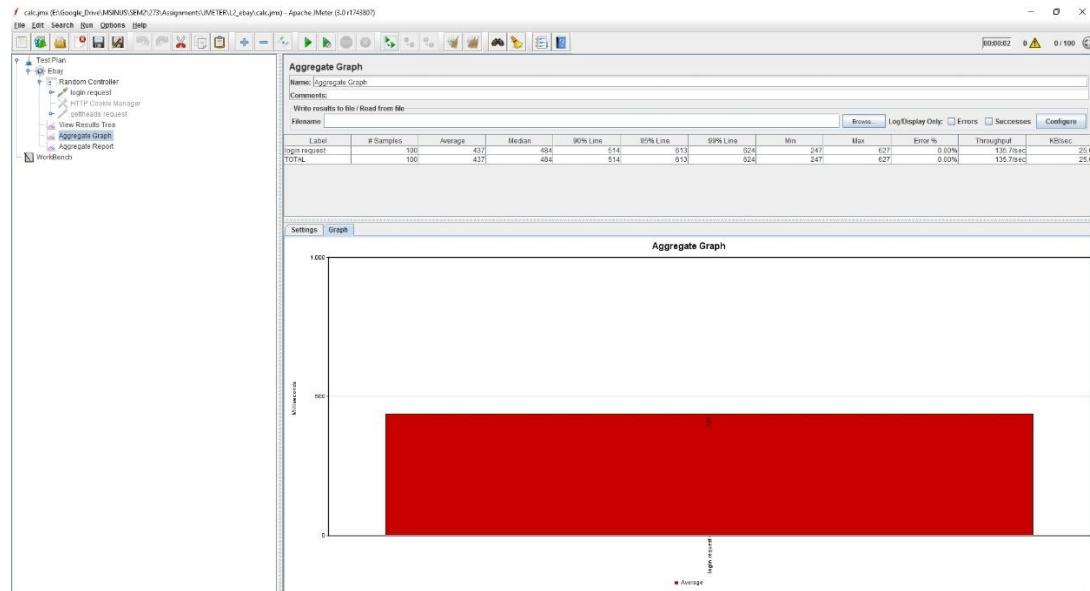


**Image description:** Throughput Comparison - number of concurrent users versus number of estimated transactions per second for sign- in operation with and without using RabbitMQ (without connection pool)

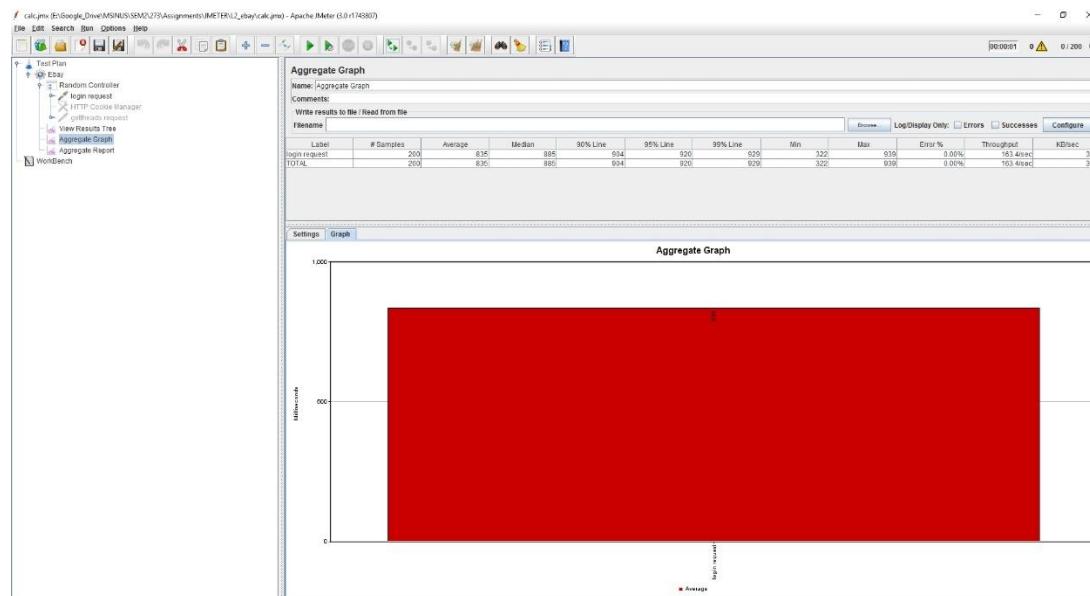
Analysis:

Response time offered with connection pooling implies better performance than the one offered without connection pooling for all the 5 test scenarios (100, 200, 300, 400, 500 users). Connection pooling improves response time, as the time associated with creation and deletion of connections are avoided.

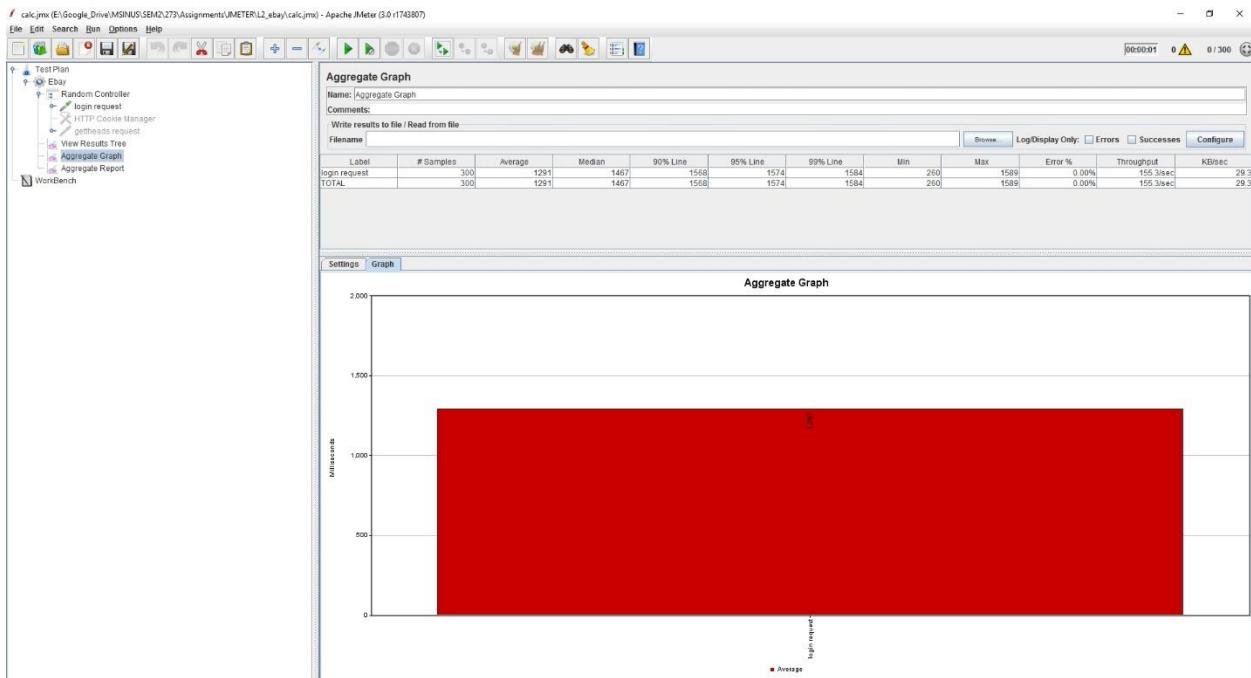
**1) Plot - Number of concurrent users versus average response time in milliseconds, with both RabbitMQ and connection pooling**



