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.

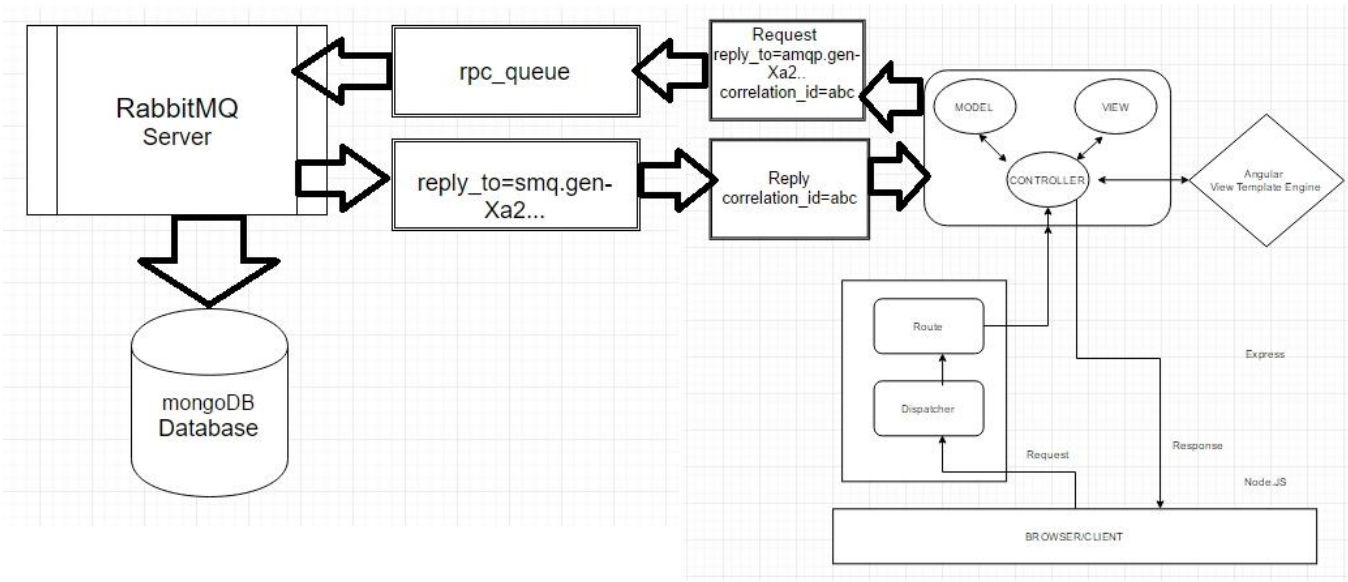
user can confirm buying of items in his shopping cart via checkout option

connection pooling for mongoDB connections

validations for all inputs provided by the user

selling, bidding and purchase

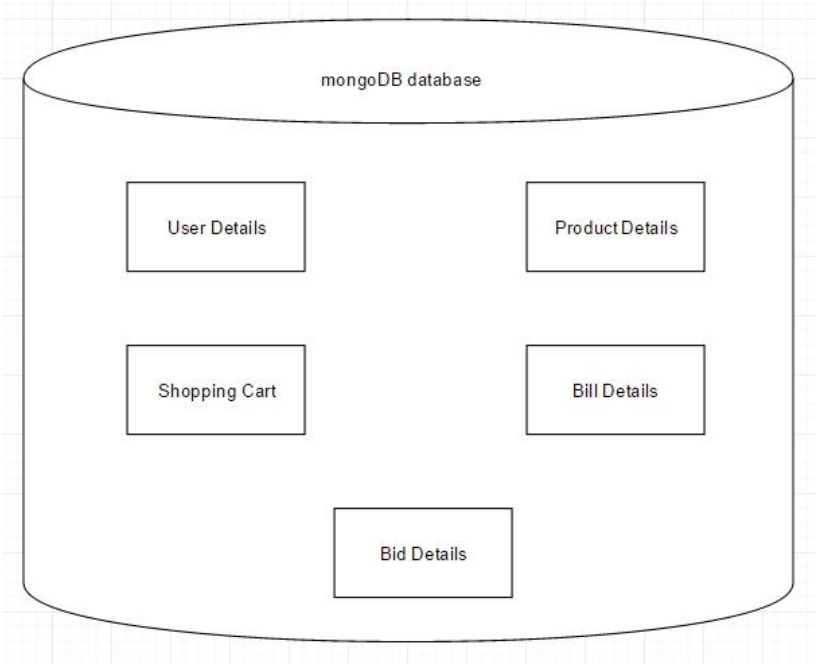
II. System Design



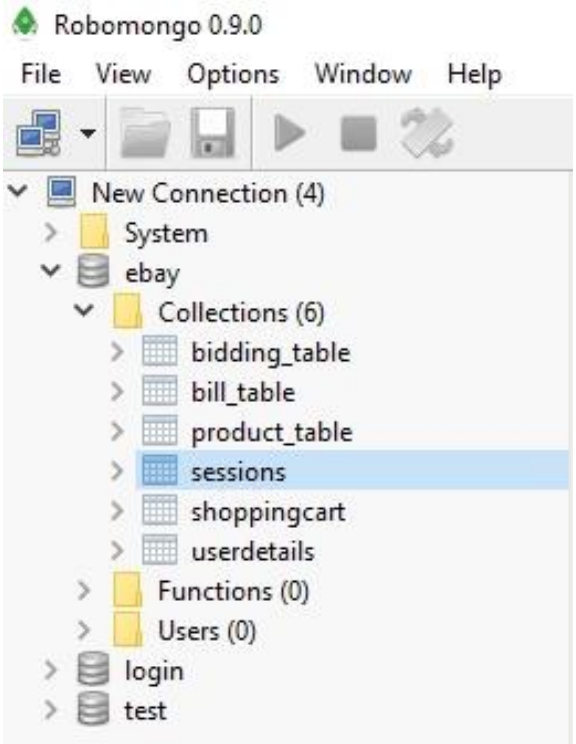
The view (HTML5) is controlled by AngularJS through Express Controller

Server side services are implemented by Node.JS

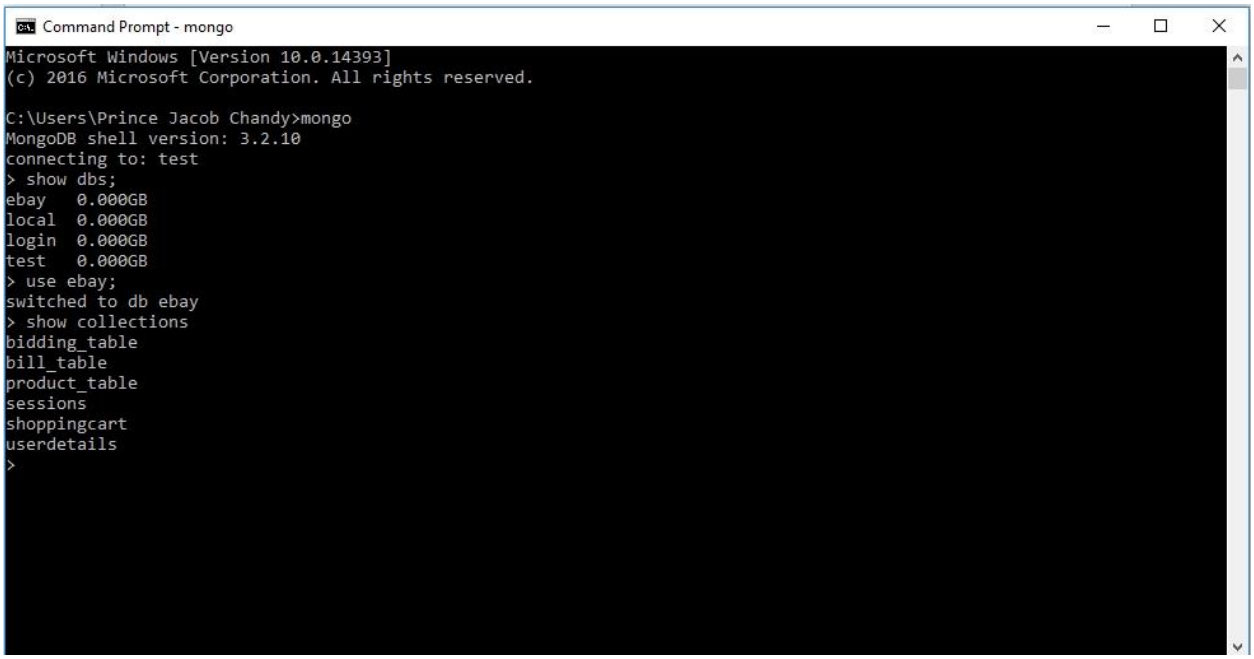
Server establishes connection via a connection pool and talks to mongoDB database module



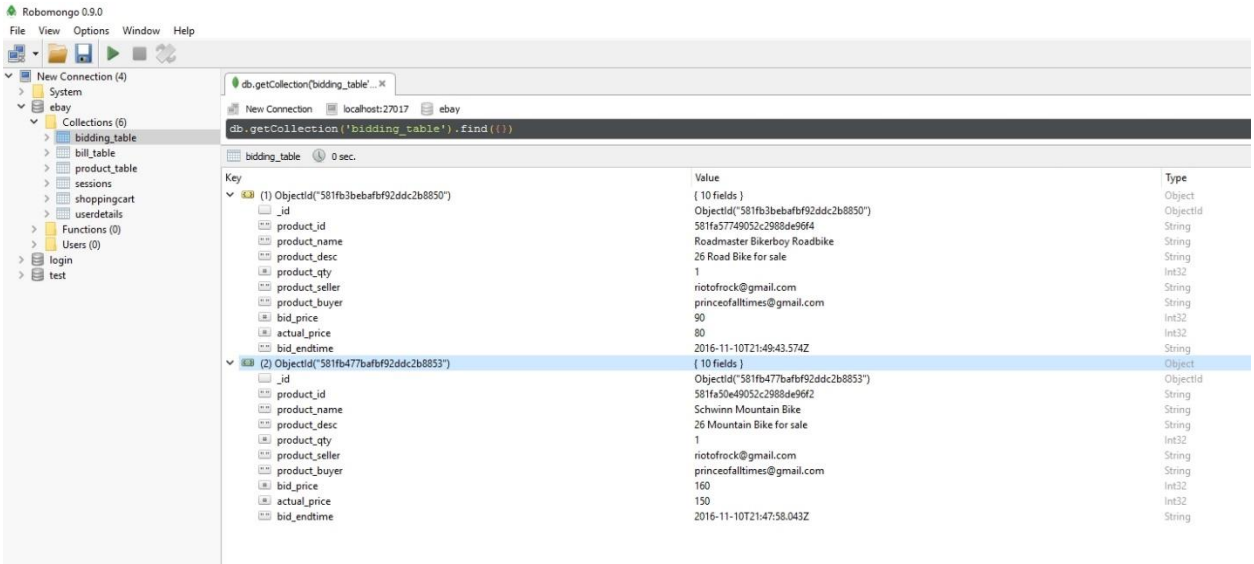
mongoDB Database design with the various collections (1)



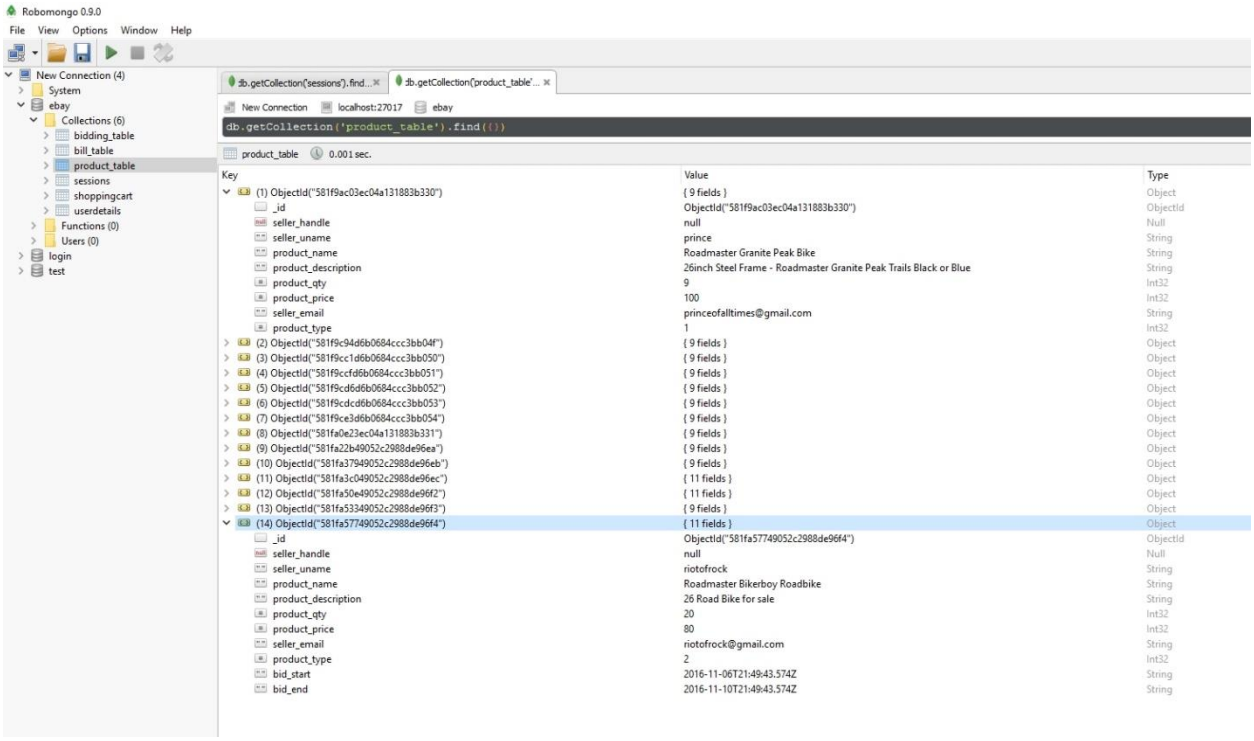
mongoDB Database design with the various collections (2)



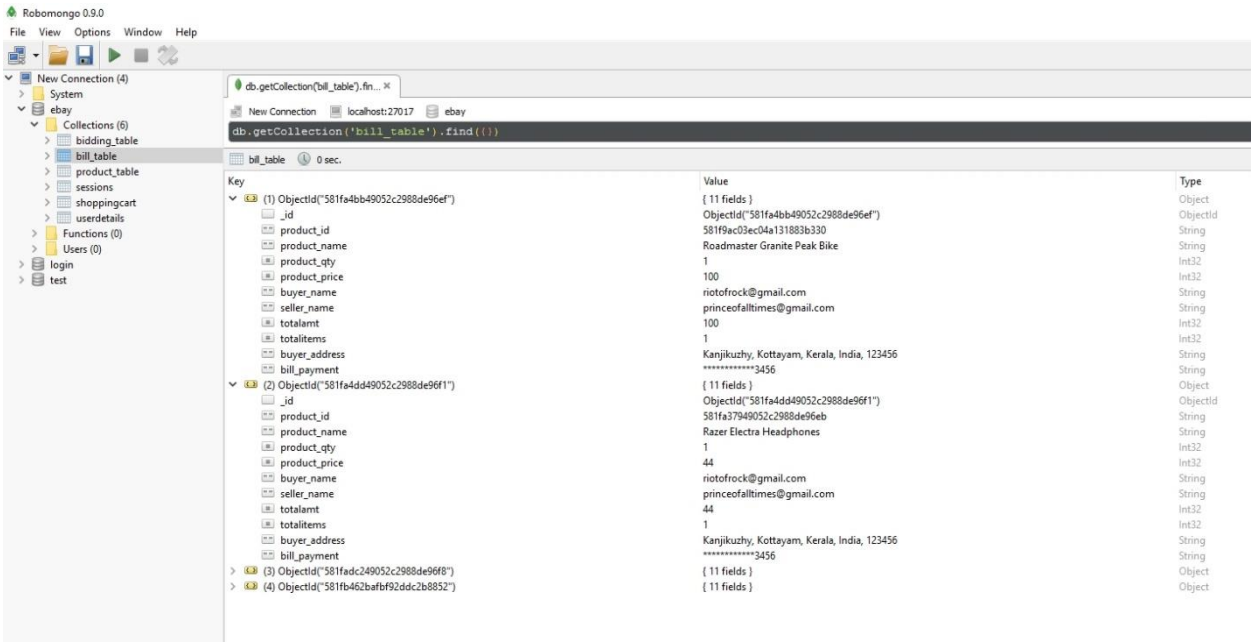
mongoDB Database design with the various collections (3)



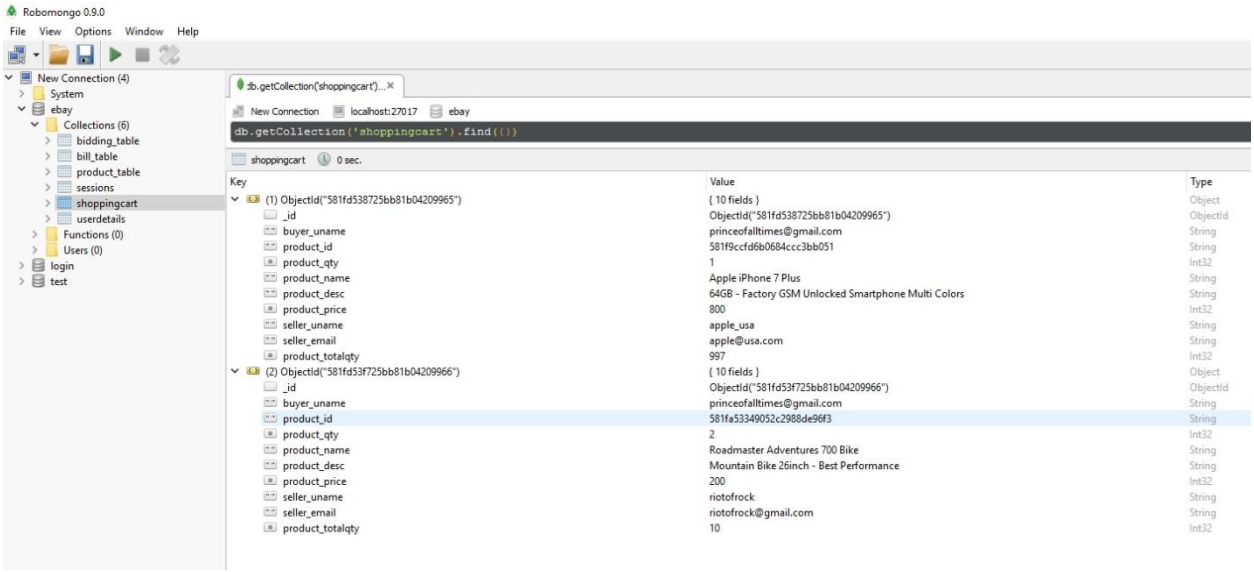
Bid details collection



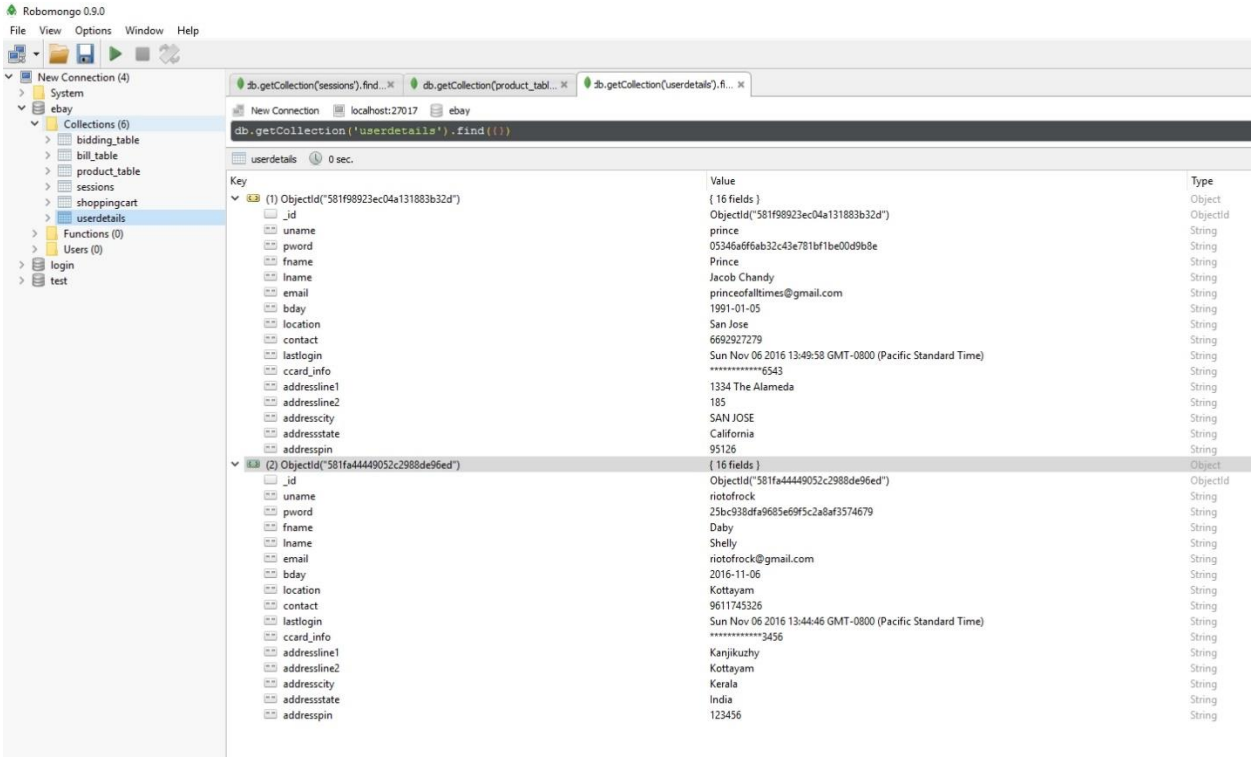
Product details collection



Bill details collection



Shopping cart collection



User details collection

III. RESULTS

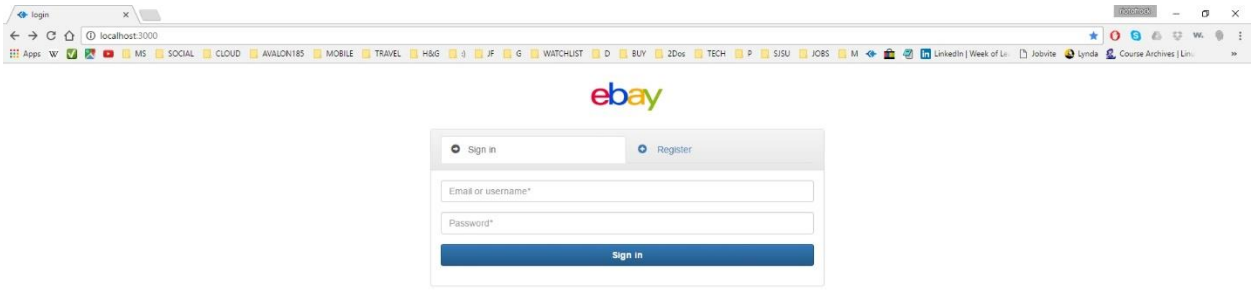


Image description: login page (sign in options)