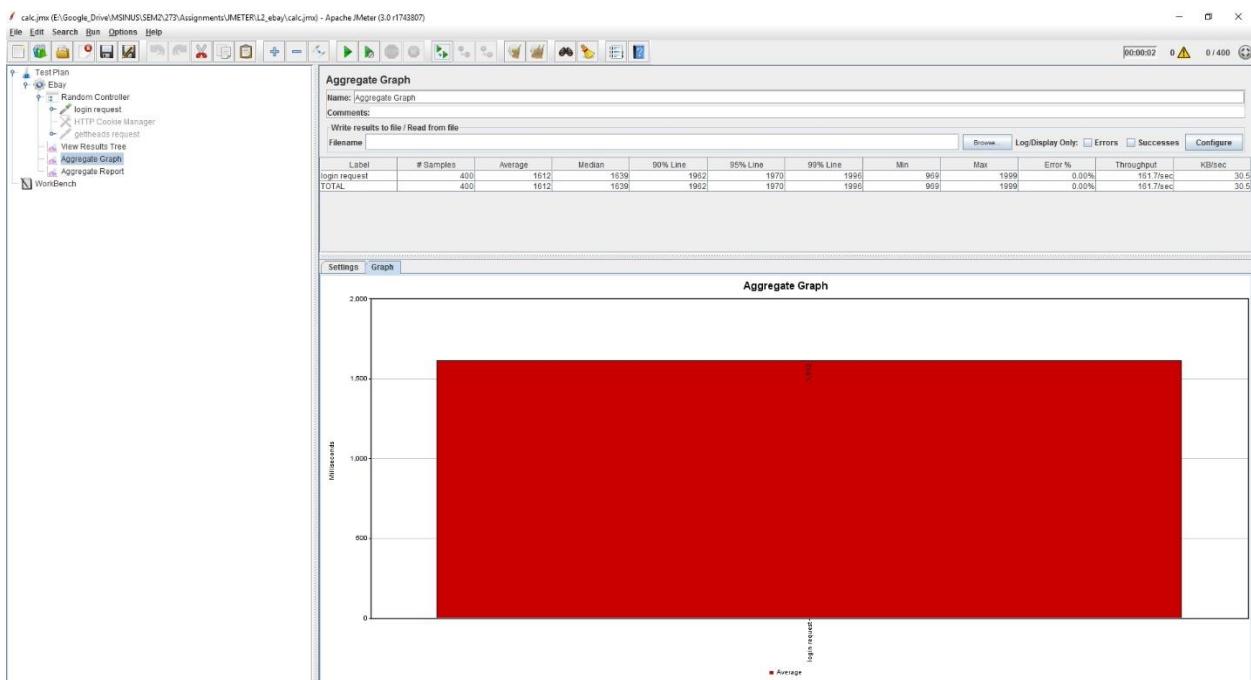
**Image description:** Number of concurrent users versus average response time in milliseconds – for 100 users



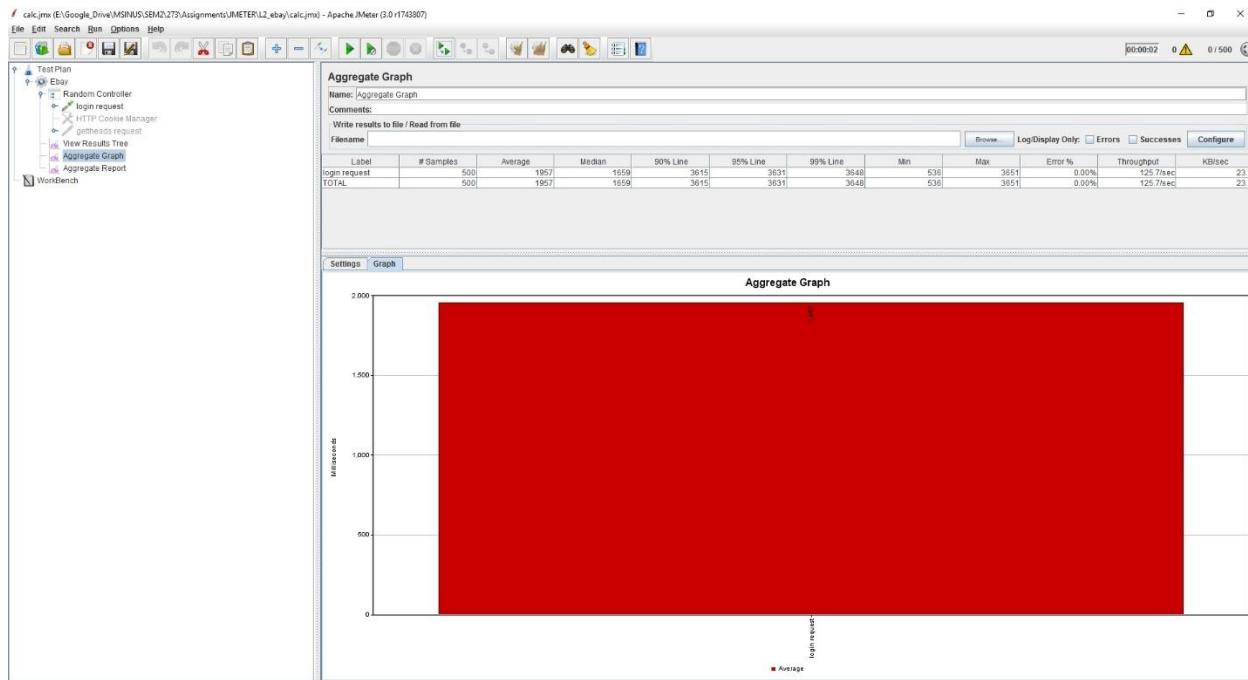
**Image description:** Number of concurrent users versus average response time in milliseconds – for 200 users



**Image description:** Number of concurrent users versus average response time in milliseconds – for 300 users

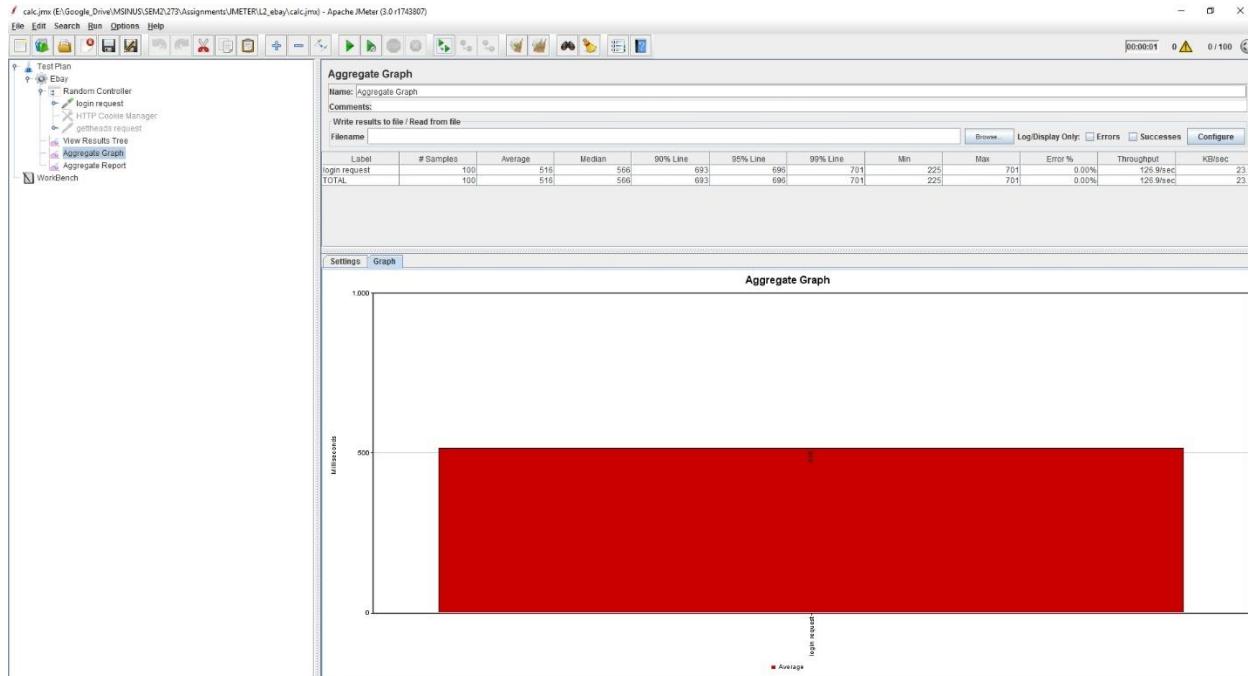


**Image description:** Number of concurrent users versus average response time in milliseconds – for 400 users

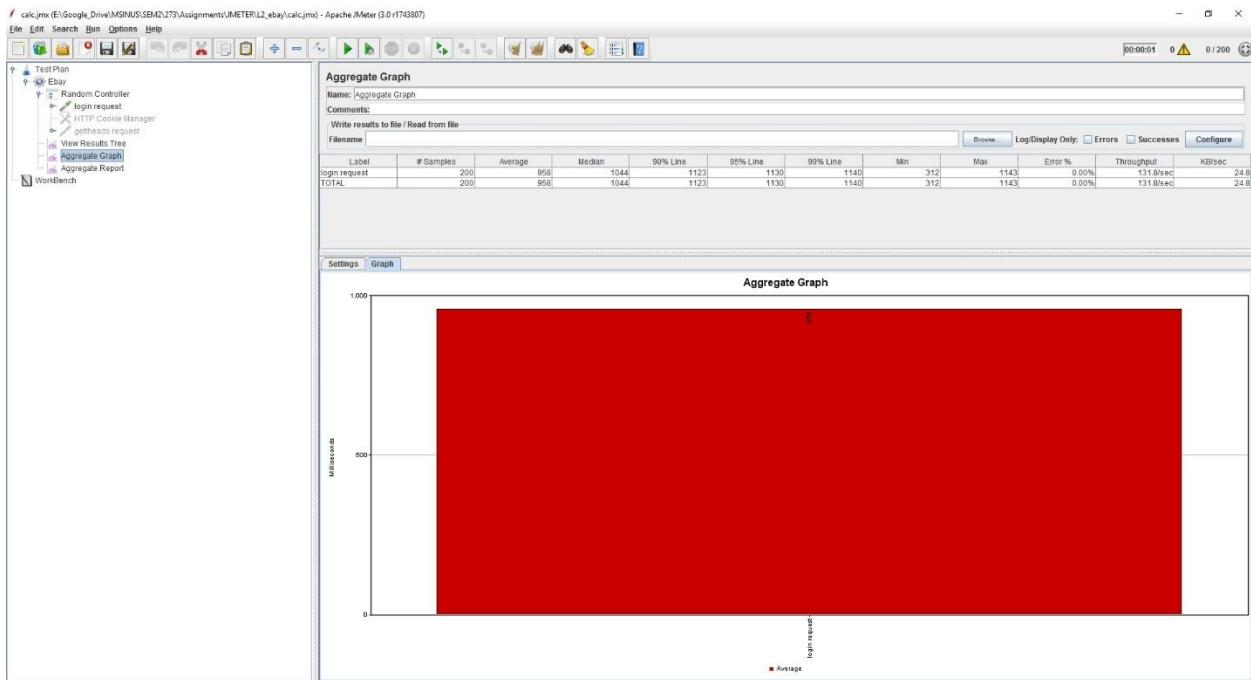


**Image description:** Number of concurrent users versus average response time in milliseconds – for 500 users

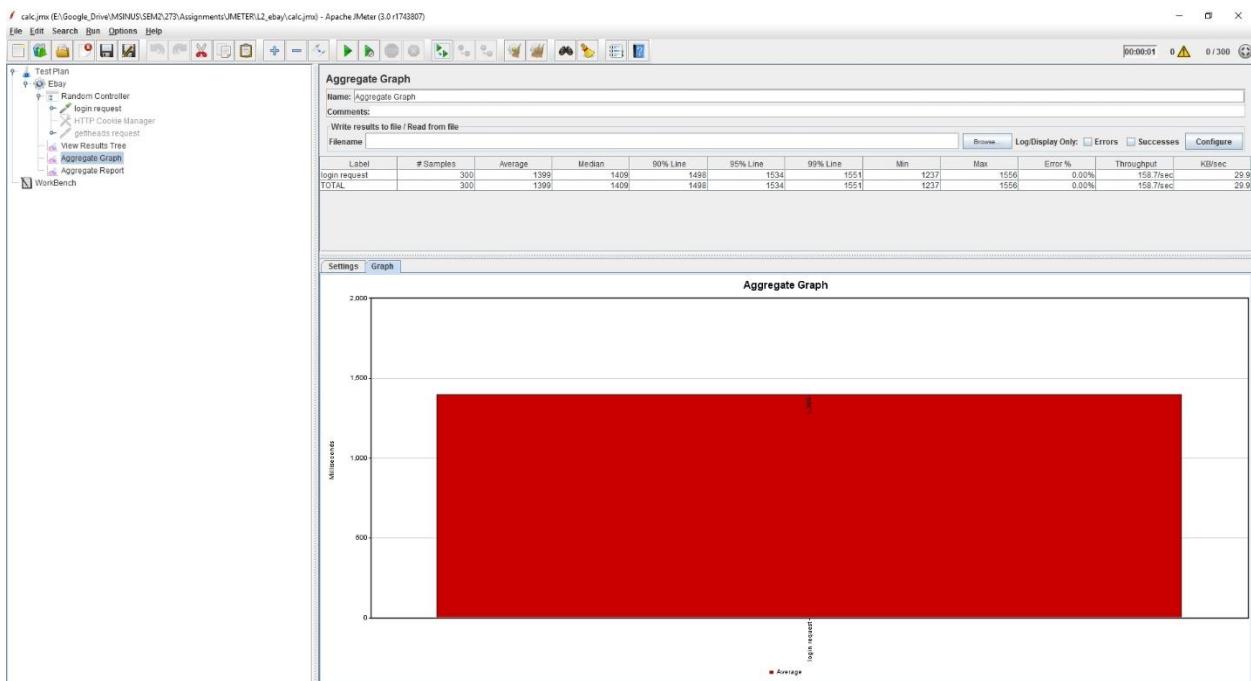
## 2) Plot - Number of concurrent users versus average response time in milliseconds, with RabbitMQ and without connection pooling



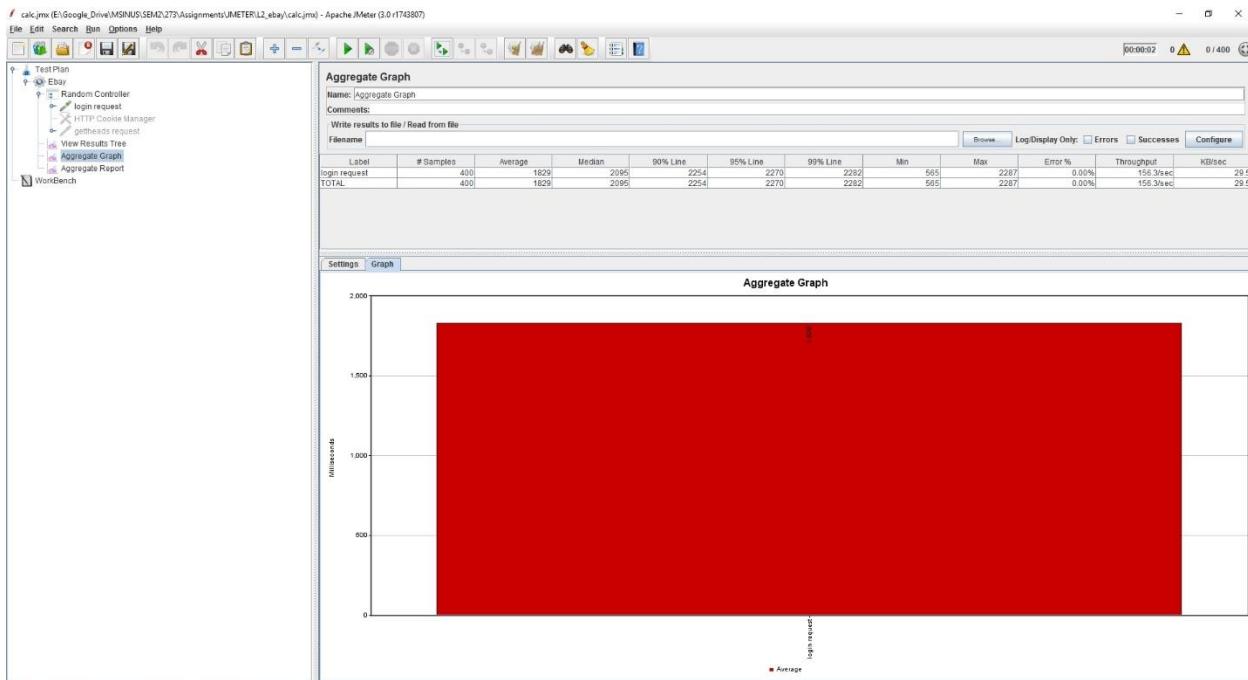
**Image description:** Number of concurrent users versus average response time in milliseconds – for 100 users