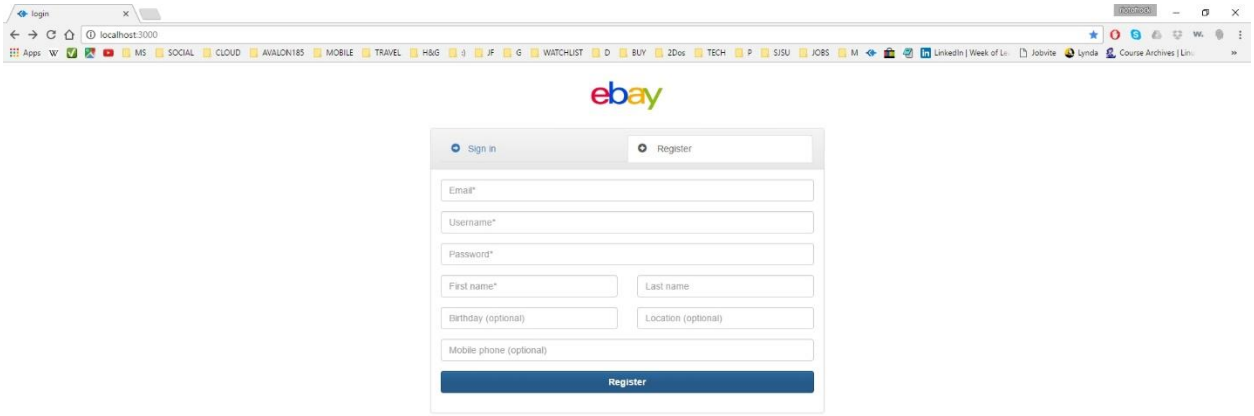


Image description: login page (sign up/register options)

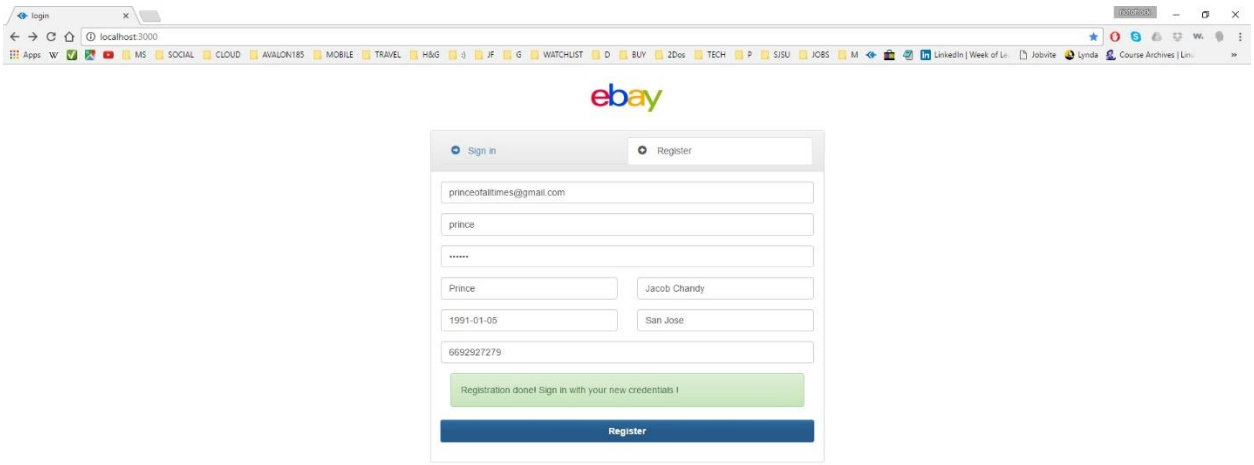


Image description: user registration success

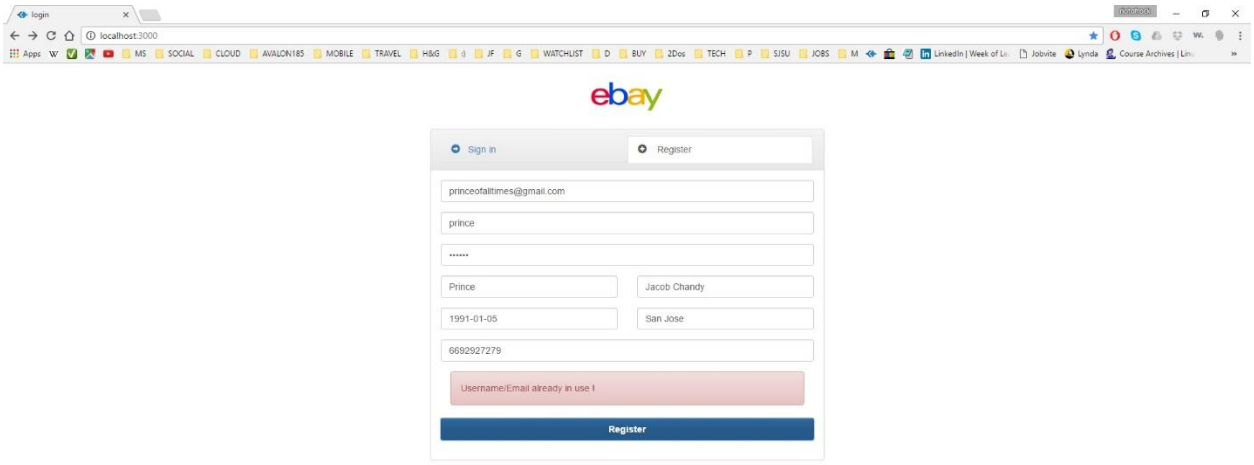


Image description: user registration error – username/email already in use

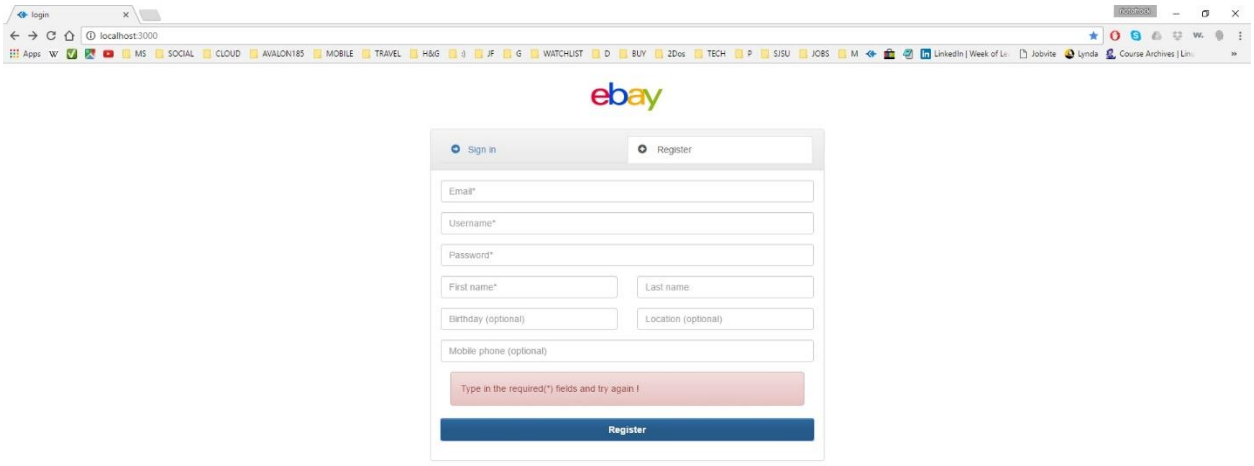


Image description: user registration error – required fields should be filled (required fields are marked by *)

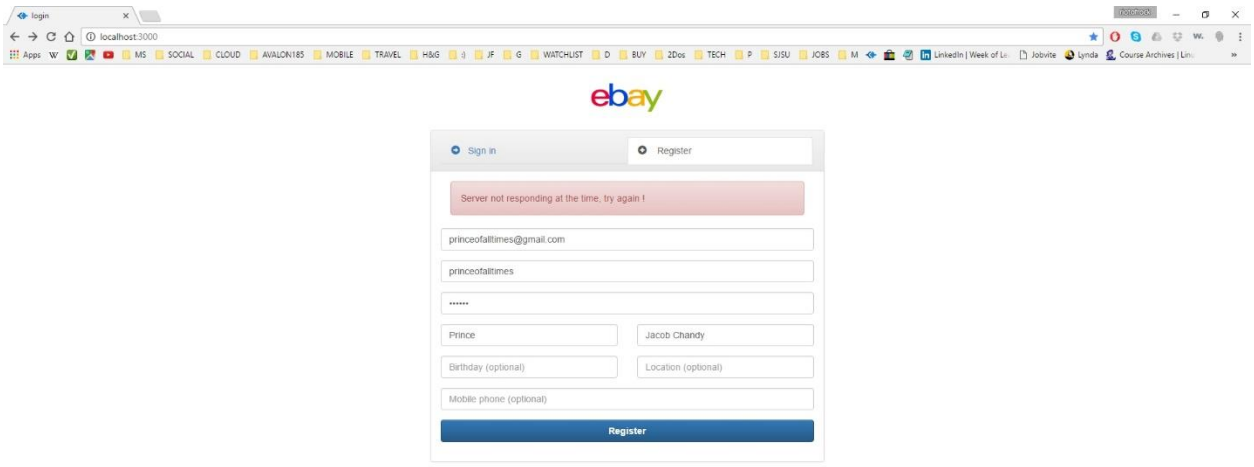


Image description: user registration error – server is not responding

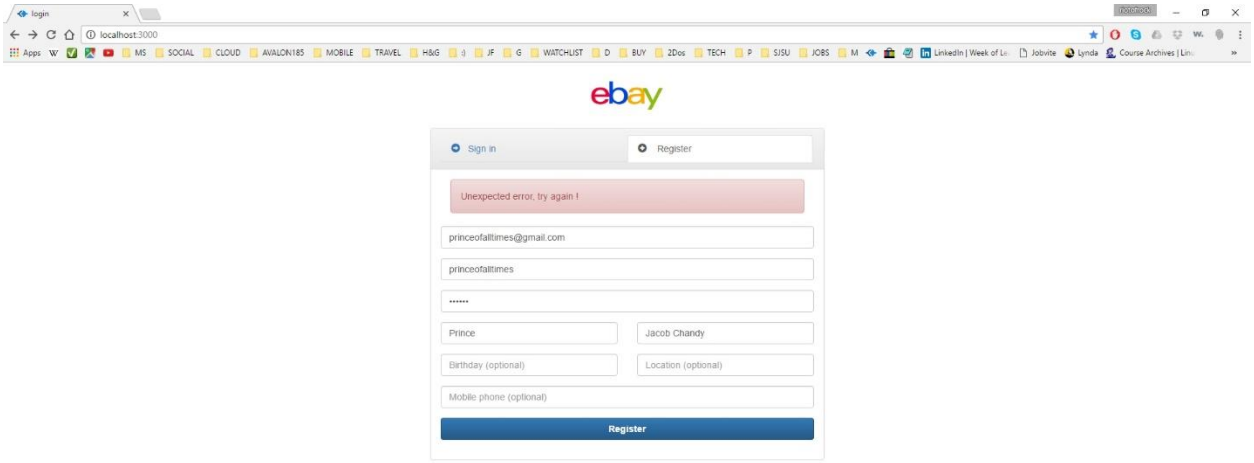


Image description: user registration error – in case of any other error during registration

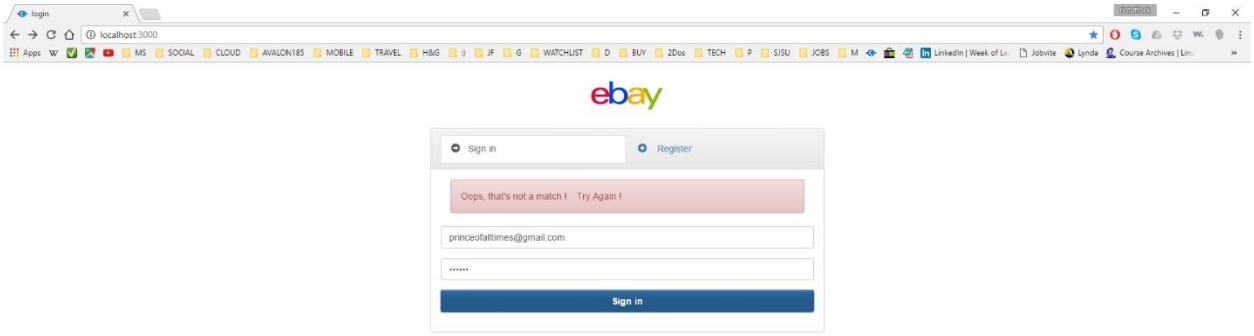


Image description: user login error – username and password is not matching (or not found in database)

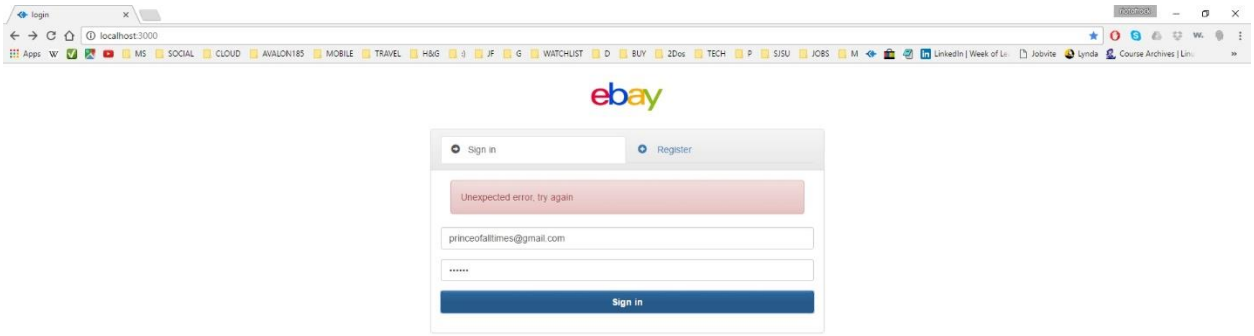


Image description: user login error – server is not responding/server down

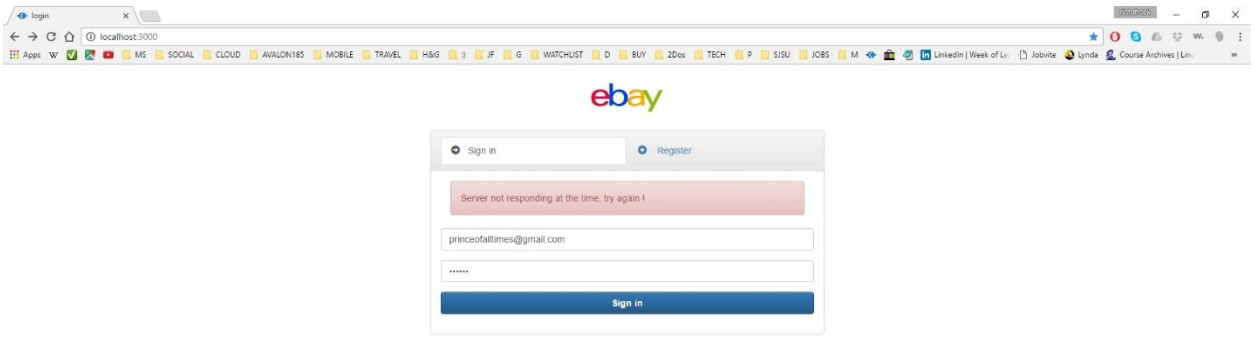


Image description: user login error – in case of any other error while signing in

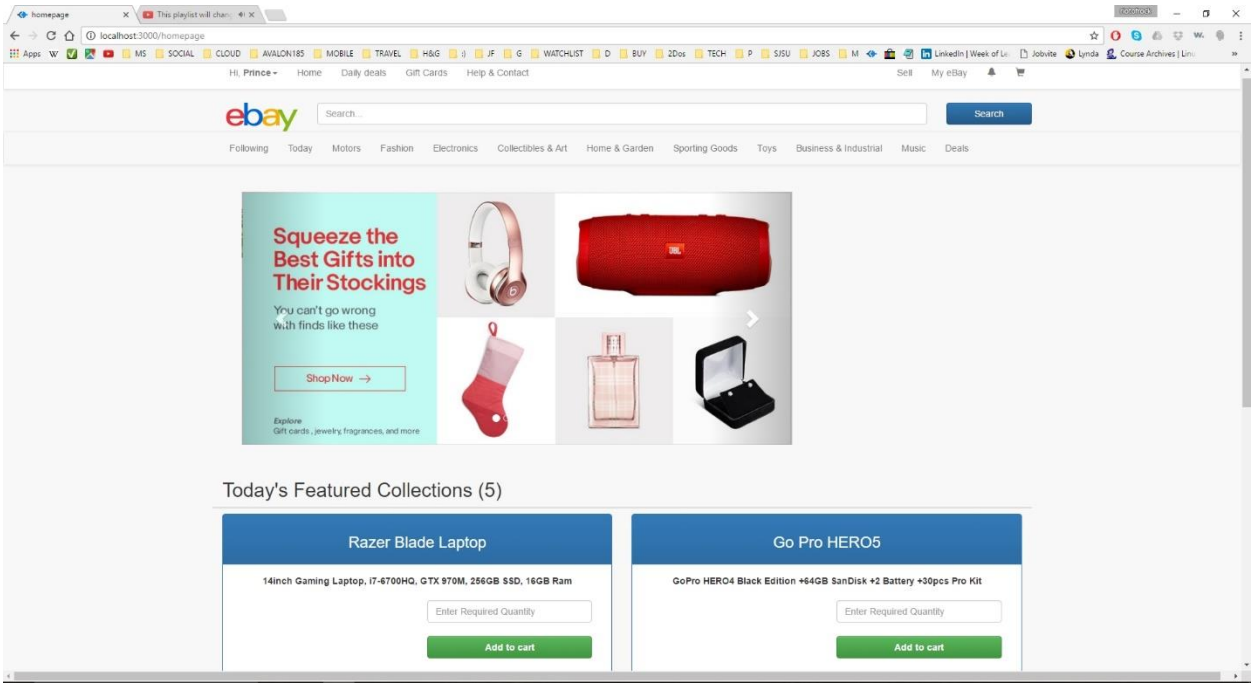


Image description: home page - for first time login

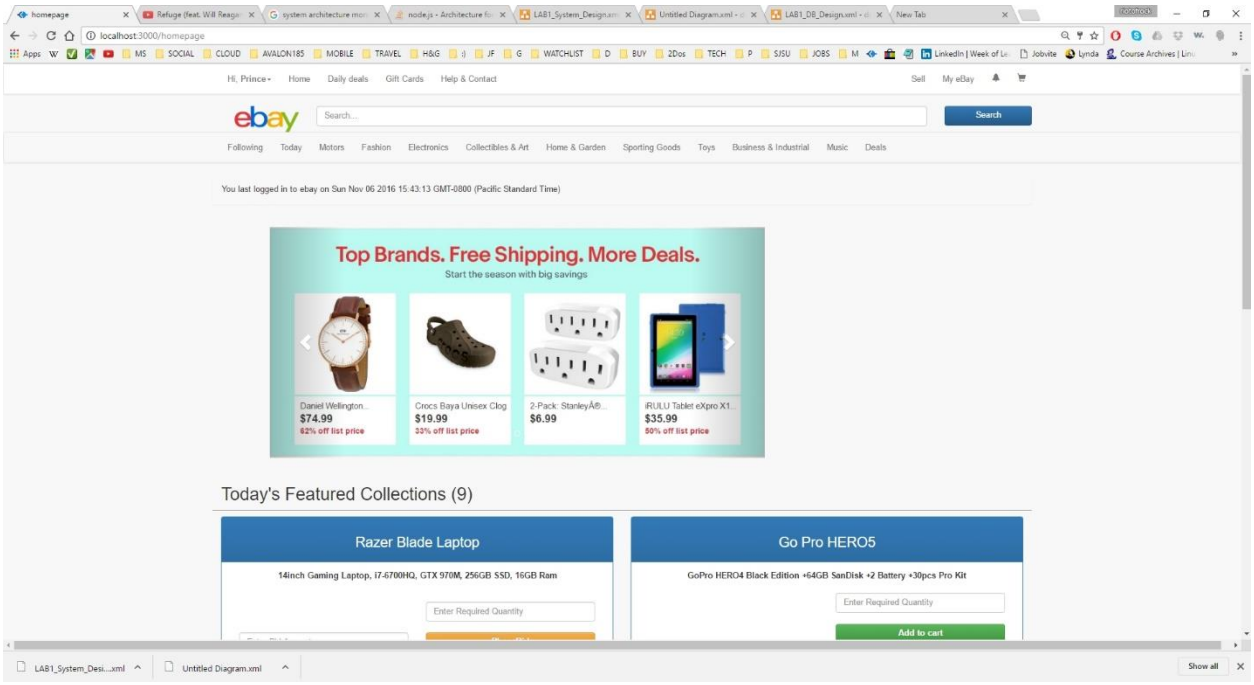


Image description: home page - for successive login (with last time logged in information)