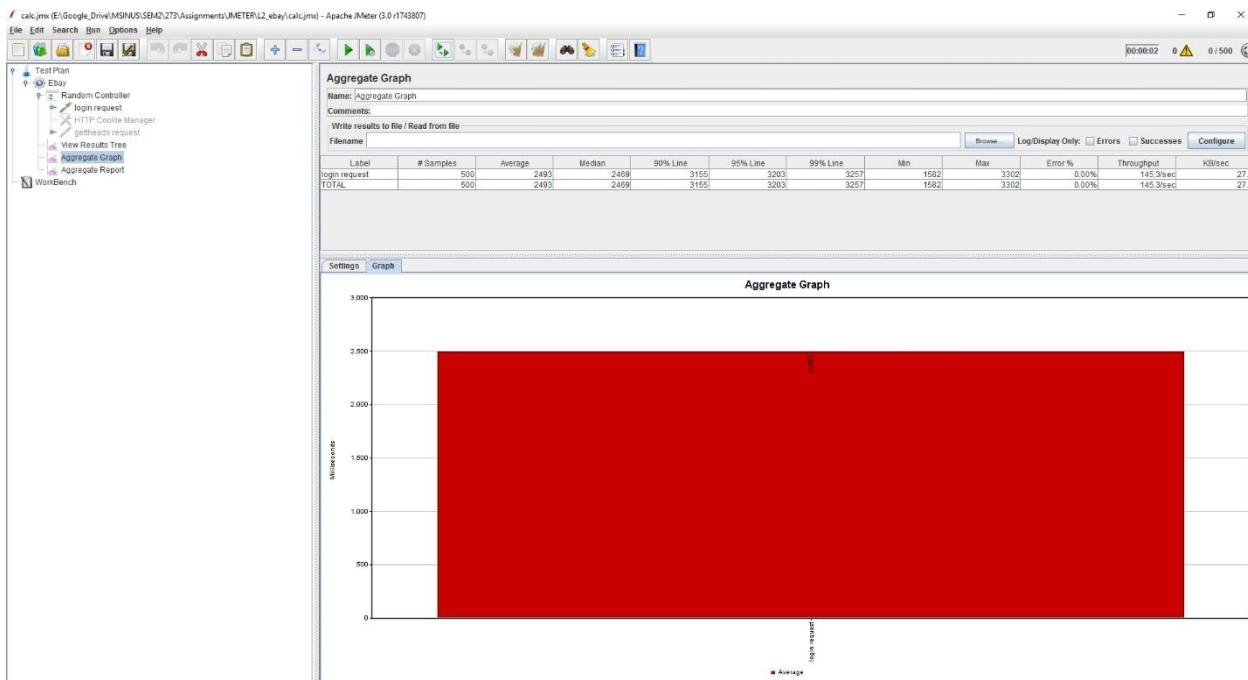
**Image description:** Number of concurrent users versus average response time in milliseconds – for 200 users