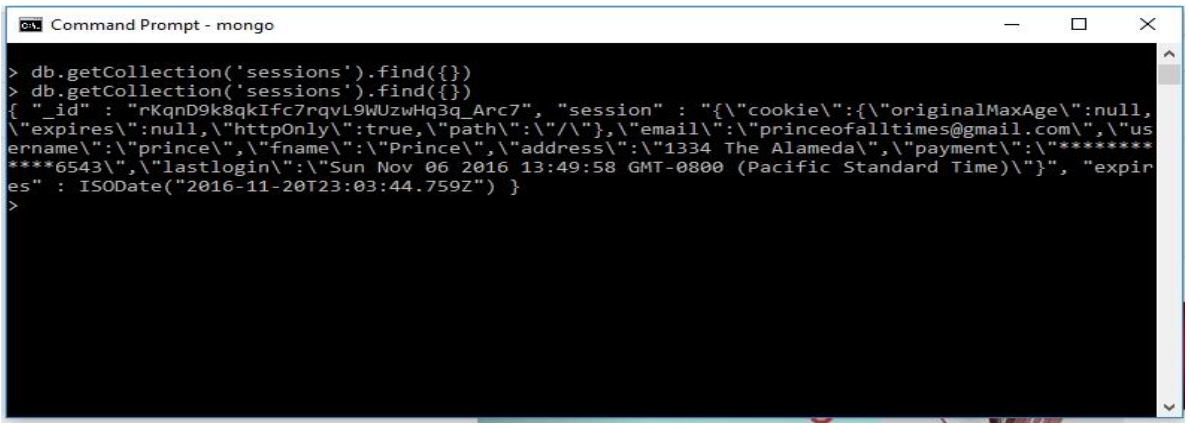


Image description: session information stored in mongoDB (1)

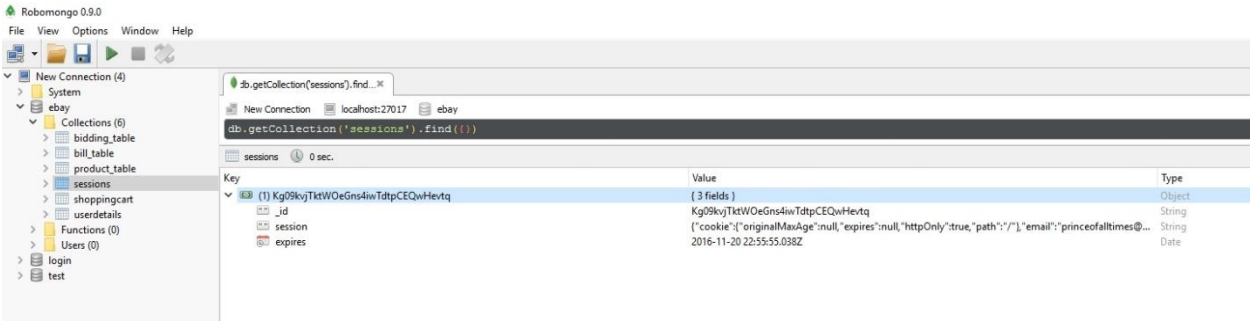


Image description: session information stored in mongoDB (2)

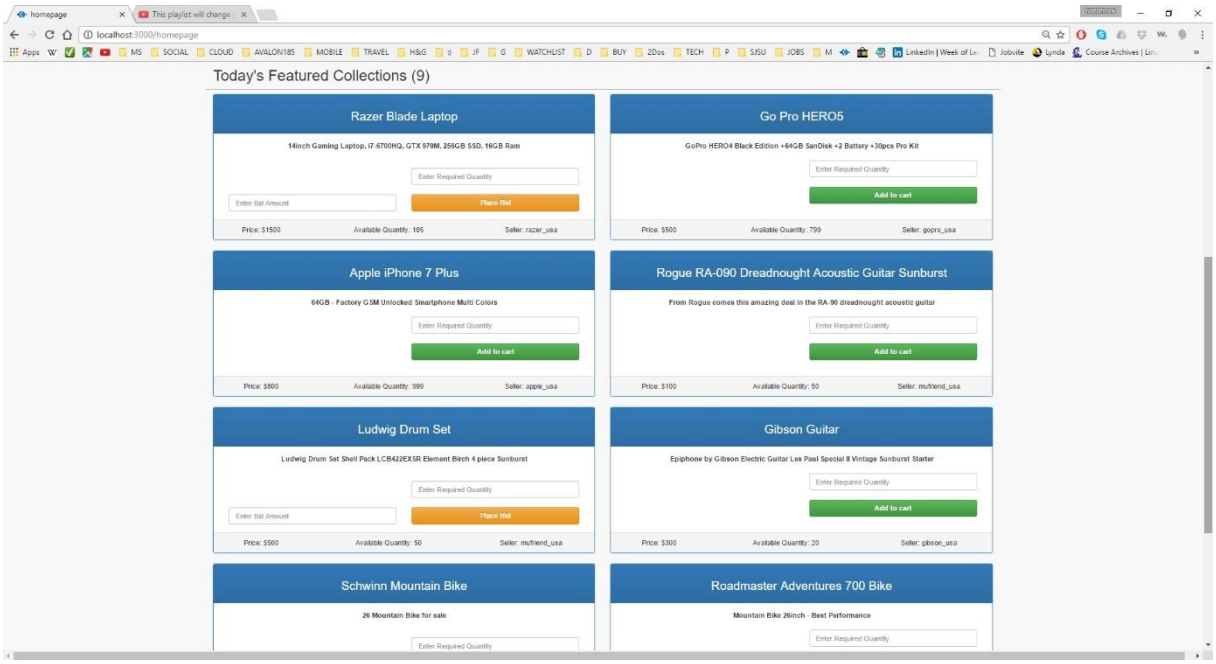


Image description: home page – advertisements posted by other users are listed (1); products sold out (available quantity=0) and products whose bid end time is over won't be listed

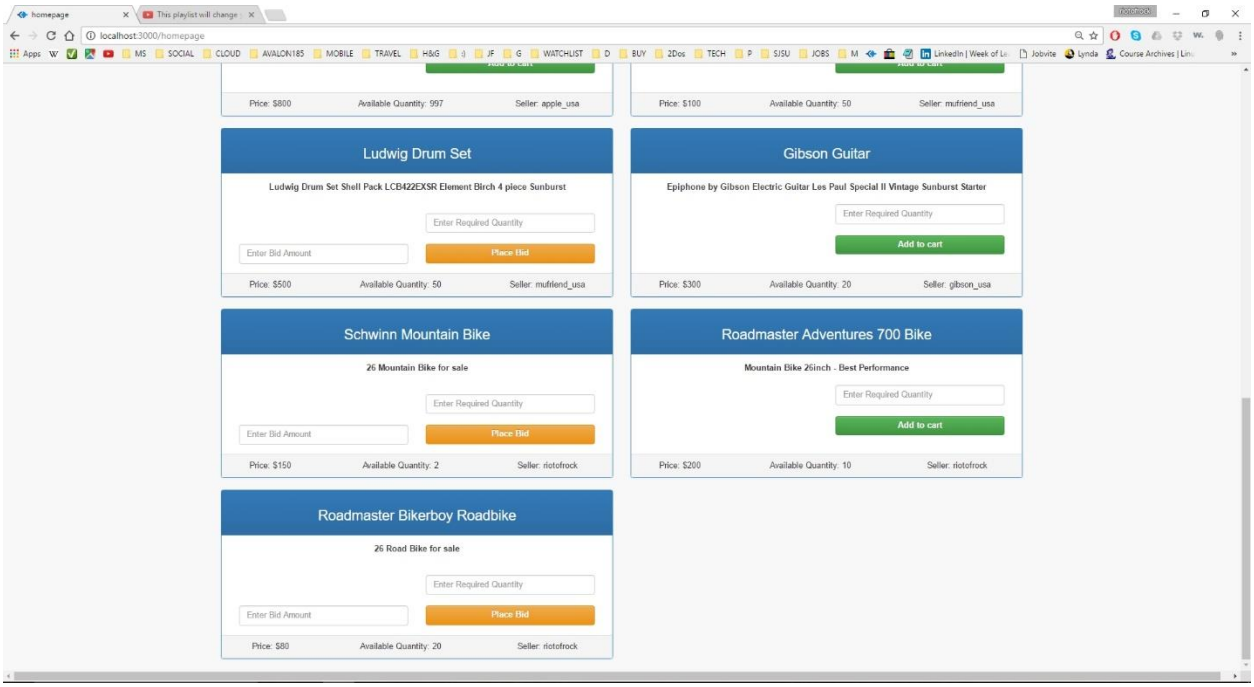


Image description: home page – advertisements posted by other users are listed (2); products sold out (available quantity=0) and products whose bid end time is over won't be listed

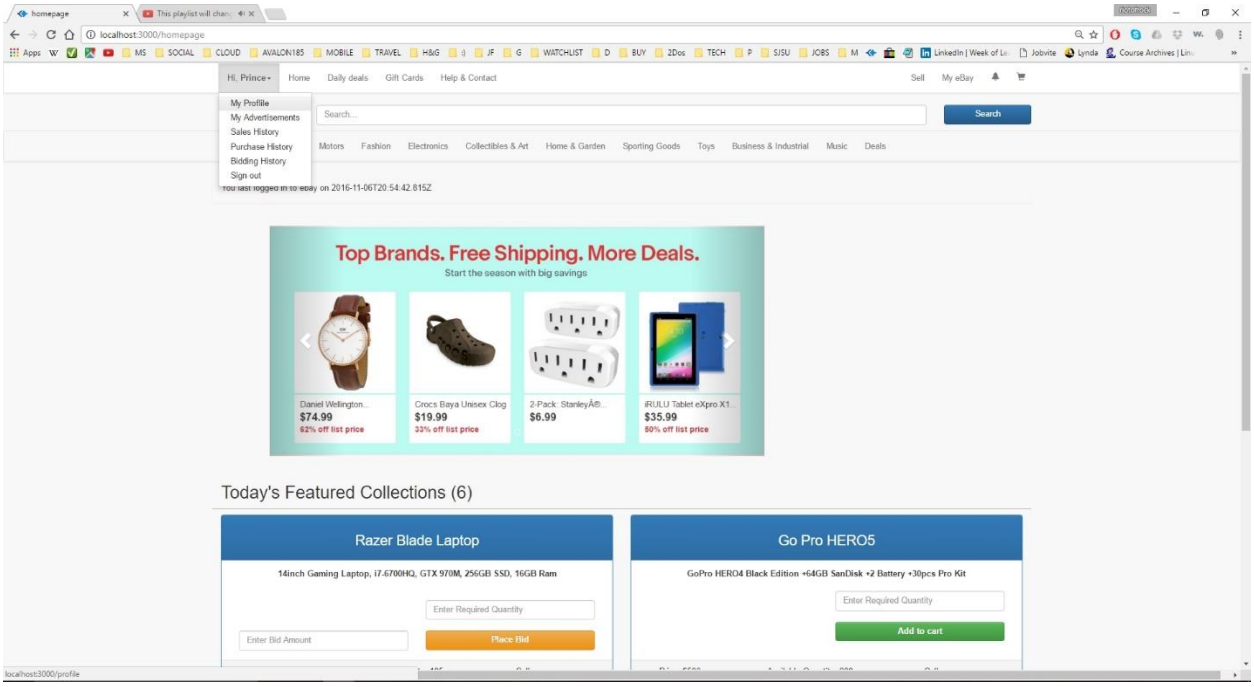


Image description: home page – drop down options listed in the main header (functionalities)

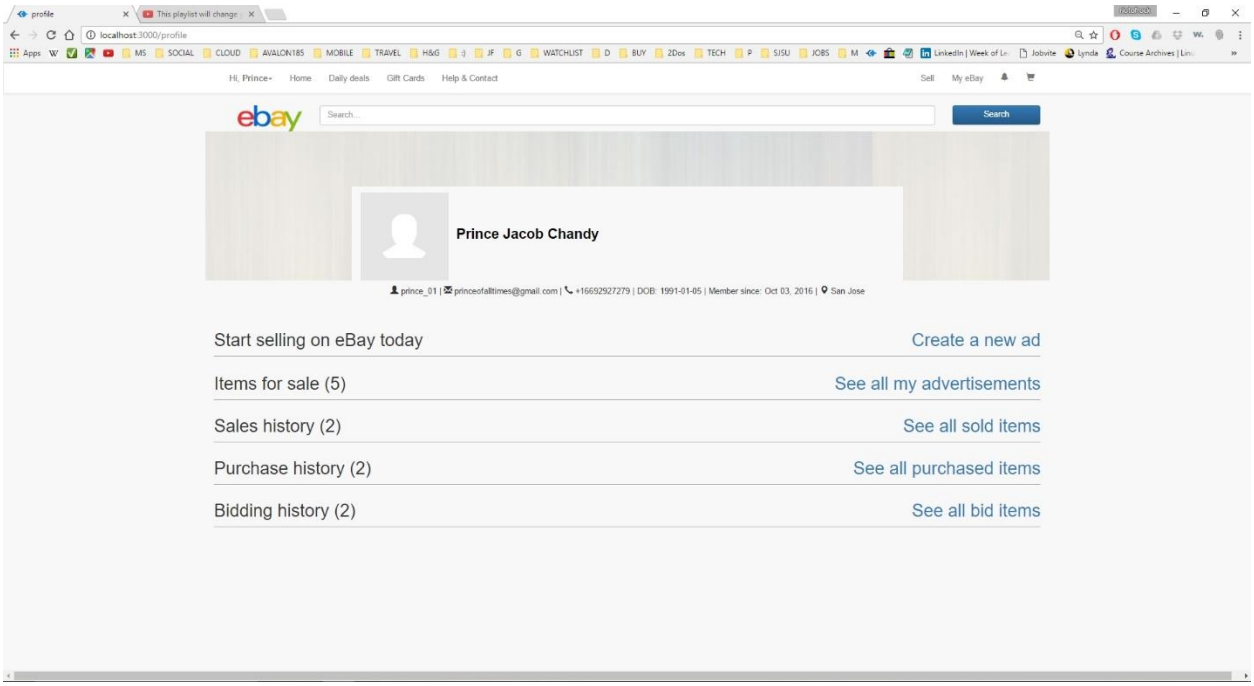


Image description: profile page – displays profile information and purchase/sale history; also allows user to create his own advertisements (normal and bid)

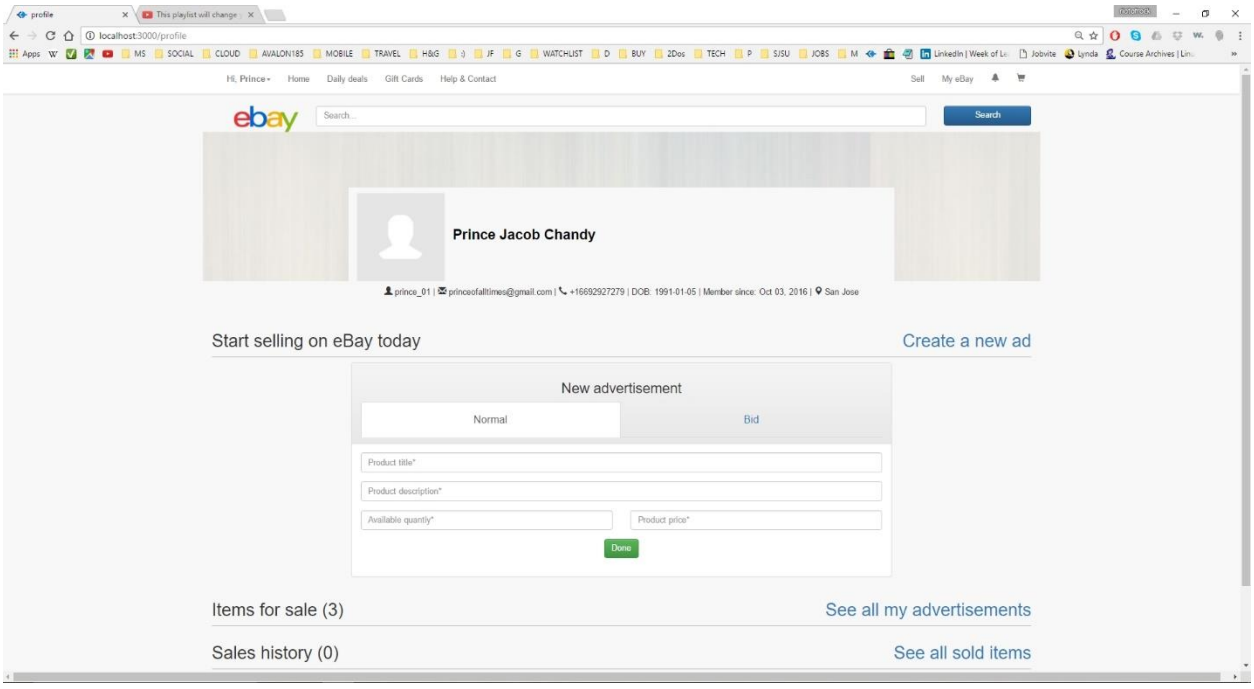


Image description: profile page – create a new normal advertisement (1) - required fields are marked by * and an ad can be created only if all the required information is filled

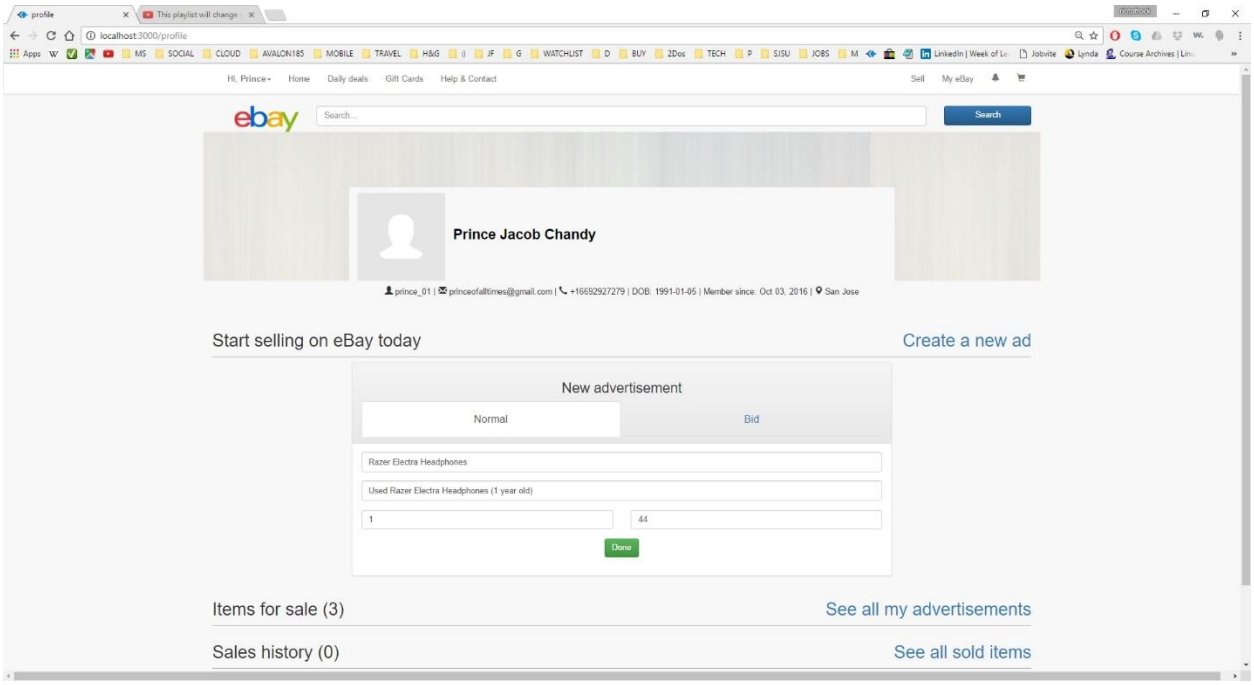


Image description: profile page – create a new normal advertisement (2)