**Image description:** Number of concurrent users versus average response time in milliseconds – for 300 users

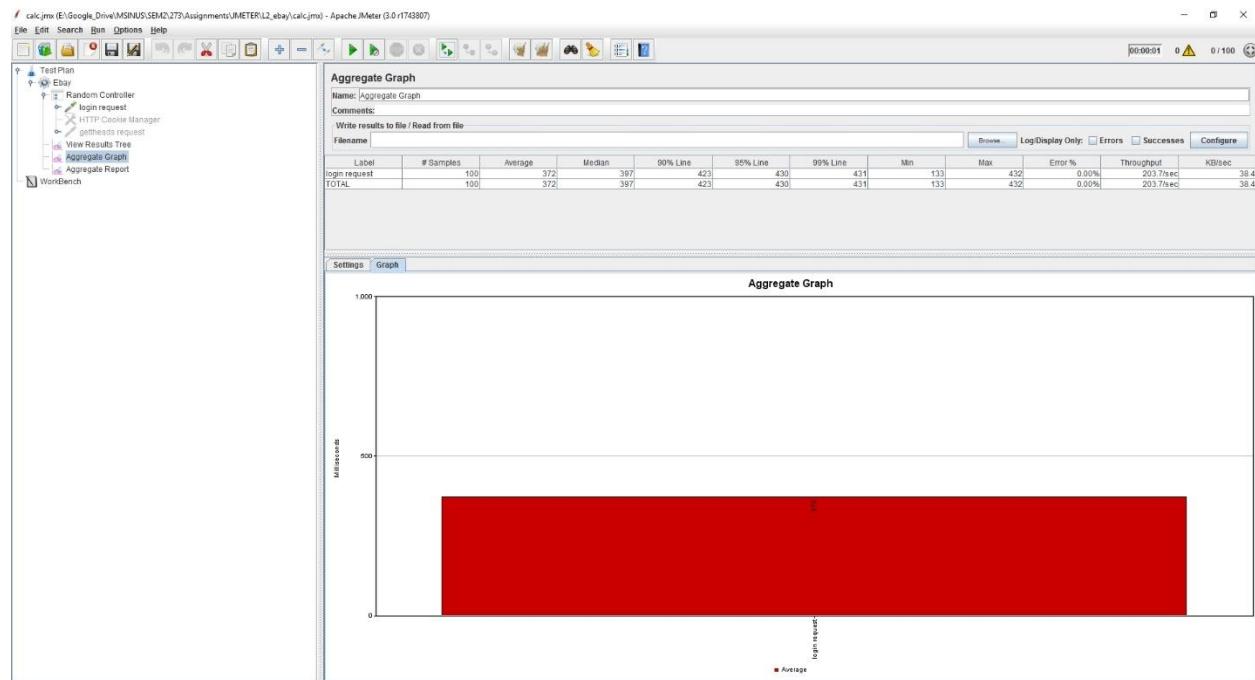


**Image description:** Number of concurrent users versus average response time in milliseconds – for 400 users

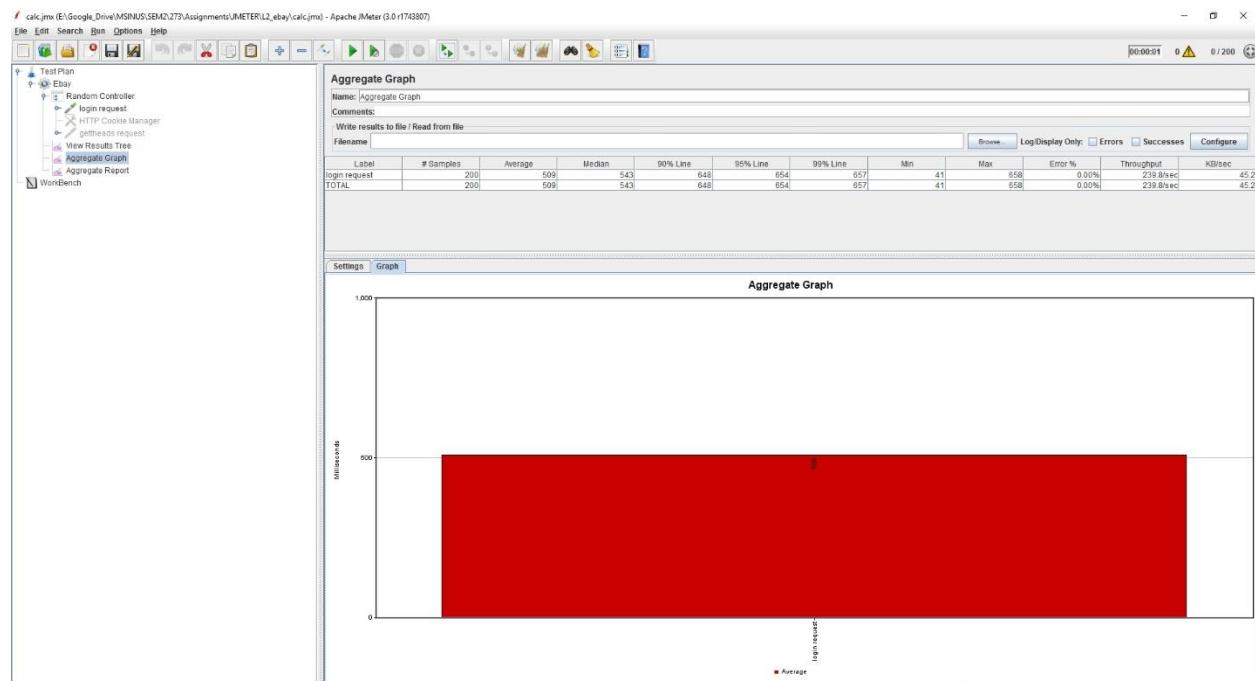


**Image description:** Number of concurrent users versus average response time in milliseconds – for 500 users

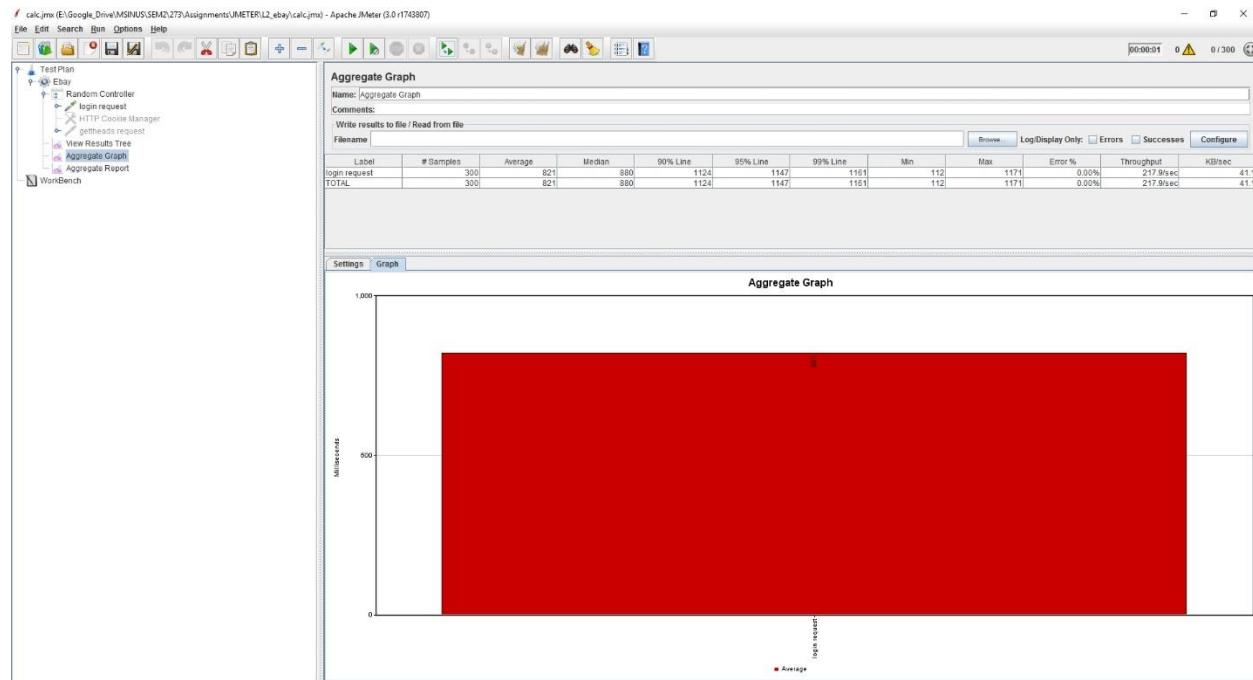
3) Plot - Number of concurrent users versus average response time in milliseconds, without RabbitMQ and with connection pooling



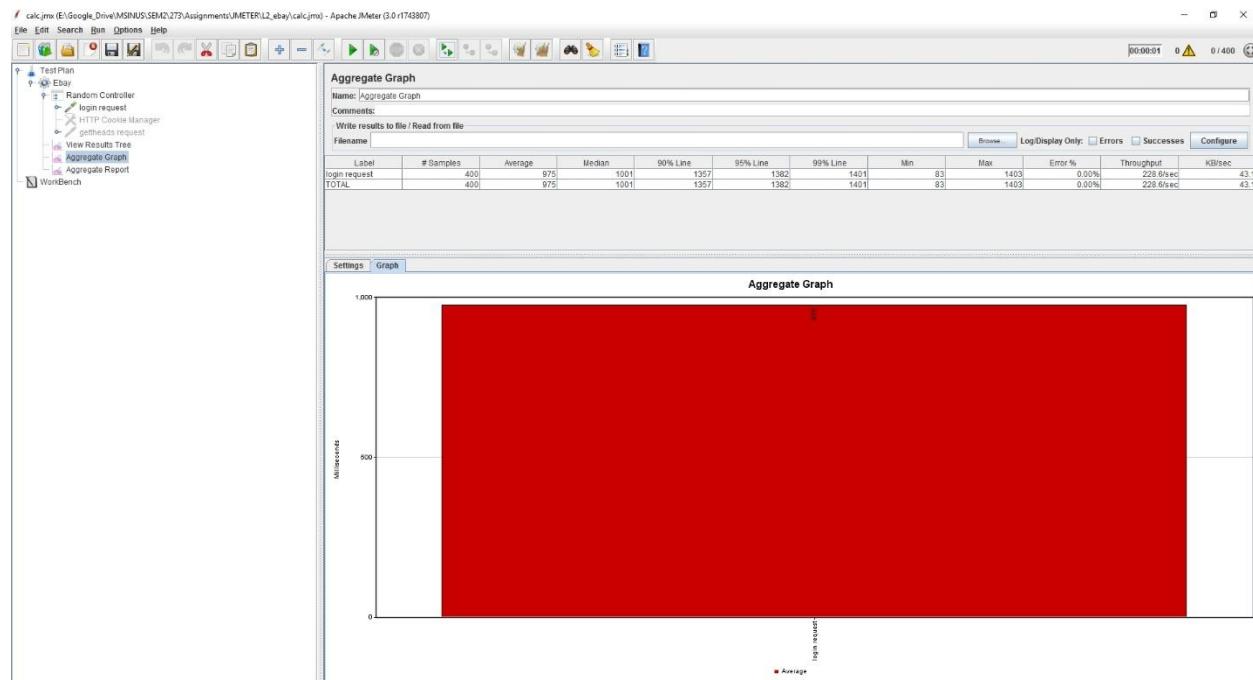
**Image description:** Number of concurrent users versus average response time in milliseconds – for 100 users