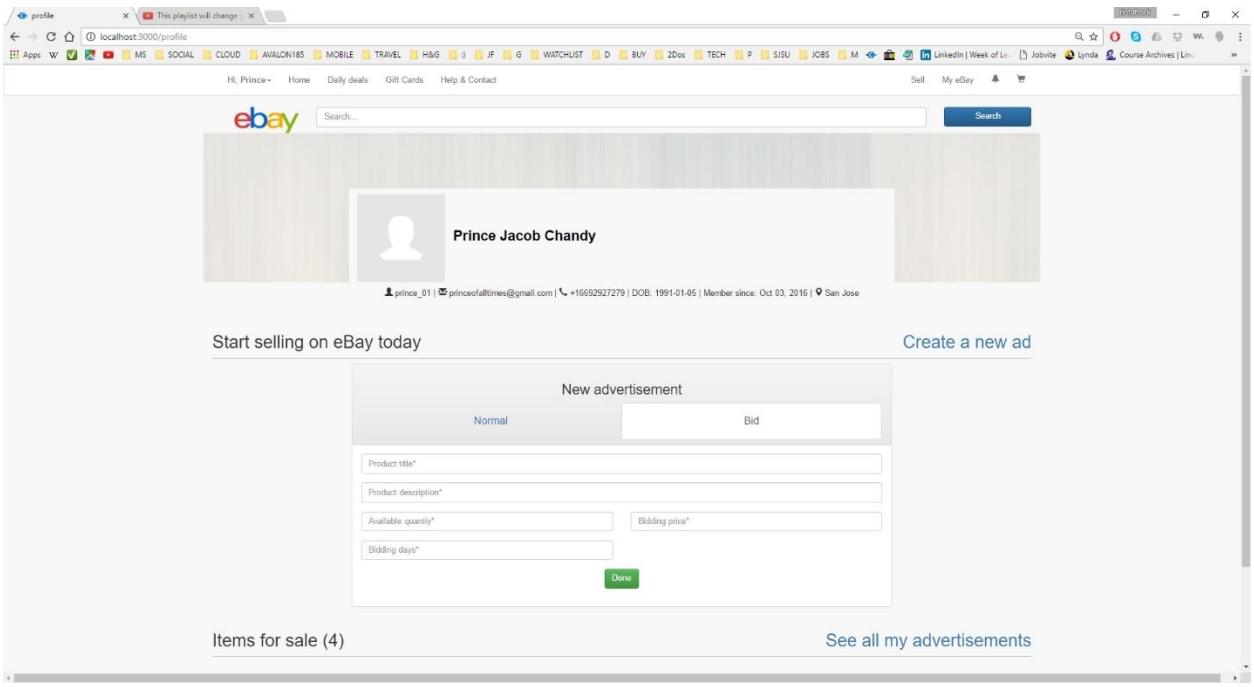


Image description: profile page – create a new bidding advertisement (1) - required fields are marked by * and an ad can be created only if all the required information is filled

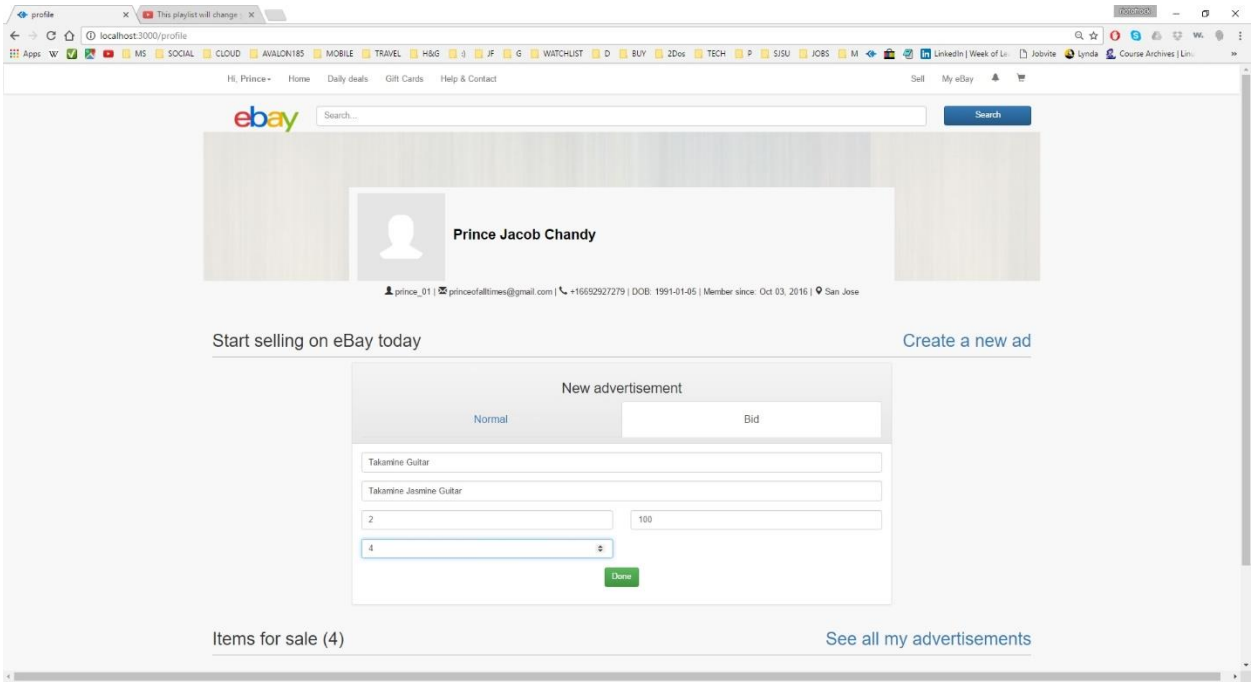


Image description: profile page – create a new bidding advertisement (2)

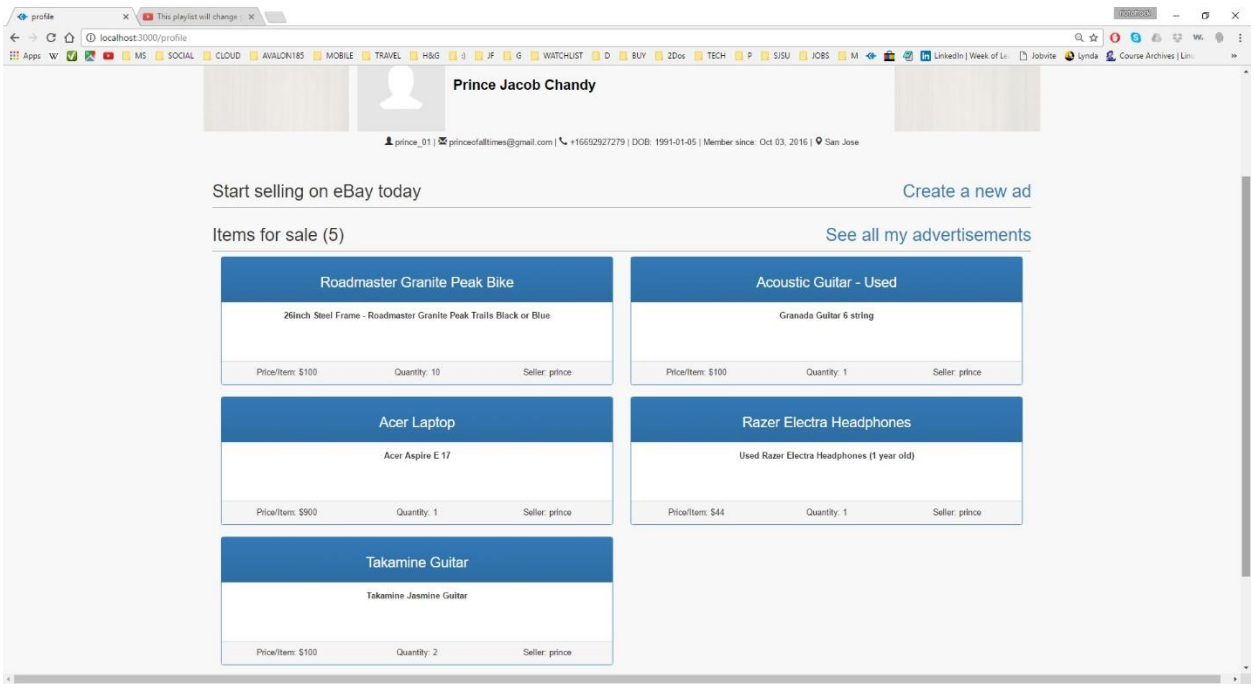


Image description: profile page – see all advertisements created by the user

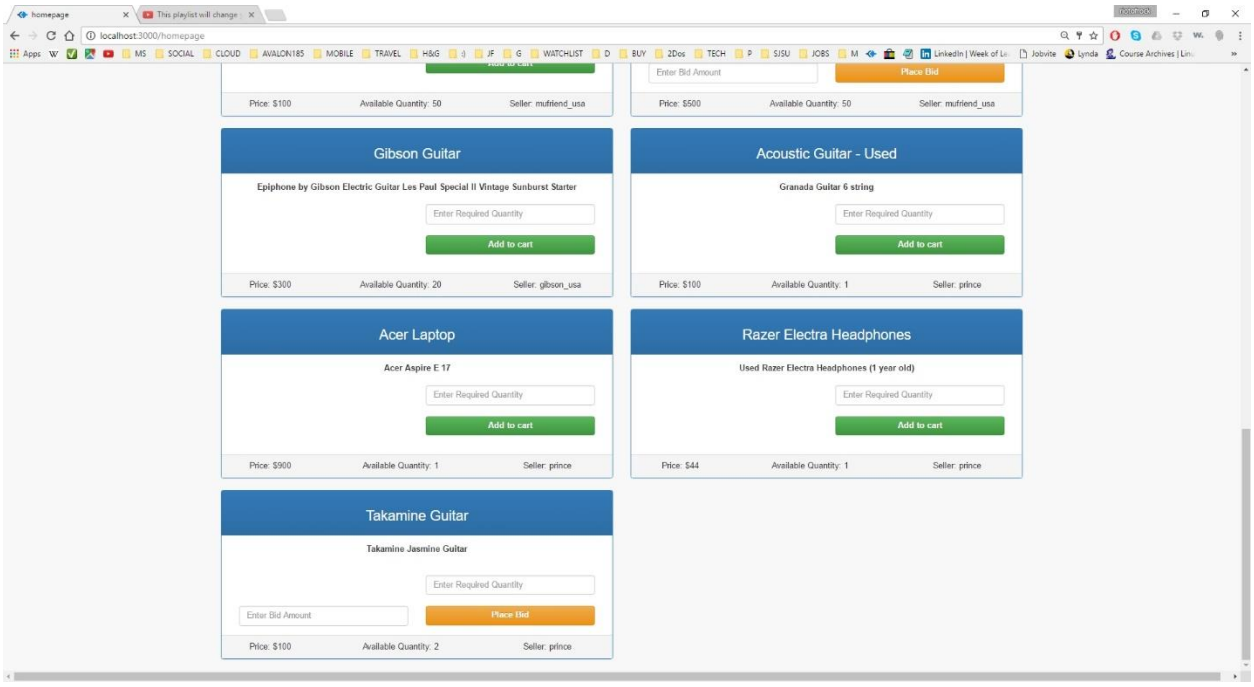


Image description: home page – newly created advertisements created as seen in another user's homepage

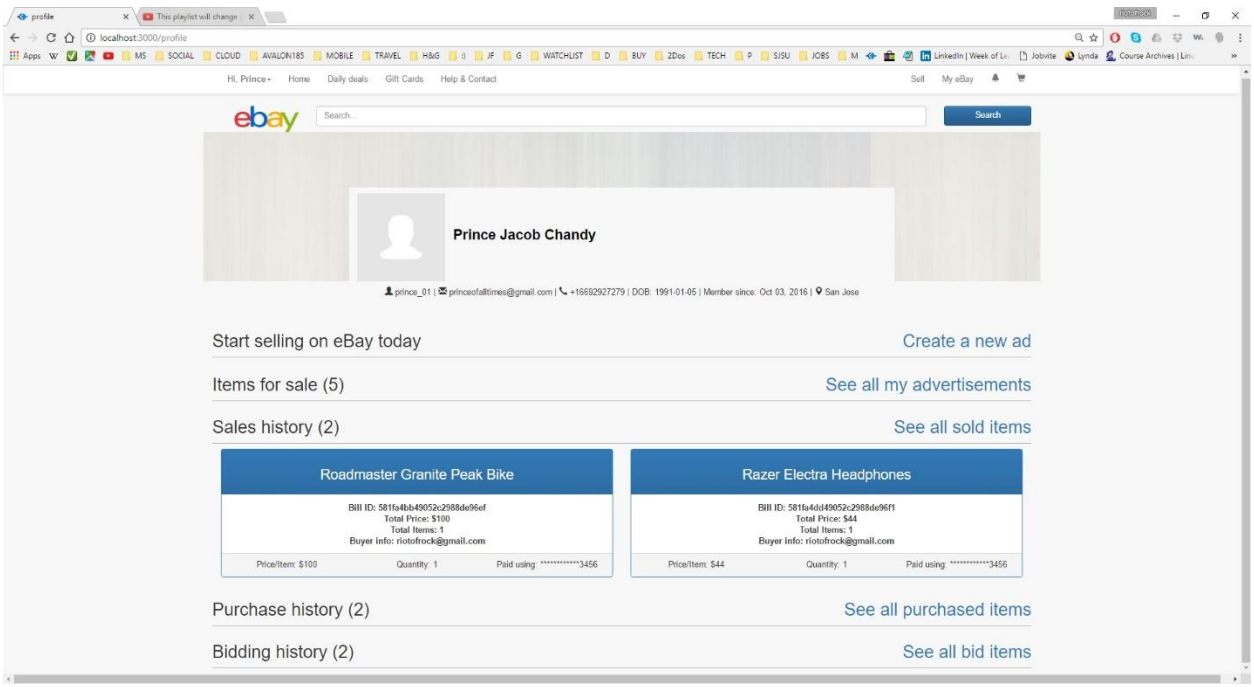


Image description: profile page – sales history of current user's products with buyer details and billing information

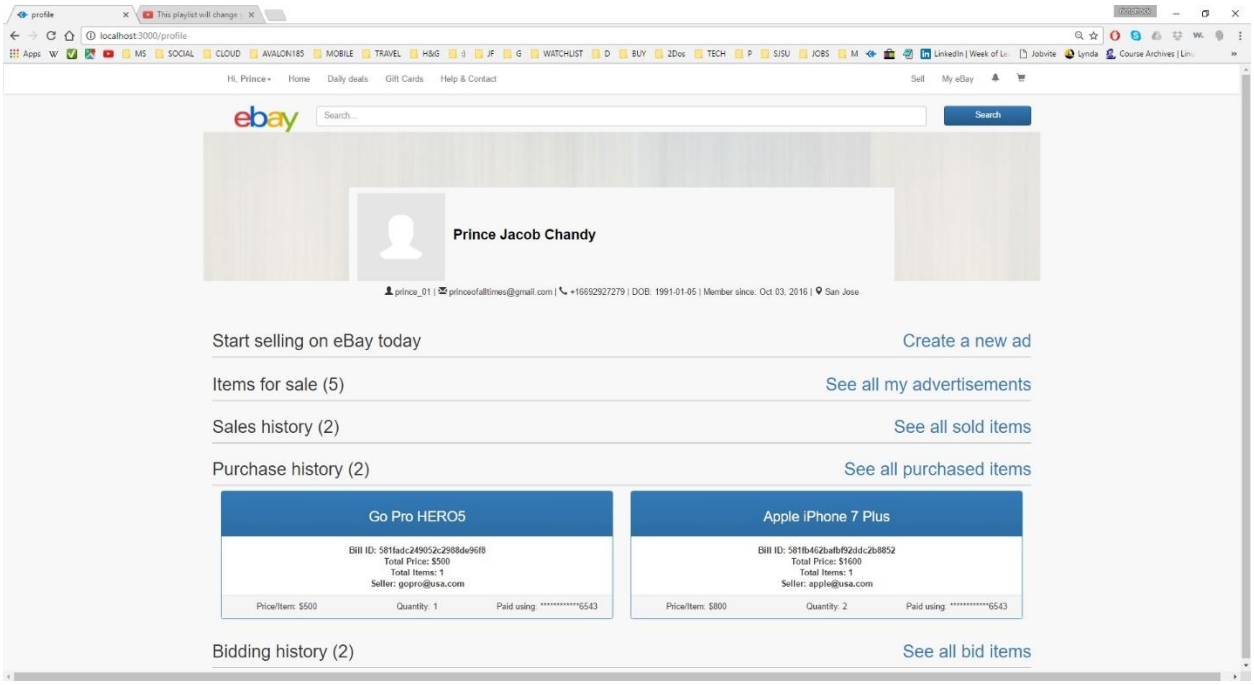


Image description: profile page – purchase history of current user with seller details and billing information

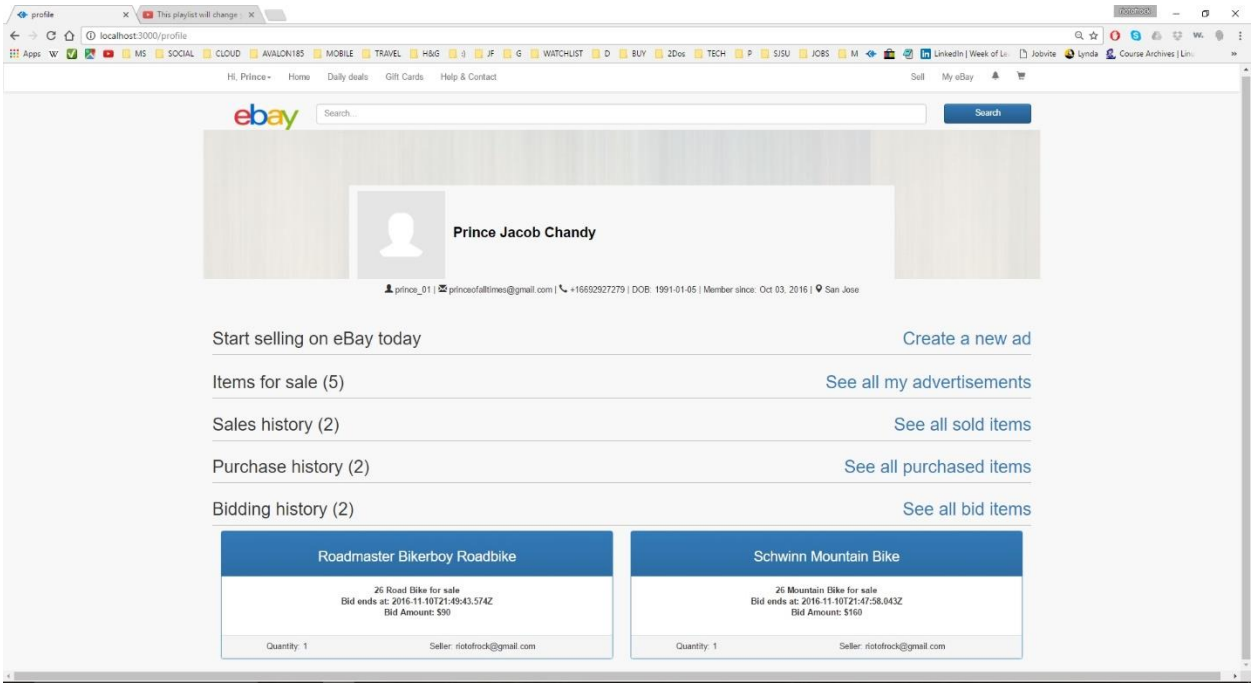


Image description: profile page – bidding history of current user with seller details and bidding information; list all the products bid by the user which is active

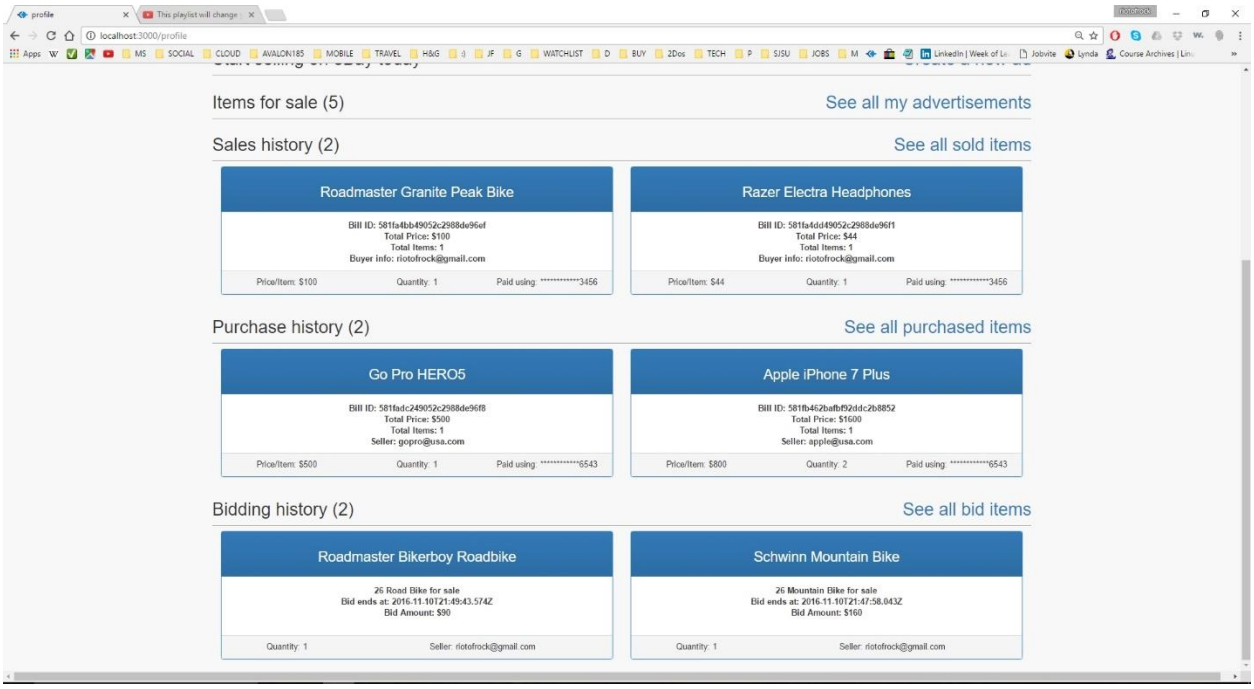


Image description: profile page – sales history, purchase history, bidding history

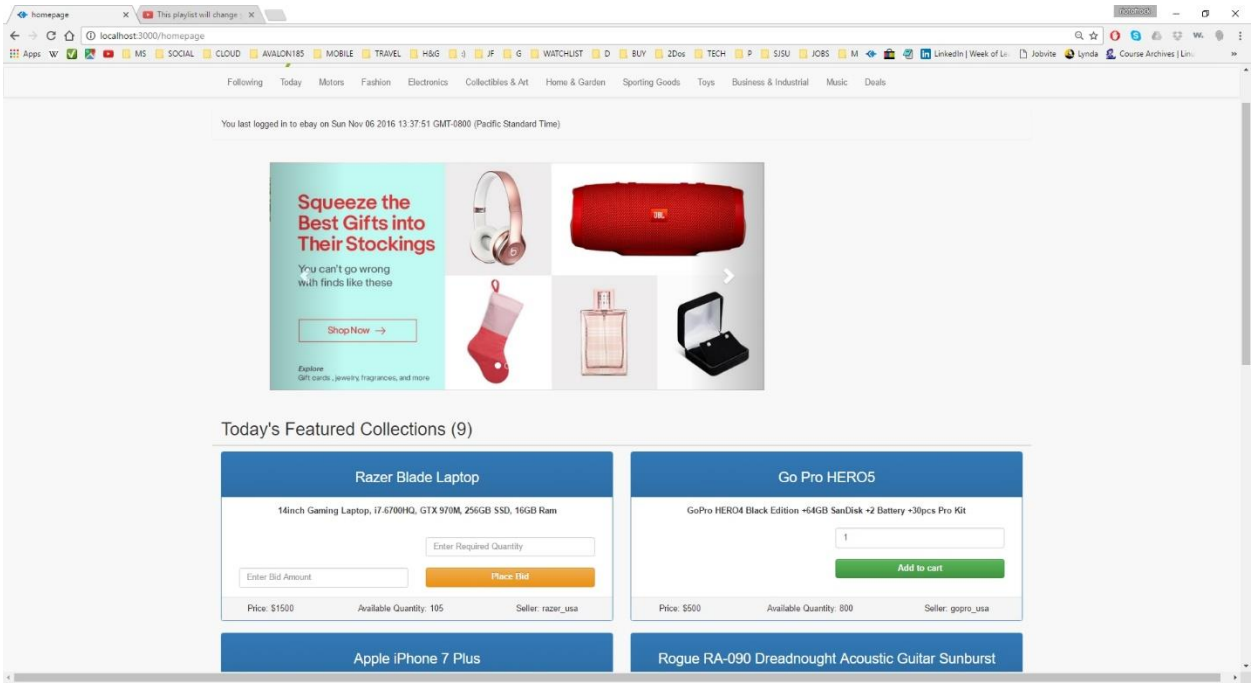


Image description: home page – buying a normal product by entering required quantity and clicking add to cart; add to cart button will proceed to cart only if the entered quantity is less than available quantity

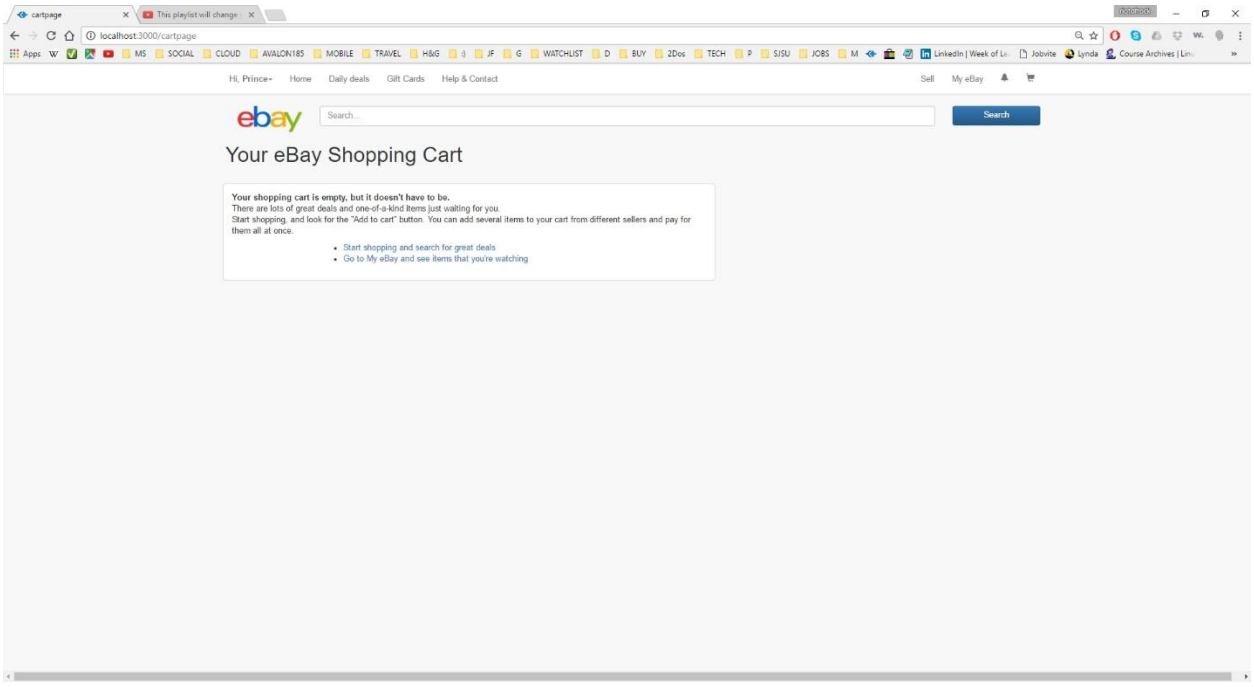


Image description: shopping cart page – when empty; Items in shopping cart when added will be added to session as well as database so that user gets his shopping cart information when he returns later after logout

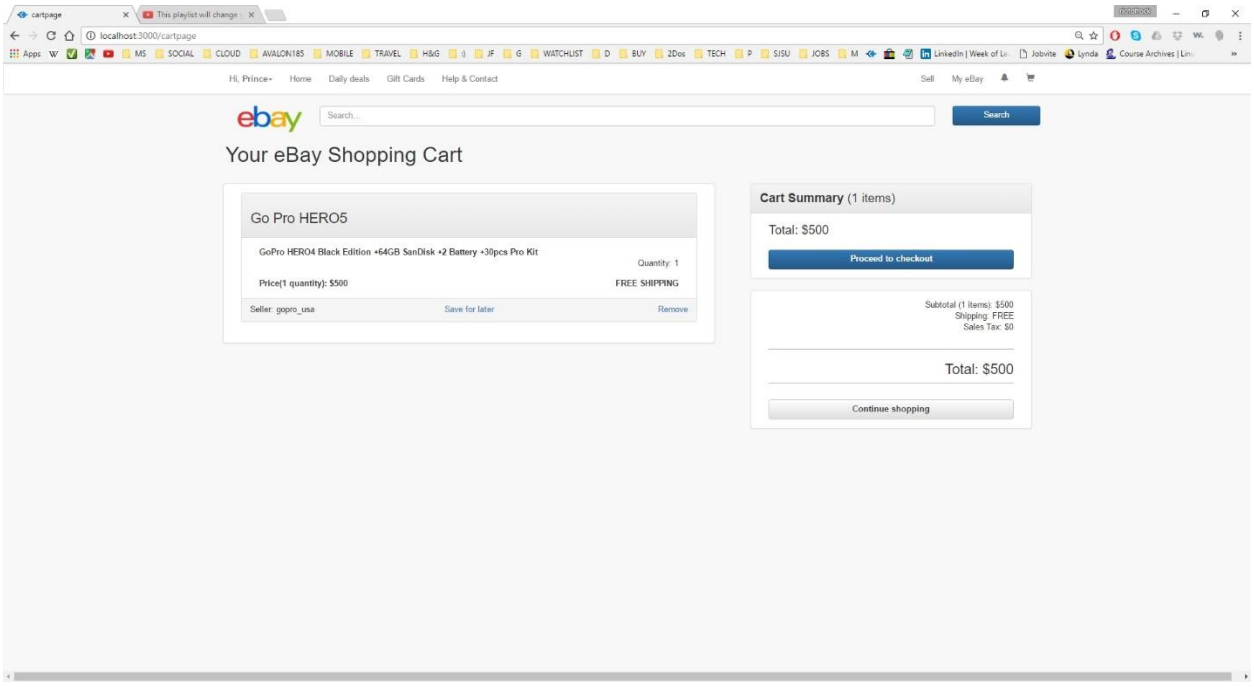


Image description: shopping cart page - all the items added by the user from homepage is listed here. Pressing add to cart from home page routes to shopping cart page. Can access this page also by clicking at the cart icon at the extreme top left

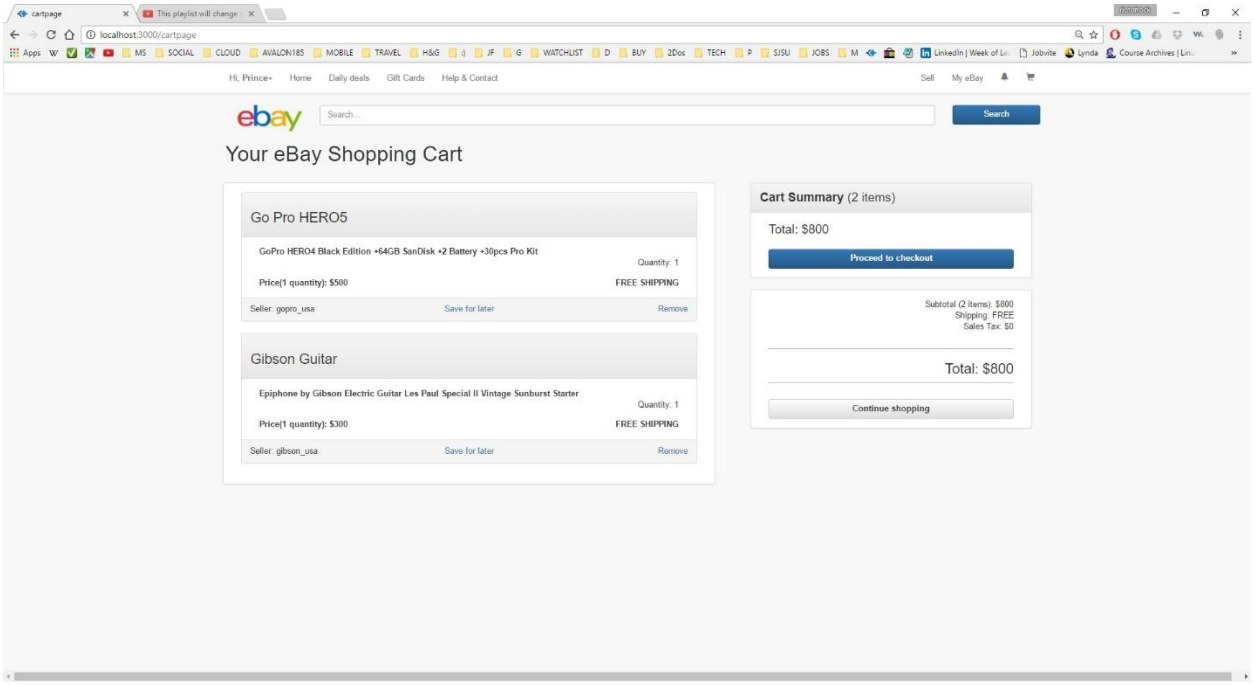


Image description: shopping cart page – with multiple items and total amount, items details; Note that delete option is provided for all items to remove them from the cart. Continue shopping routes back to the homepage

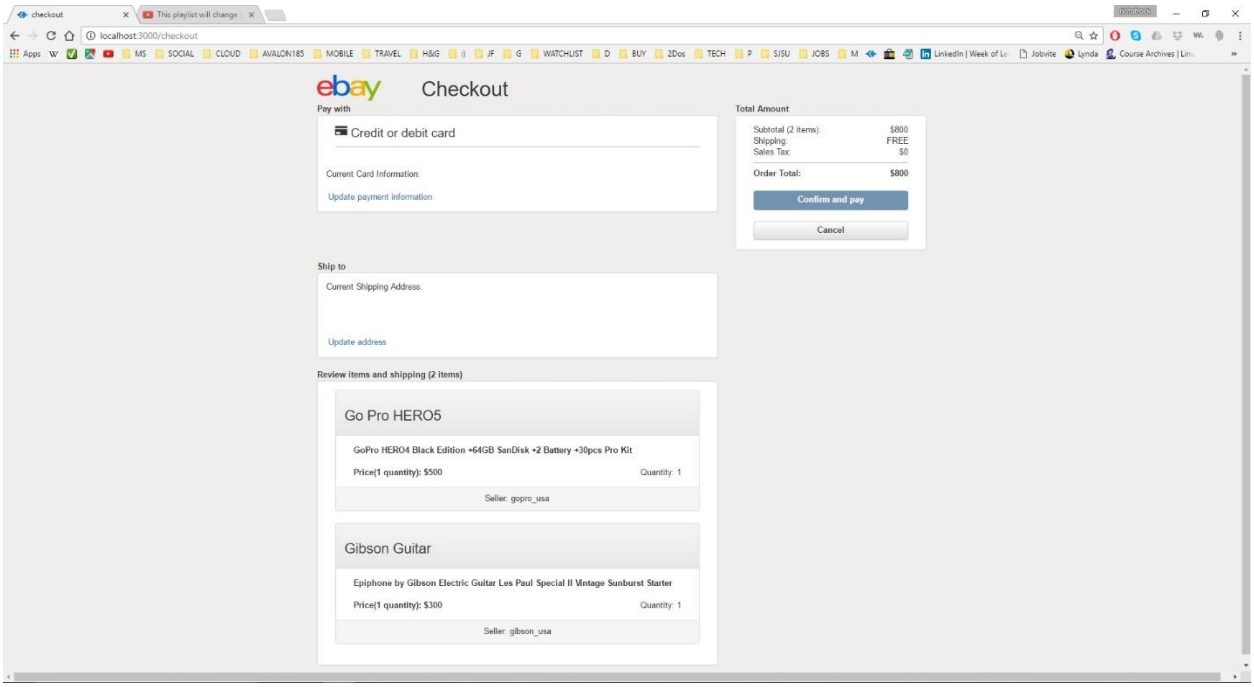


Image description: checkout page – user can provide (add/update) payment information and shipping address information. Confirm and pay button will be active only after entering valid creditcard information and shipping address

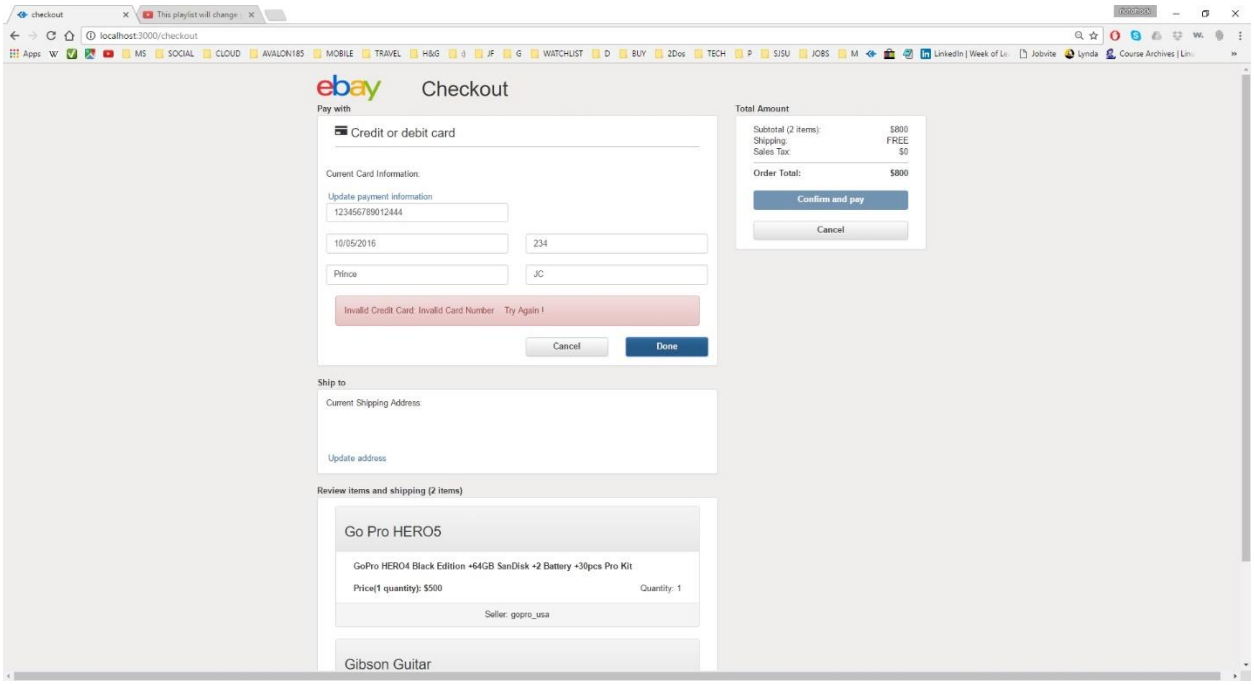


Image description: checkout page – payment/credit card validation (1)

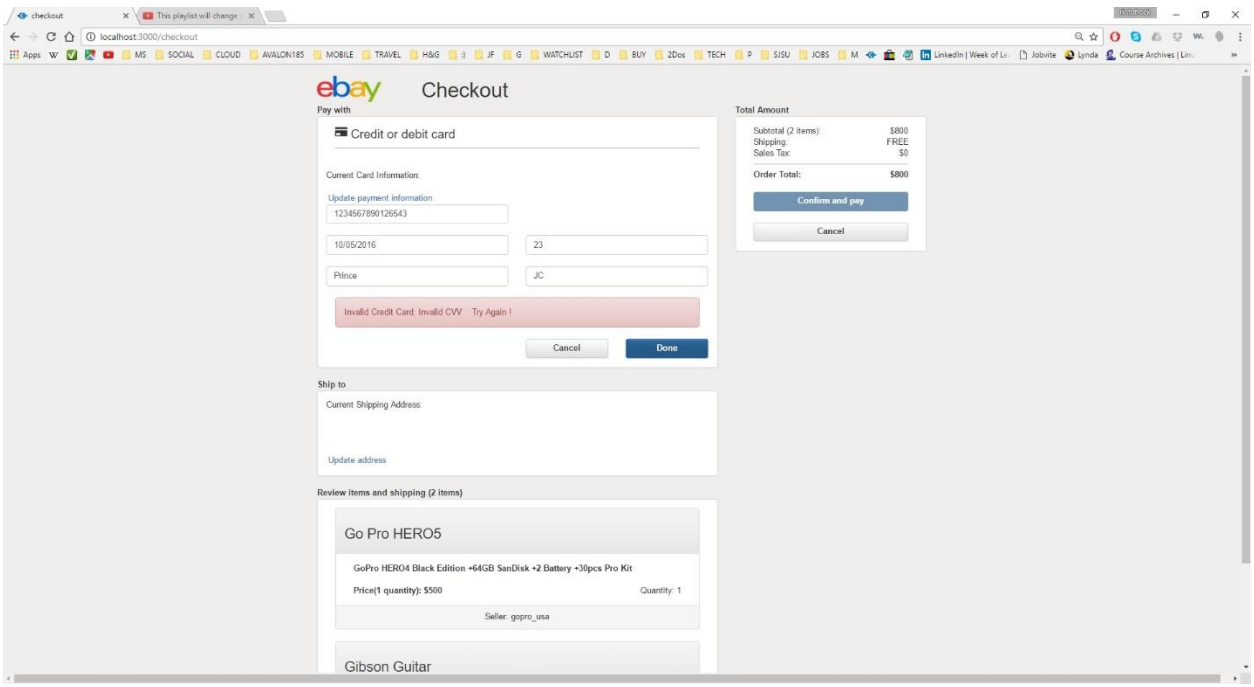


Image description: checkout page – payment/credit card validation (2)