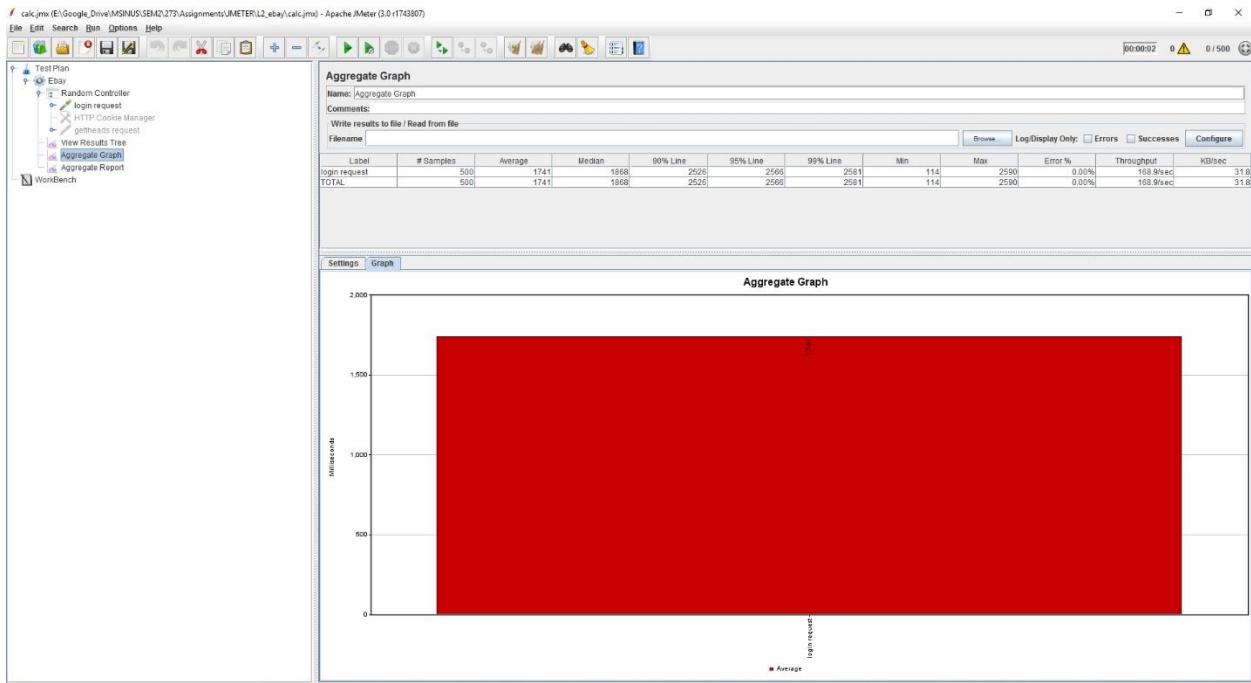
**Image description:** Number of concurrent users versus average response time in milliseconds – for 200 users