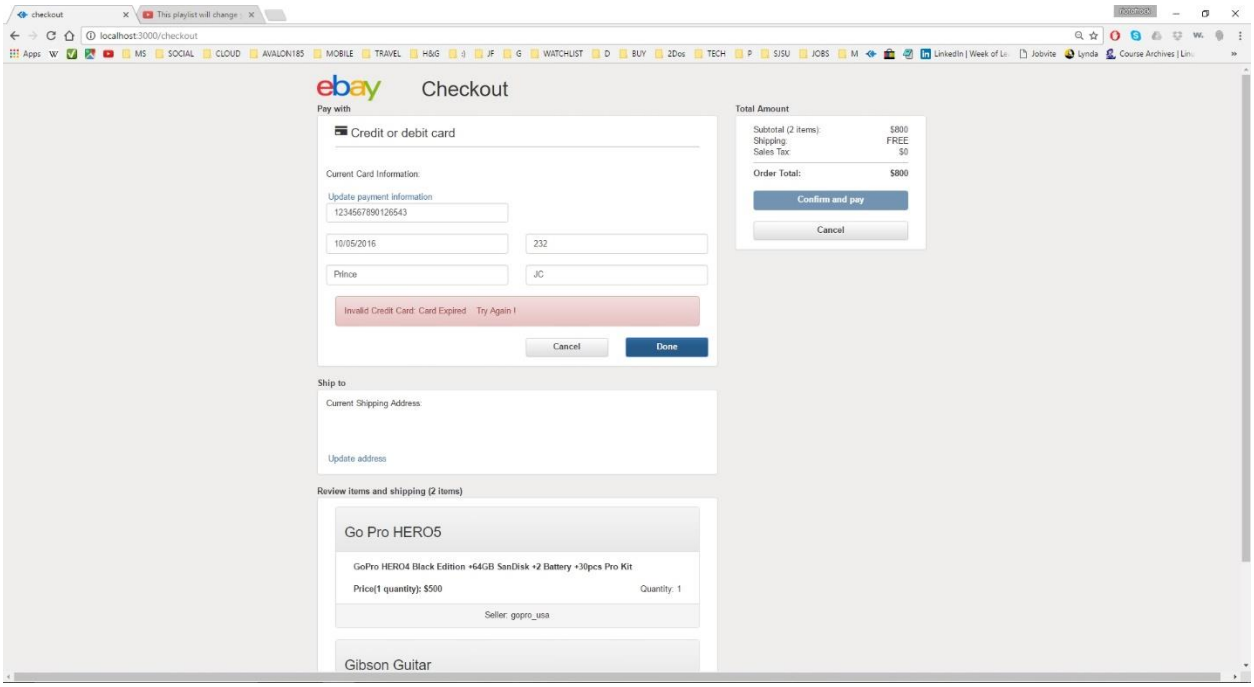


Image description: checkout page – payment/credit card validation (3)

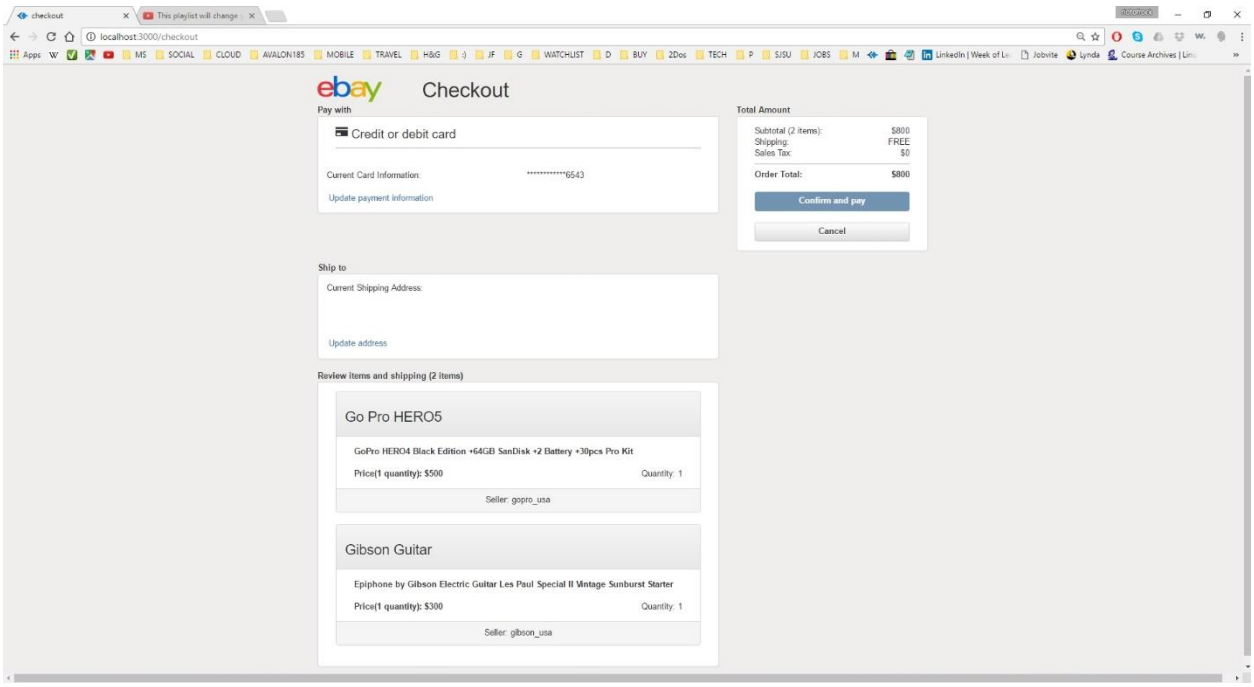


Image description: checkout page – payment/credit card validation success and information have been updated in database

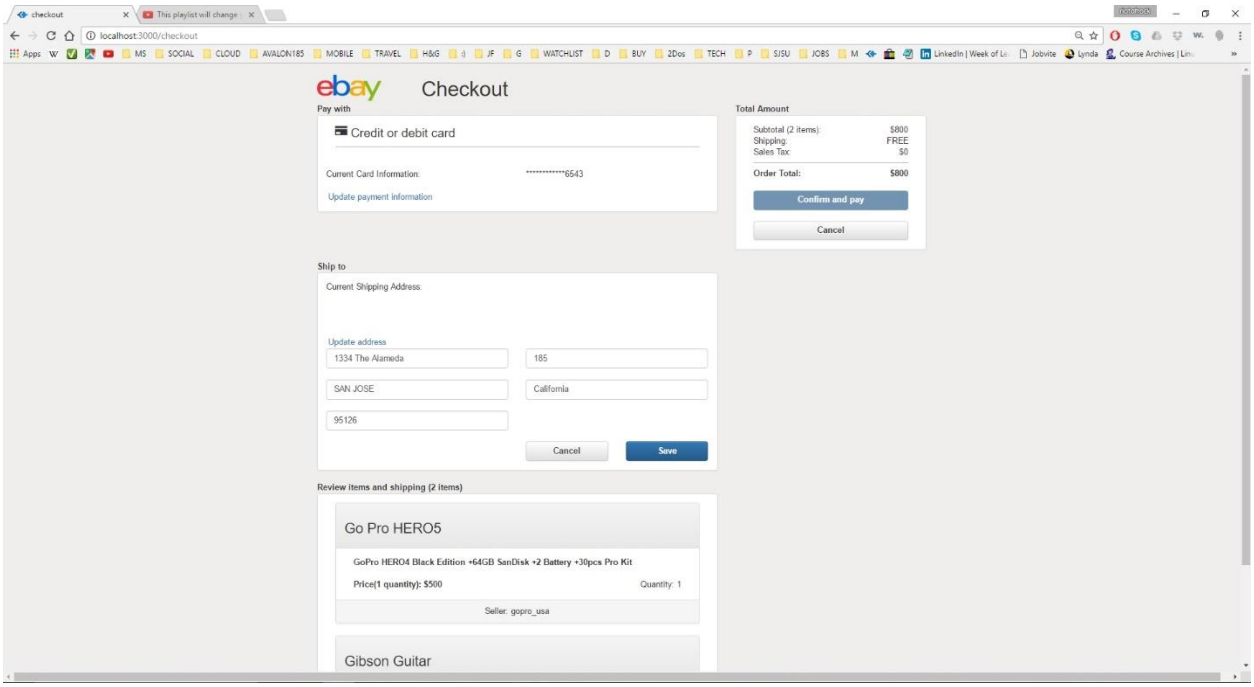


Image description: checkout page – shipping address addition/update

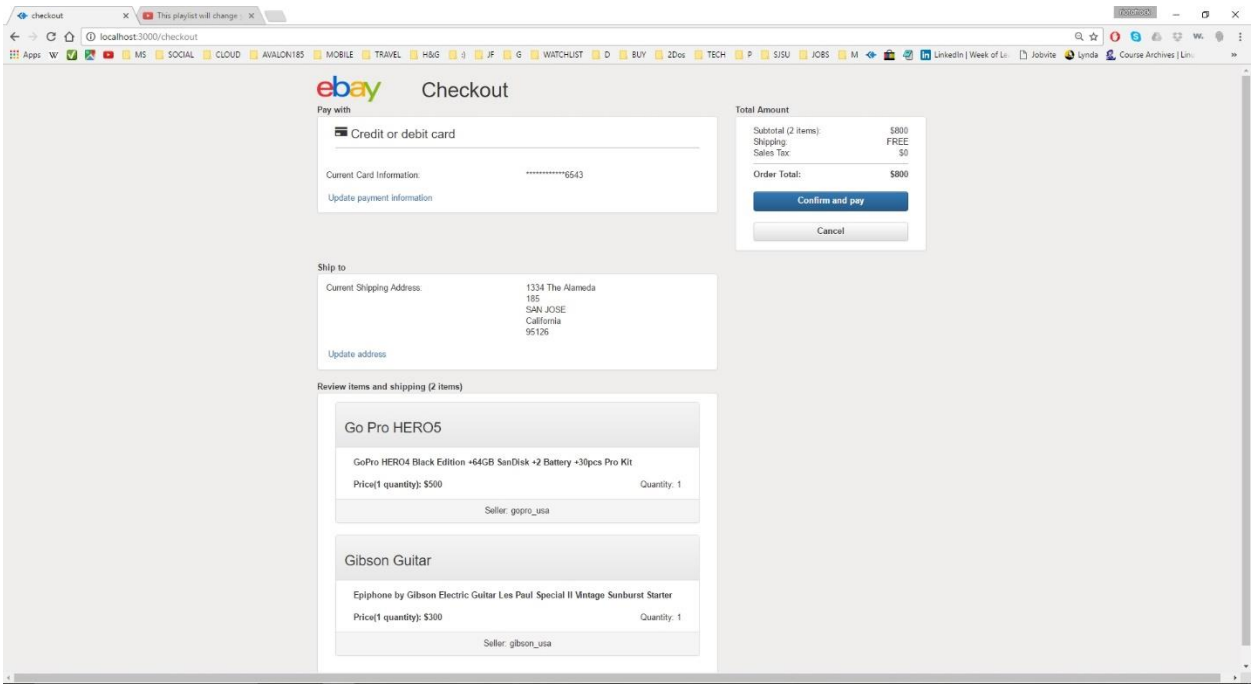


Image description: checkout page – confirm and pay button active since now we have valid payment and shipping information; checkout checks one final time in the database where the product and quantity in the cart is available and proceed to successful checkout page if available, or else to fail checkout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

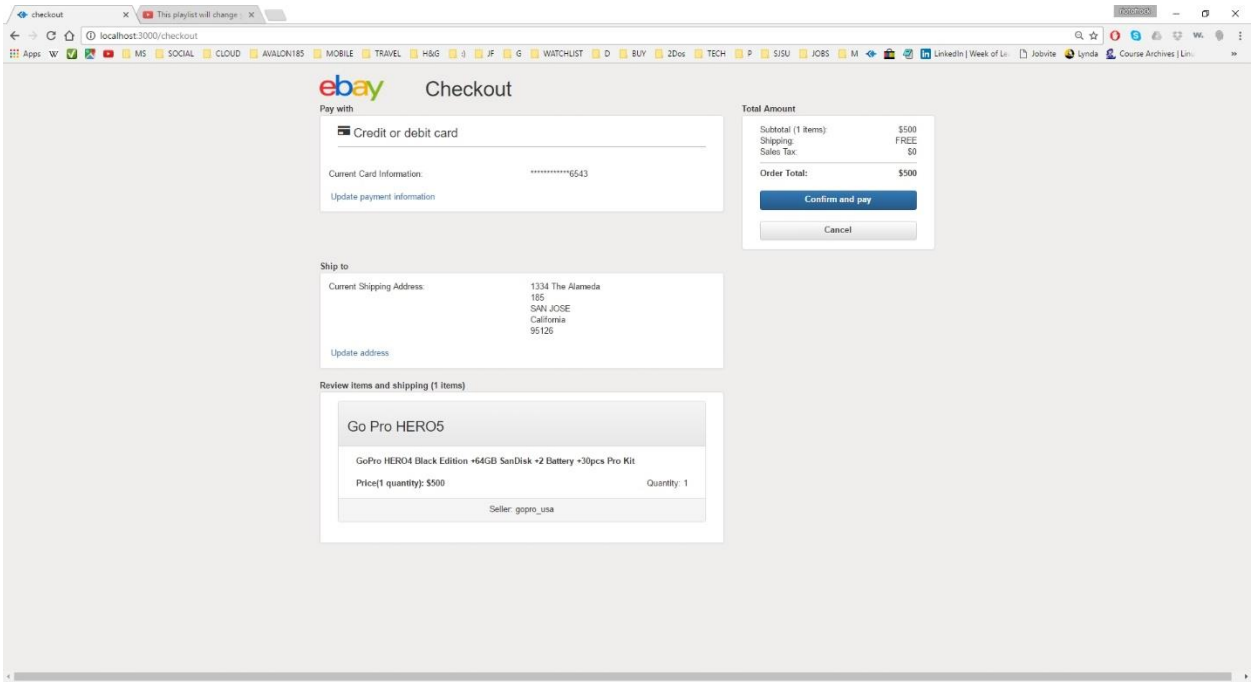


Image description: checkout page with single item in cart

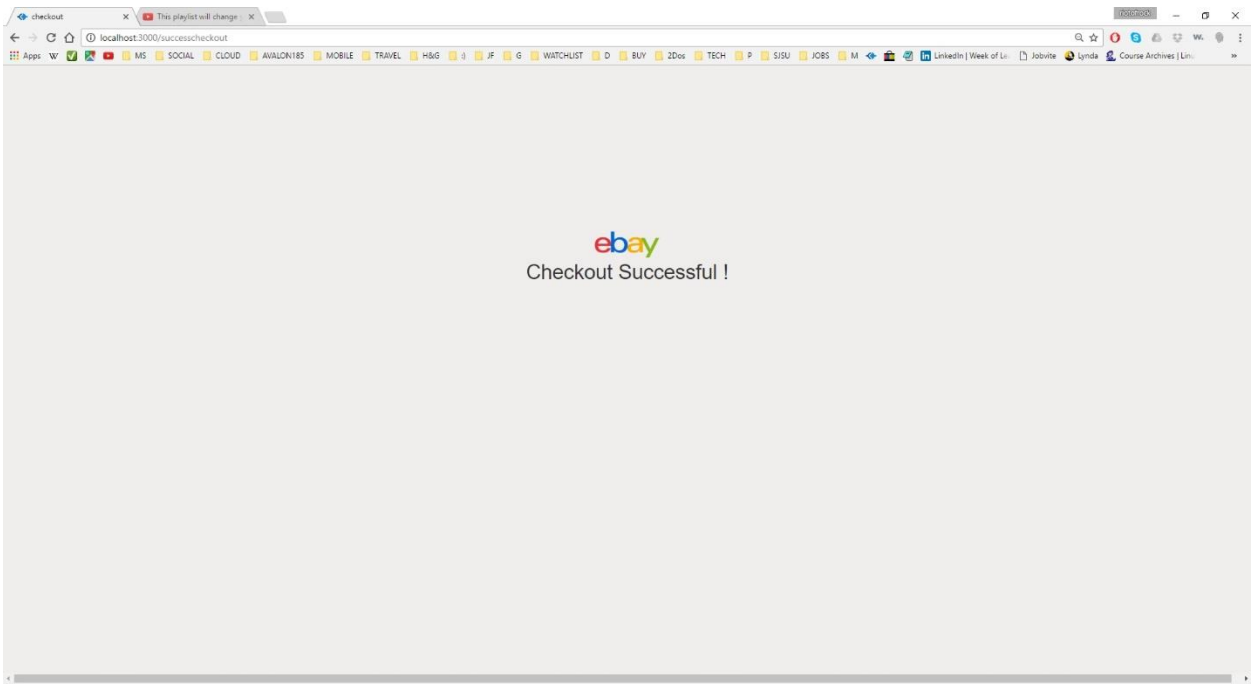


Image description: checkout successful page; can return to main page by clicking on ebay image

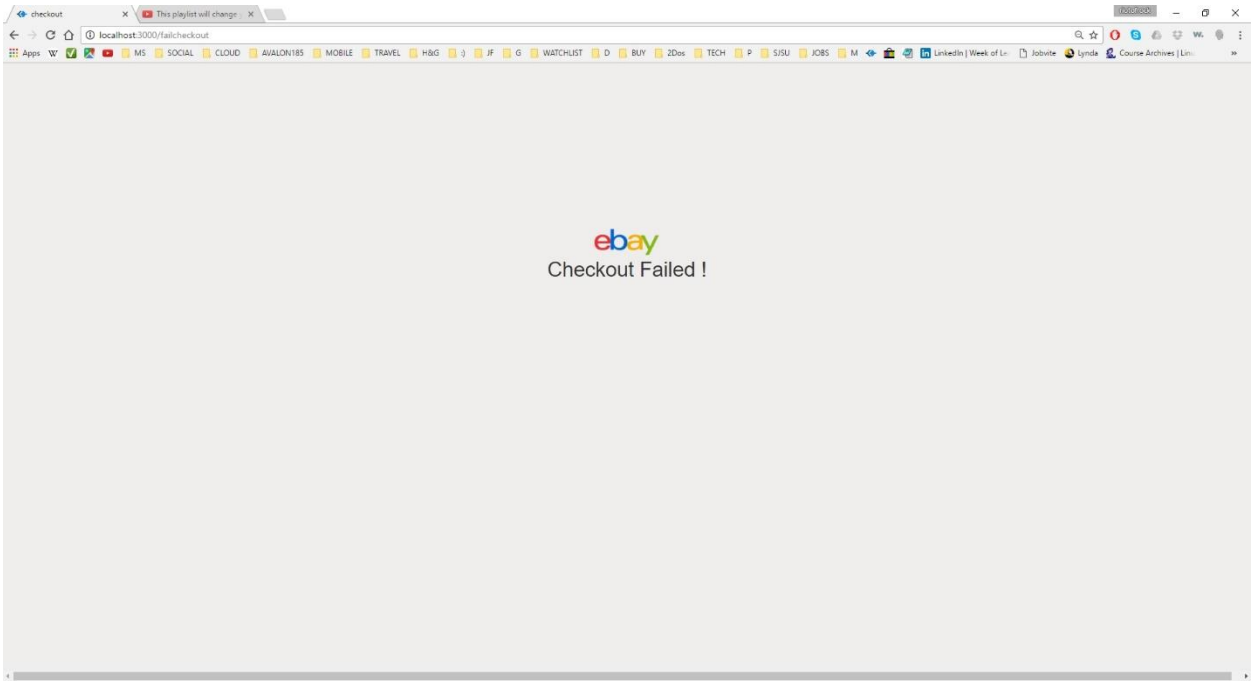


Image description: checkout failure page; can return to main page by clicking on ebay image

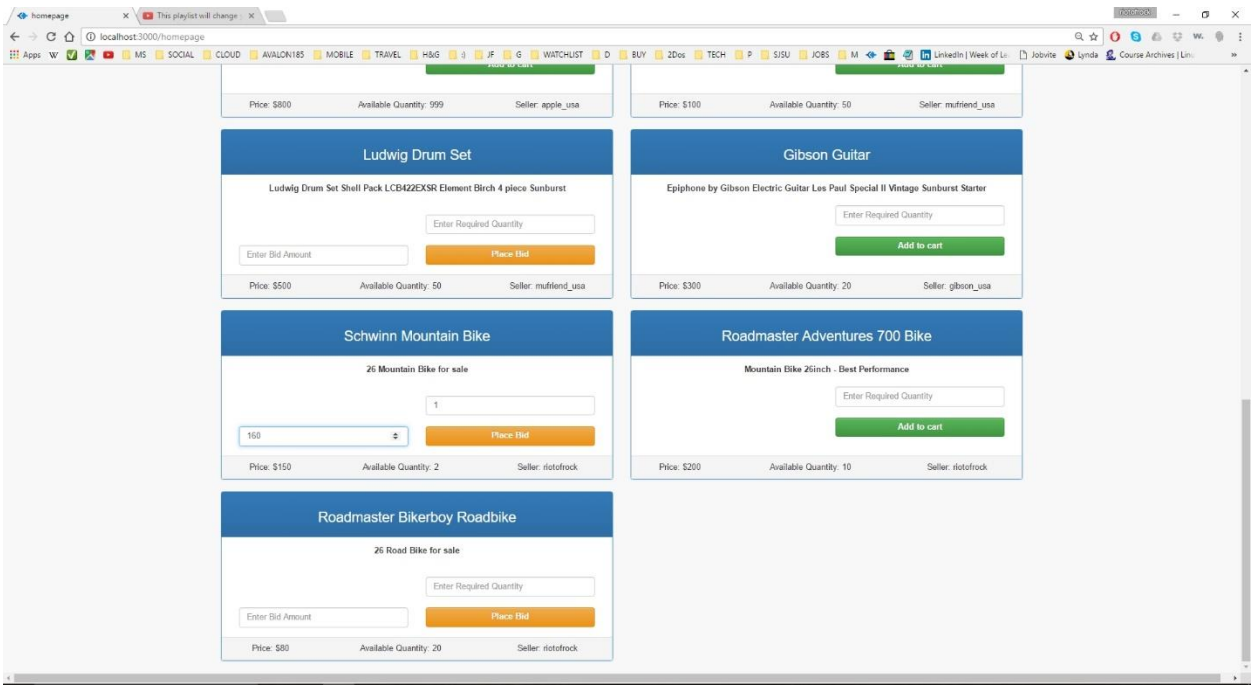


Image description: home page – bidding an item; user can bid an item by entering bid amount and the required quantity. Bidding will be allowed if the required quantity is more than the available quantity and the bid amount is more than the current product price displayed.