**Image description:** Number of concurrent users versus average response time in milliseconds – for 300 users

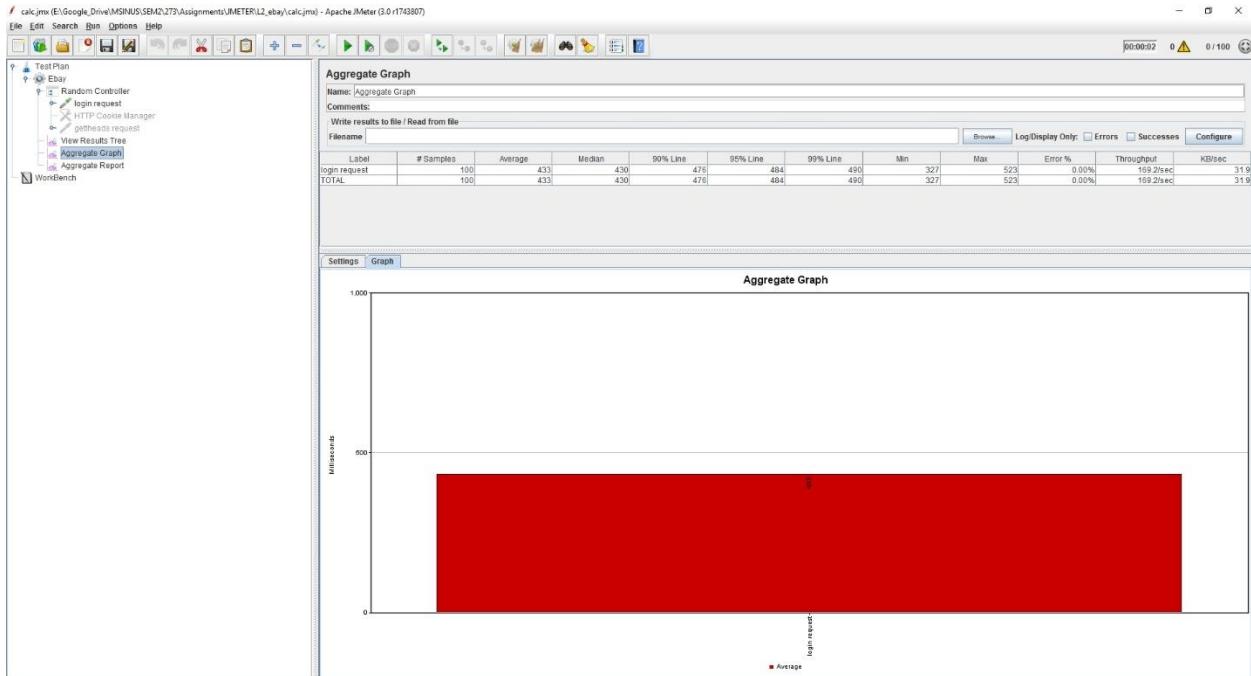


**Image description:** Number of concurrent users versus average response time in milliseconds – for 400 users

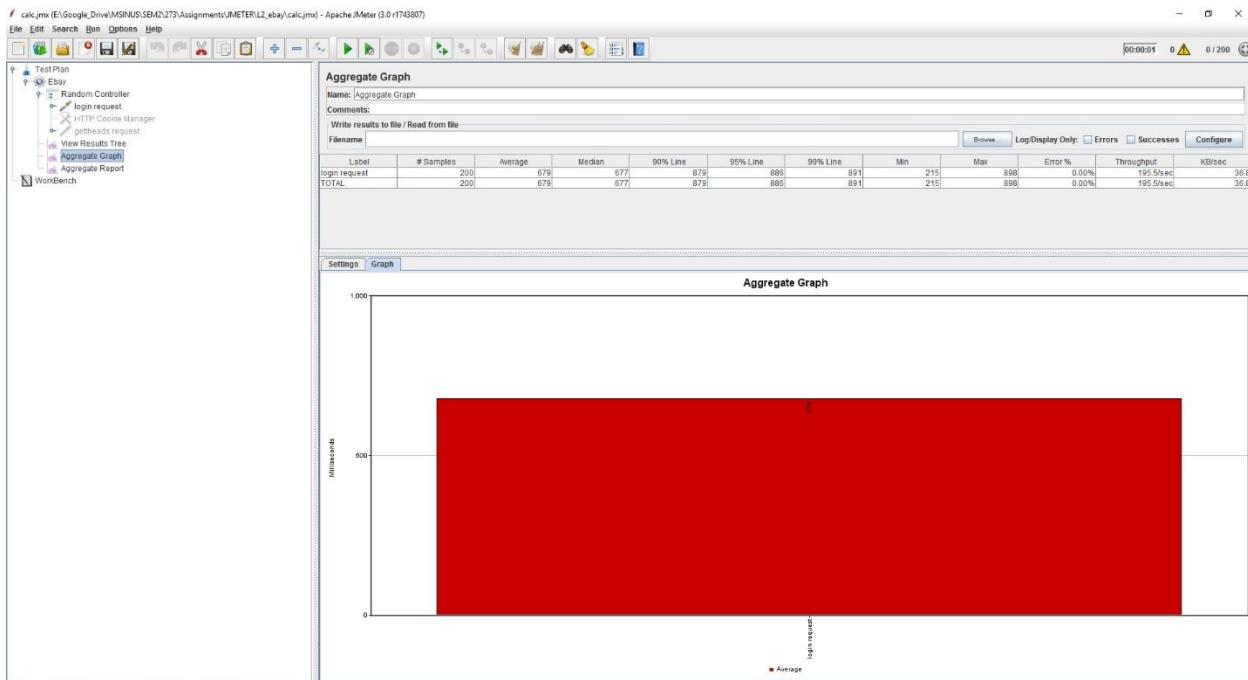


**Image description:** Number of concurrent users versus average response time in milliseconds – for 500 users

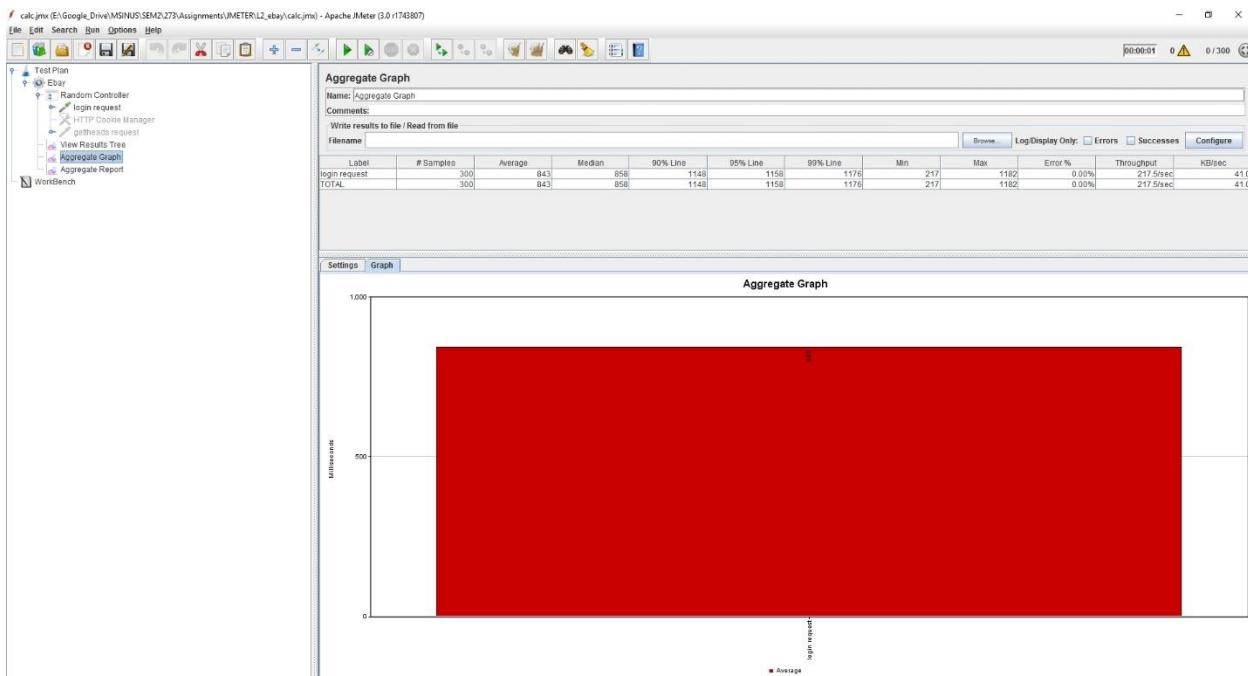
4) Plot - Number of concurrent users versus average response time in milliseconds, without both RabbitMQ and connection pooling



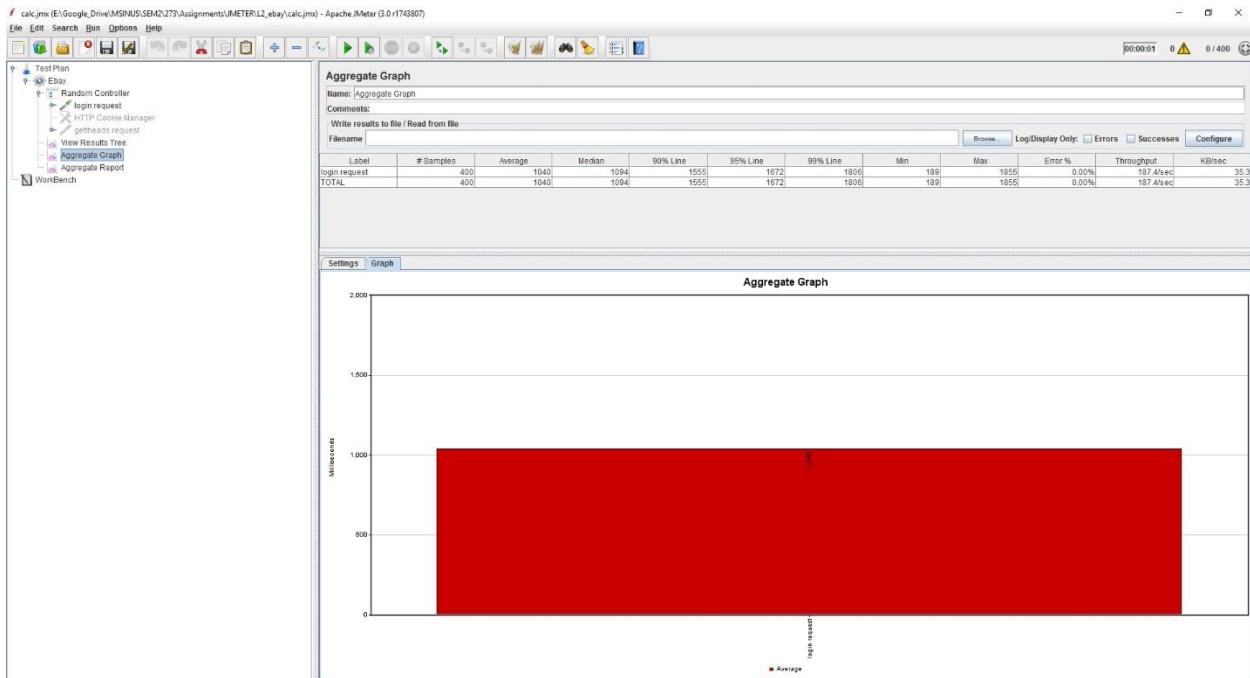
**Image description:** Number of concurrent users versus average response time in milliseconds – for 100 users



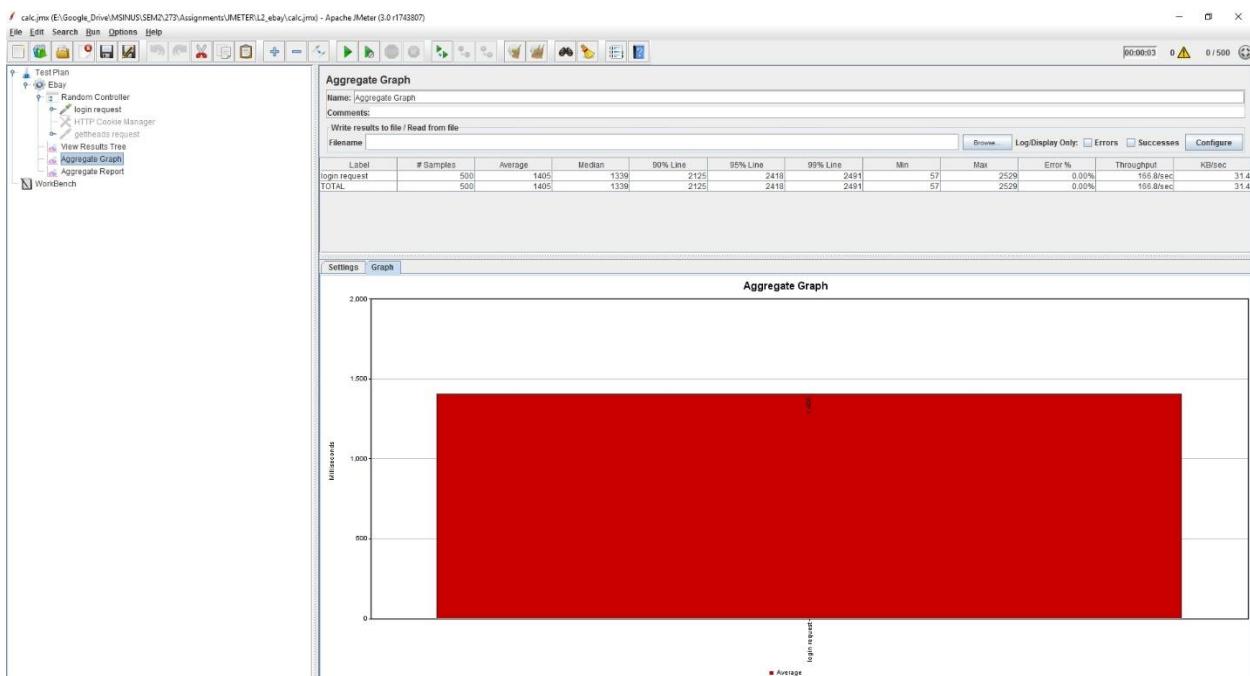
**Image description:** Number of concurrent users versus average response time in milliseconds – for 200 users



**Image description:** Number of concurrent users versus average response time in milliseconds – for 300 users



**Image description:** Number of concurrent users versus average response time in milliseconds – for 400 users



**Image description:** Number of concurrent users versus average response time in milliseconds – for 500 users

**Testing Using mocha framework:**

```
C:\WINDOWS\system32\cmd.exe
E:\Google_Drive\eclipse-enide\WORKSPACE\LAB2_ebay\LAB2_ebay_client\MochaTest>mocha mochatest.js

http tests
  ✓ Load signin page
  ✓ Validate Login Operation
  ✓ Load Profile
  ✓ Load Shopping Cart
  ✓ Checkout

  5 passing (82ms)
```

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

it('Validate Login Operation', function(done) {
  request.post(
    'http://localhost:3000/checklogin',
    { form: { username: "princeofalltimes@gmail.com", password: "prince" } },
    function (error, response, body) {
      assert.equal(200, response.statusCode);
      done();
    }
  );
});

it('Load Profile', function(done) {
  request.get(
    'http://localhost:3000/profile',
    function (error, response, body) {
      assert.equal(200, response.statusCode);
      done();
    }
  );
});

it('Load Shopping Cart', function(done) {
  request.get(
    'http://localhost:3000/cartpage',
    function (error, response, body) {
      assert.equal(200, response.statusCode);
      done();
    }
  );
});

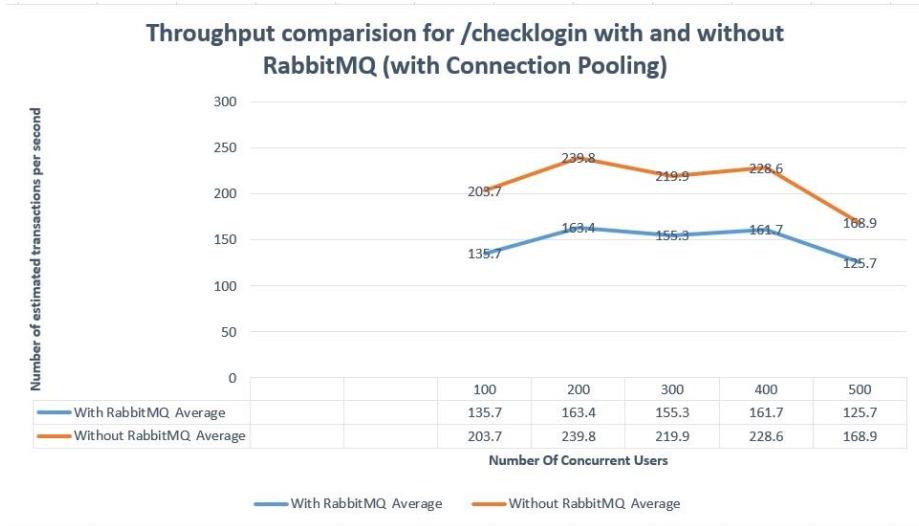
it('Checkout', function(done) {
  request.get(
    'http://localhost:3000/checkout',
    function (error, response, body) {
      assert.equal(200, response.statusCode);
      done();
    }
  );
});
```

**Functions tested as part of Mocha Unit Testing**

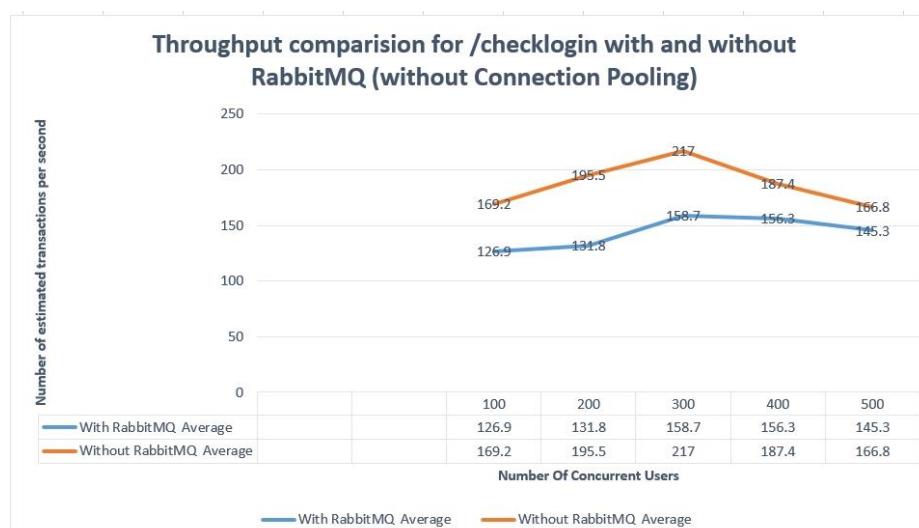
### Part 3 – Question & Answers

**Question 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.

**Answer:** From the jmeter testing/results, it can be observed the case without RabbitMQ provides better response time as well as throughput when compared with the case using RabbitMQ. This is due to the addition of the extra layer in the communication. And also, the current system consists of both the RabbitMQ server and the ebay client in the same machine, which is not optimum to efficiently utilize the resources. Improved reliability and consistency associated with message queues comes at the expense of performance.



**Image description:** Throughput Comparison - number of concurrent users versus number of estimated transactions per second for sign- in operation with and without using RabbitMQ (with connection pool)



**Image description:** Throughput Comparison - number of concurrent users versus number of estimated transactions per second for sign- in operation with and without using RabbitMQ (without connection pool)

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

**Answer:**

Lab1 application uses AES encryption algorithm

- AES is more secure (it is less susceptible to cryptanalysis than 3DES)
- AES supports larger key sizes than 3DES's 112 or 168 bits
- AES is faster in both hardware and software
- AES's 128-bit block size makes it less open to attacks via the birthday problem than 3DES with its 64-bit block size.
- AES is required by the latest U.S. and international standards.

Lab2 application uses passport authentication process

- Passport is authentication middleware for Node.js.
- Extremely flexible and modular
- Passport can be dropped in to any Express-based web application.
- Designed for simple, easy authentication
- Fall short (by design) of broader user management needs (still left to design, implement, and maintain all your other user infrastructure)

**Question 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.

**Answer:** From ebay/Airbnb point of view, it is better to store document related data such as images, videos etc.. to be stored in mongoDB. At the same time, I would be storing the credit card details and user-pertaining fixed/important/less-accessed data in MySQL database.

Another best example would be the booking engine behind a travel reservation system, which also typically involves complex transactions. While the core booking engine might run on MySQL, those parts of the app that engage with users – serving up content, integrating with social networks, managing sessions – would be better placed in MongoDB. Applications that require complex, multi-row transactions (e.g., a double-entry bookkeeping system) would be good examples that depend upon MySQL.

Relation databases has fixed structure and it's better to use them for storing secured and confidential data (important data), whereas mongoDB is lightweight and can be used where quick processing of data and huge amount data need to be dealt with. It's flexible and scalable.