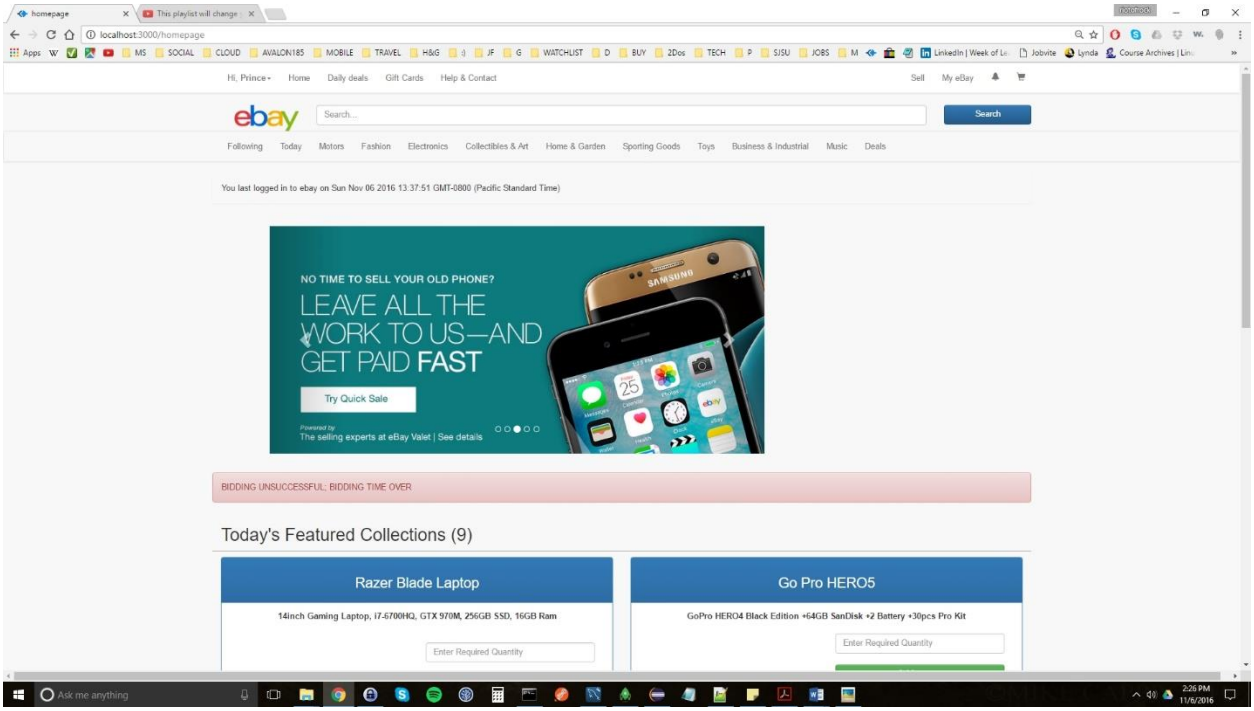


Image description: home page – bidding unsuccessful prompt will be shown when user try to bid a product whose bidding period is over

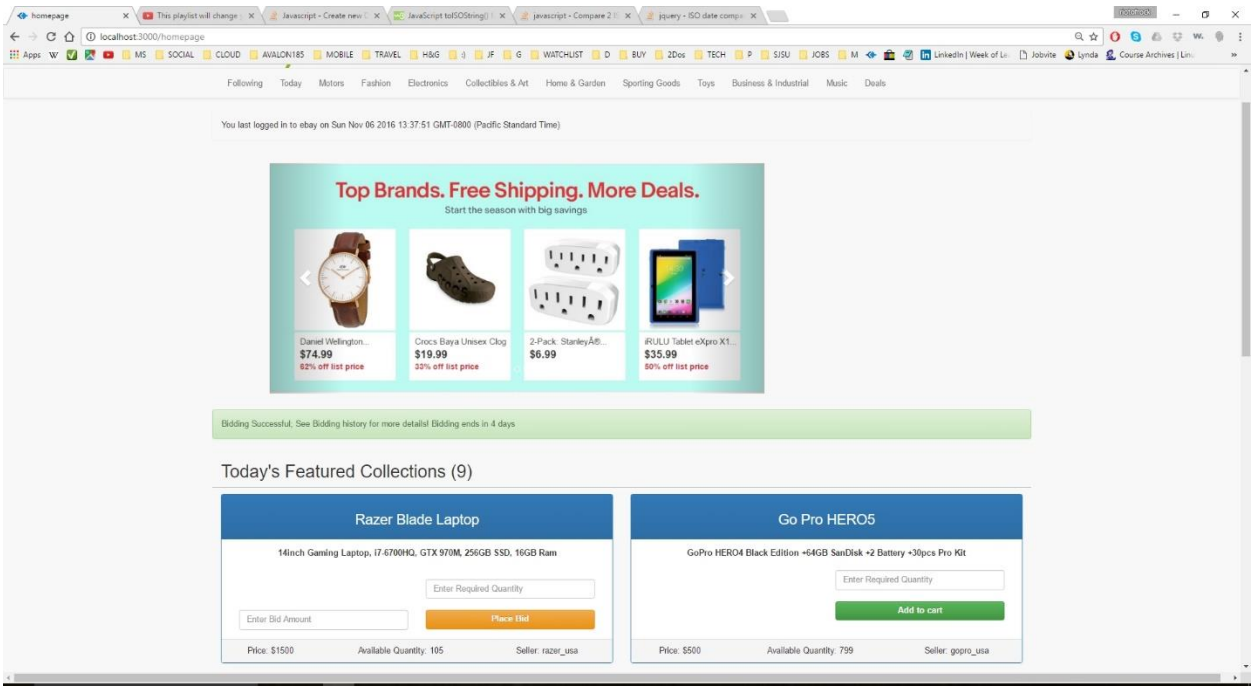


Image description: home page – bidding successful prompt will be shown to the user with the bidding timer information (time left till the bidding ends)

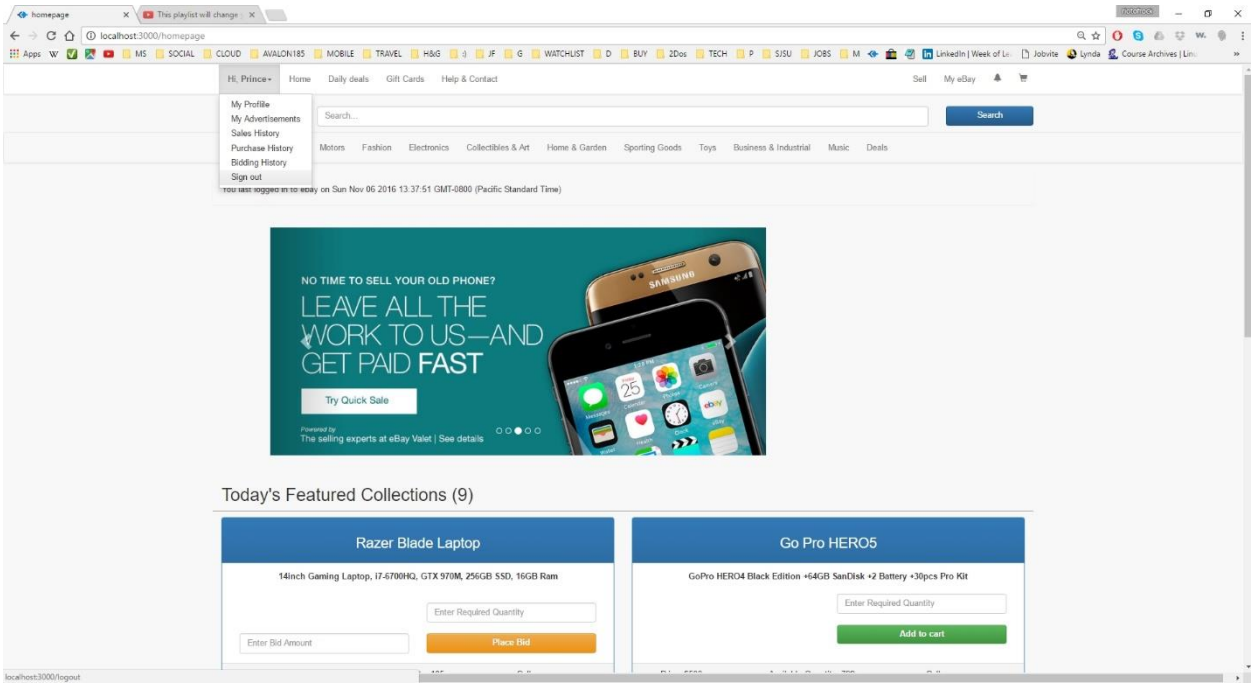


Image description: home page – user logging out

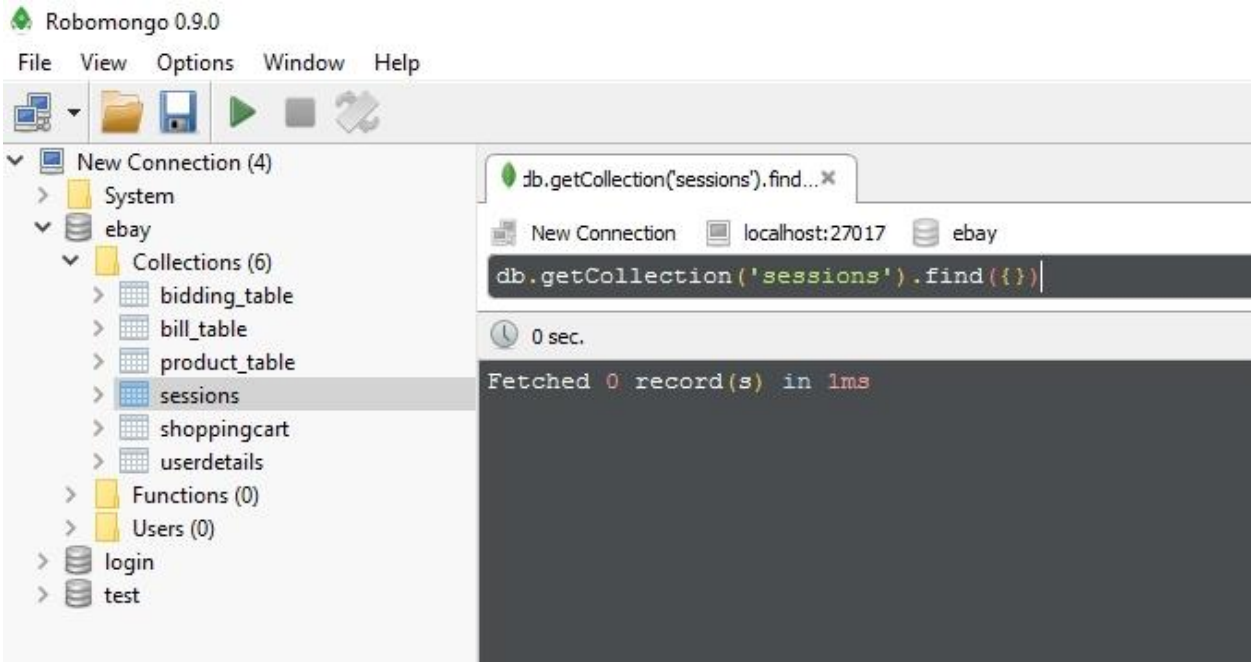


Image description: respective session information will be cleared/destroyed when user log out of the system (1)

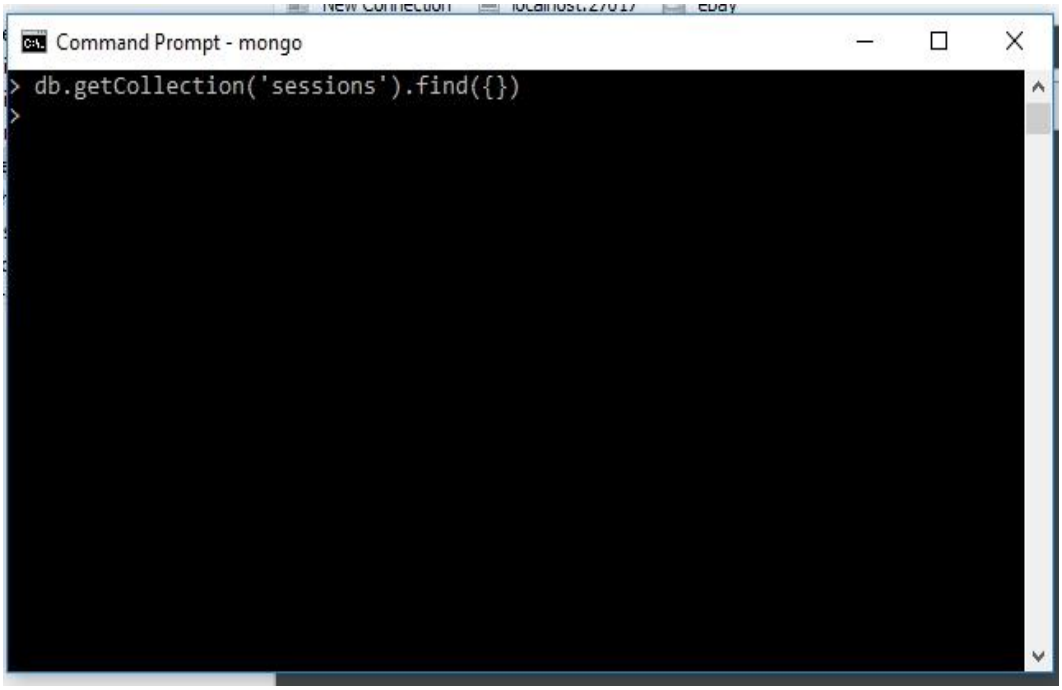


Image description: respective session information will be cleared/destroyed when user log out of the system (2)

```
//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
addaddresstodb_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			
gettheads_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

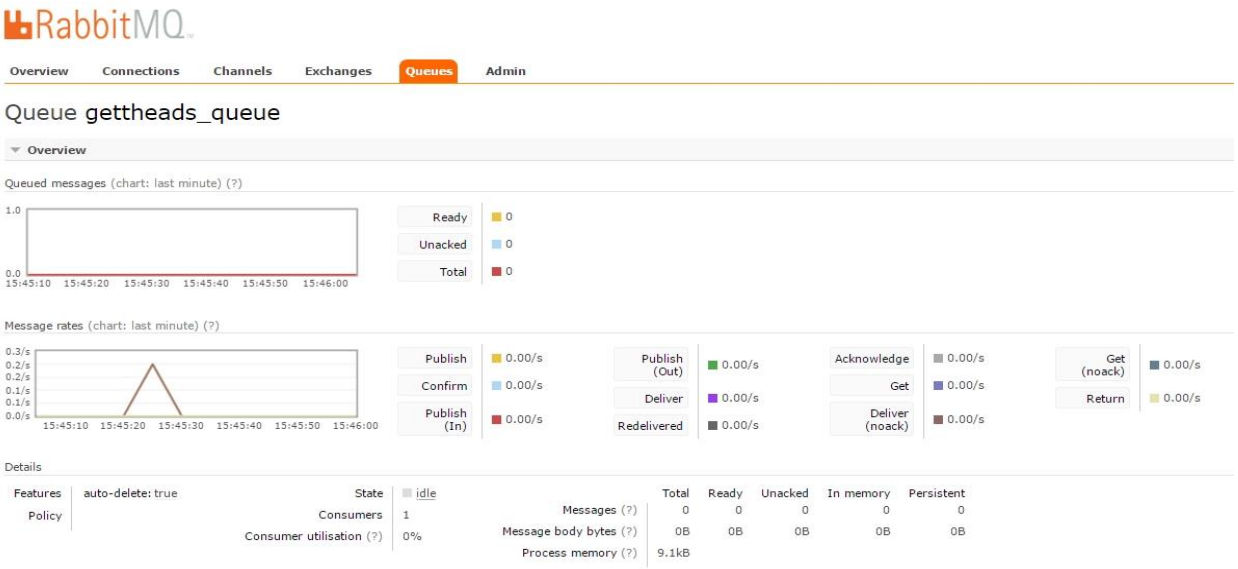


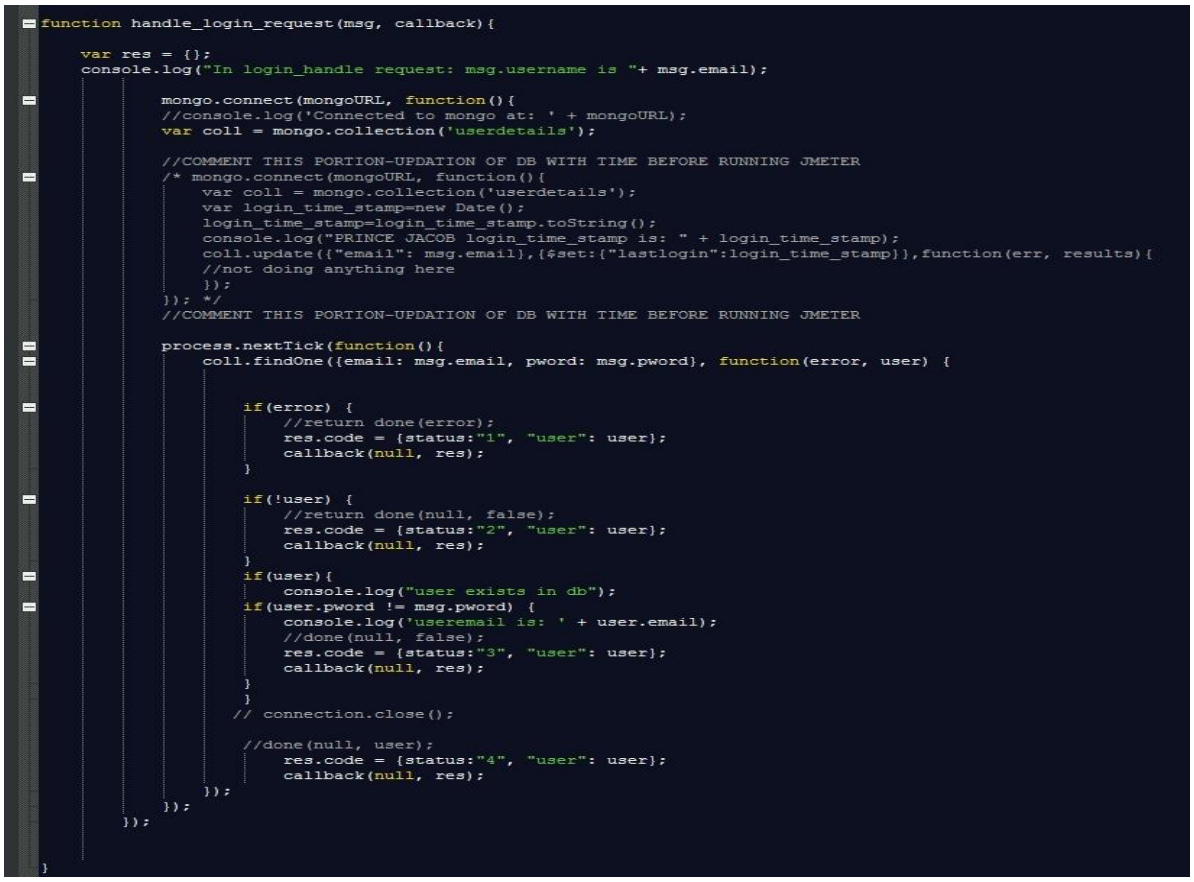
Image description: sample operation – reading advertisements from the database by a user using the gettheads_queue

```
//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 initilized");
        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.uname;
    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 succesfully : ' +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(error);

            }
            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
                });
            }); */
            //COMMENT THIS PORTION-UPDATION OF DB WITH TIME BEFORE RUNNING JMETER

            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

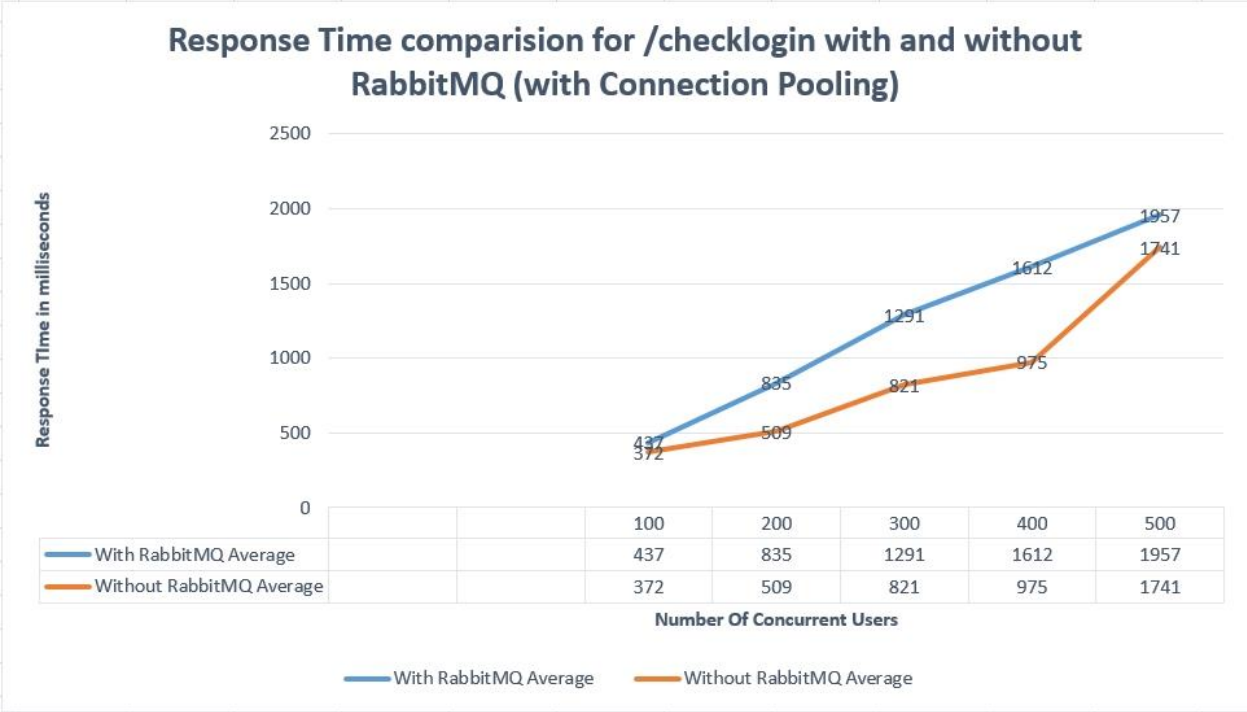


Image description: Comparison - number of concurrent users versus average response time in milliseconds for sign- in operation with and without using RabbitMQ (with connection pool)

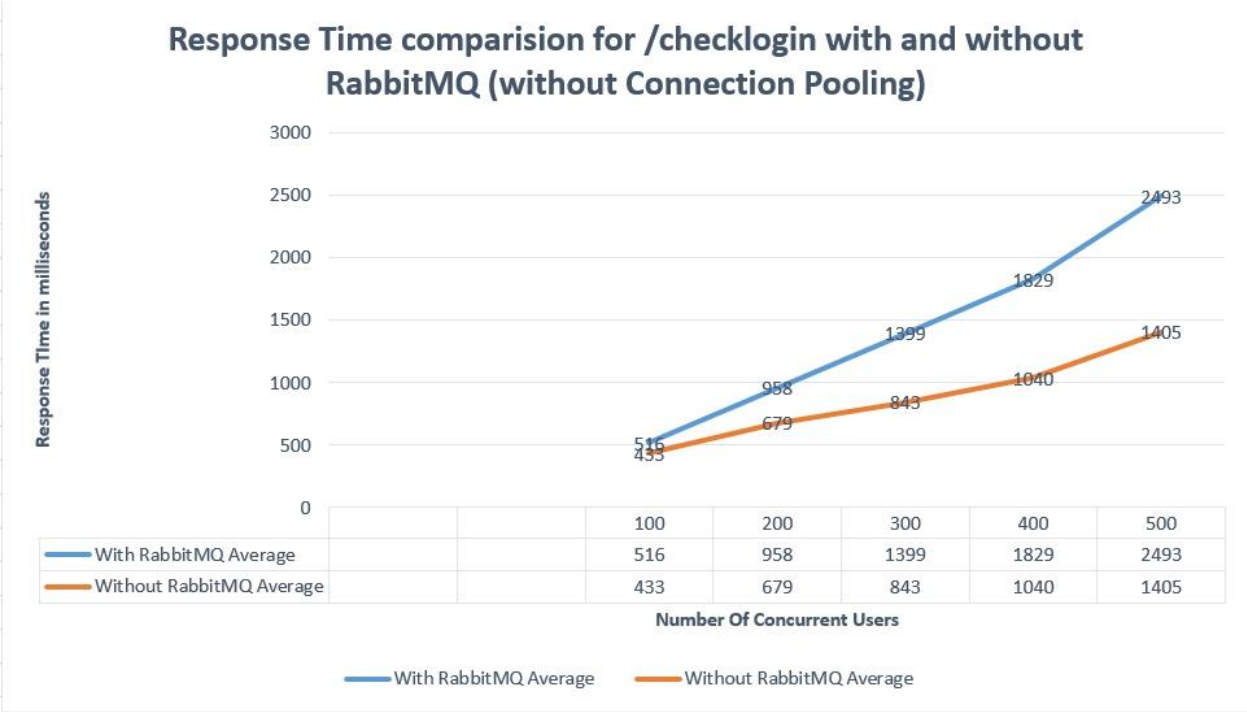


Image description: Comparison - number of concurrent users versus average response time in milliseconds for sign- in operation with and without using RabbitMQ (without connection pool)

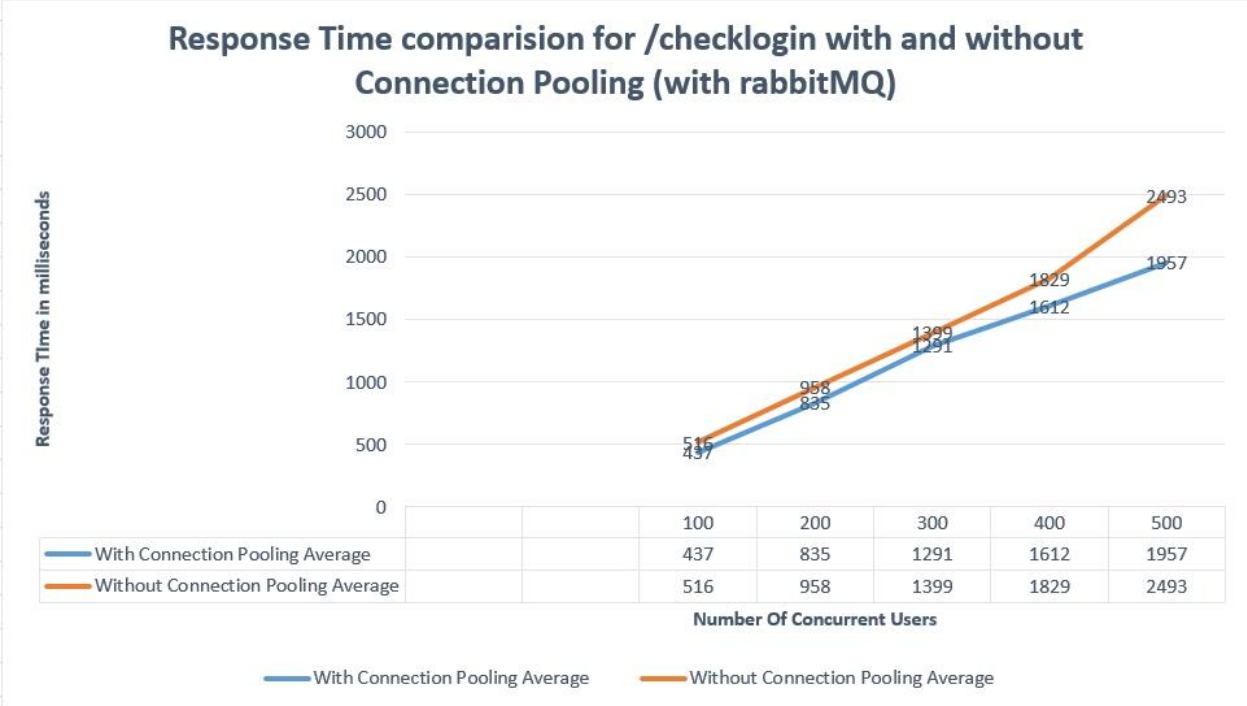


Image description: Comparison - number of concurrent users versus average response time in milliseconds for sign- in operation with and without using connection pool (with RabbitMQ)

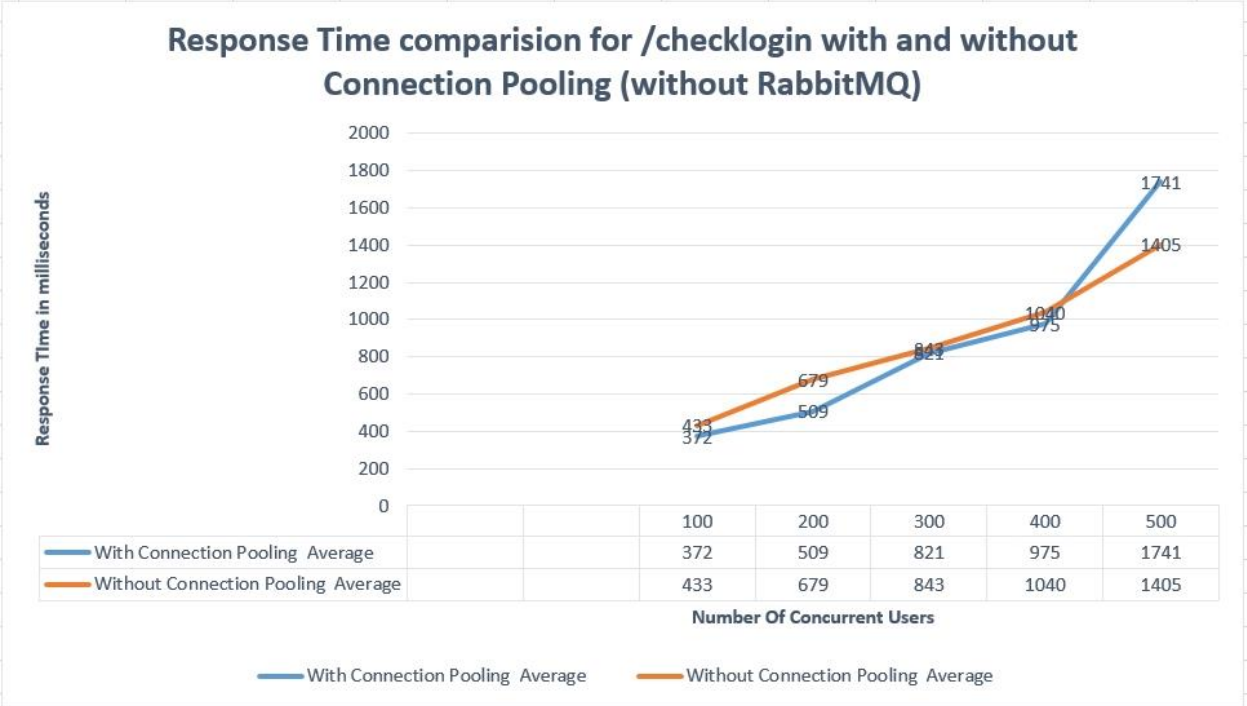


Image description: Comparison - number of concurrent users versus average response time in milliseconds for sign- in operation with and without using connection pool (without RabbitMQ)

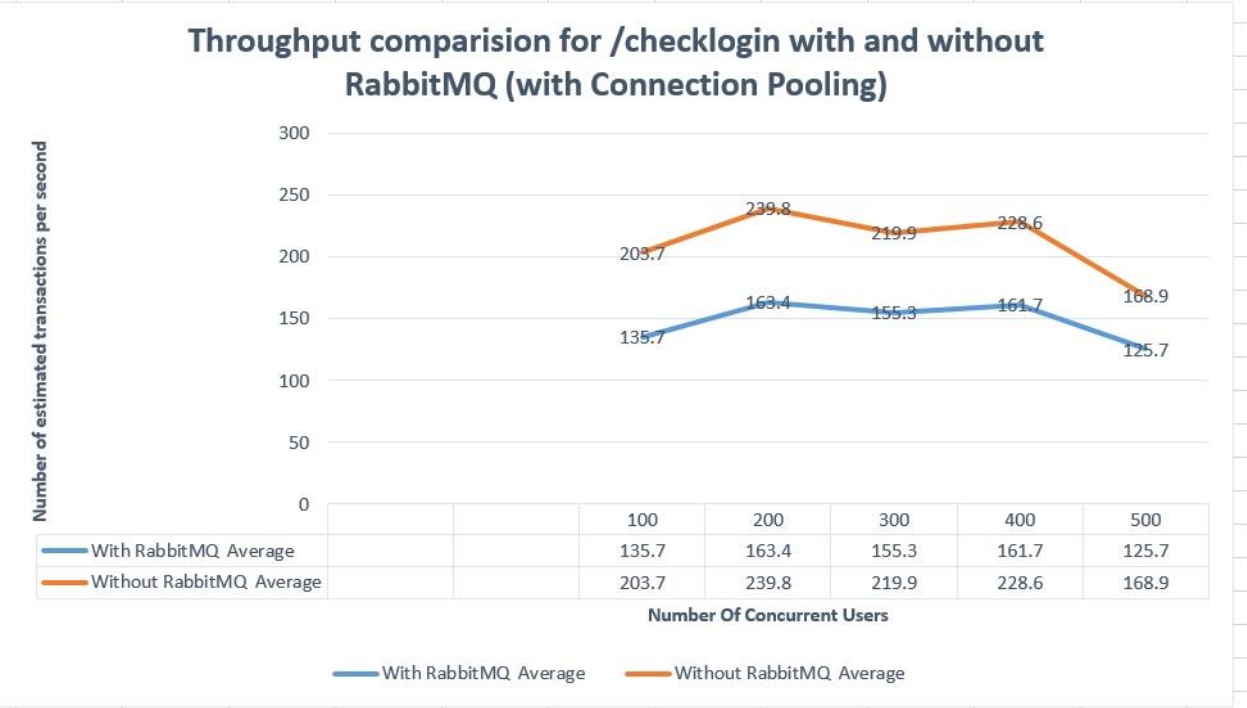


Image description: Throughput Comparison - number of concurrent users versus number of estimated transactions per second for sign- in operation with and without using RabbitMQ (with connection pool)

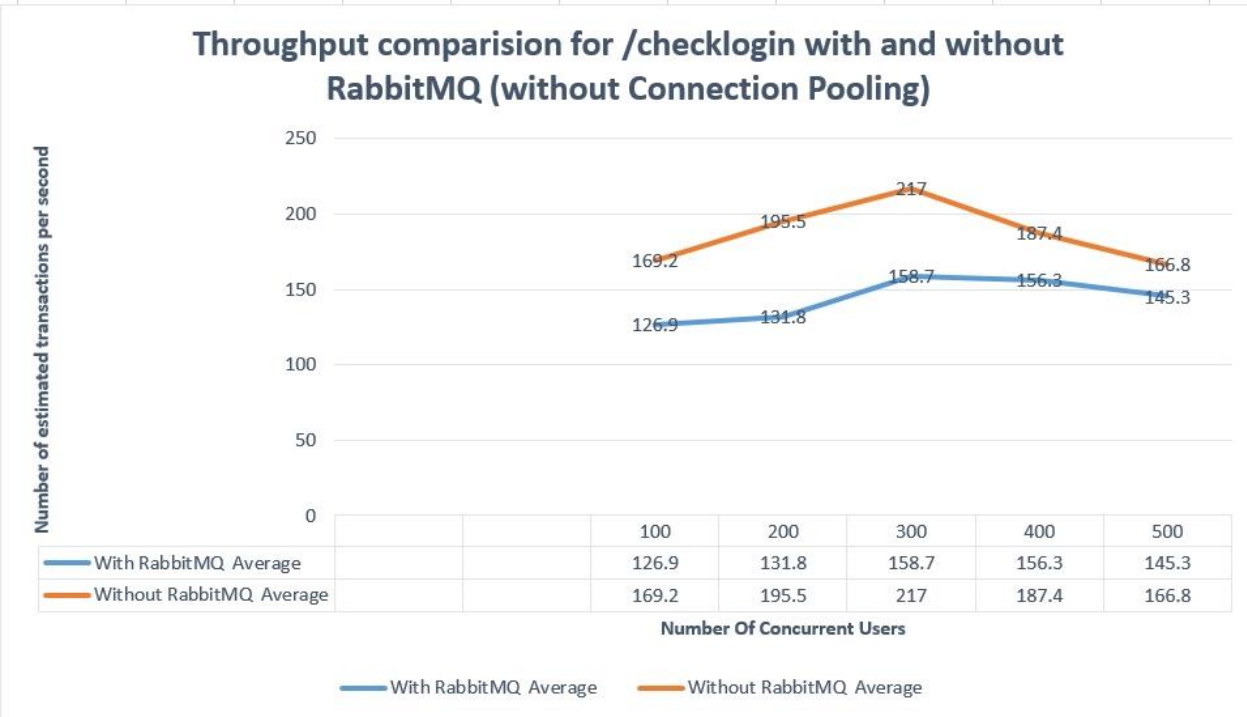


Image description: Throughput Comparison - number of concurrent users versus number of estimated transactions per second for sign- in operation with and without using RabbitMQ (without connection pool)

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

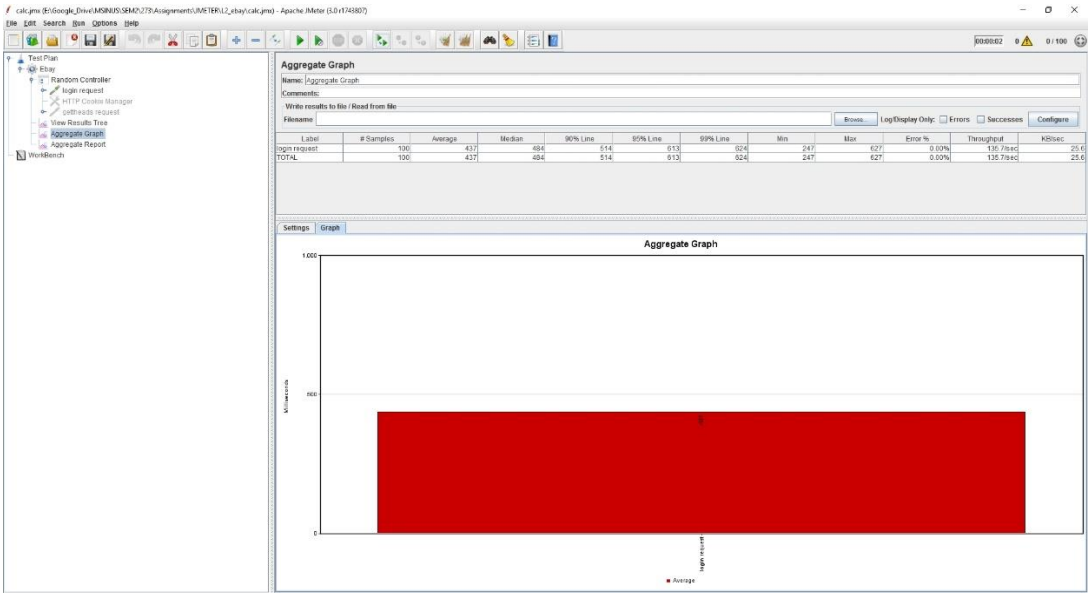


Image description: Number of concurrent users versus average response time in milliseconds – for 100 users

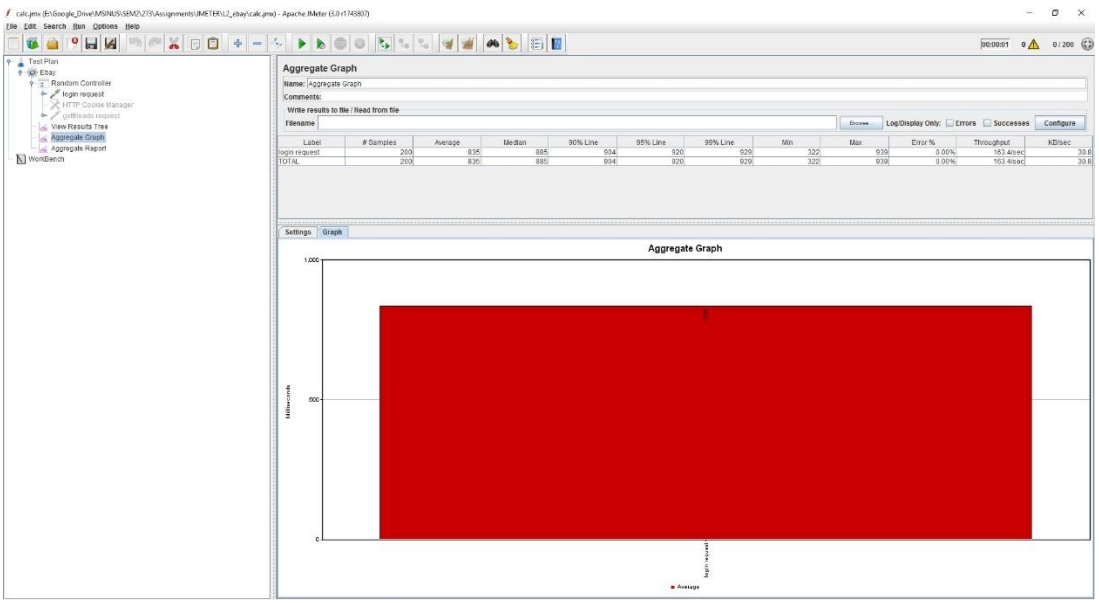


Image description: Number of concurrent users versus average response time in milliseconds – for 200 users

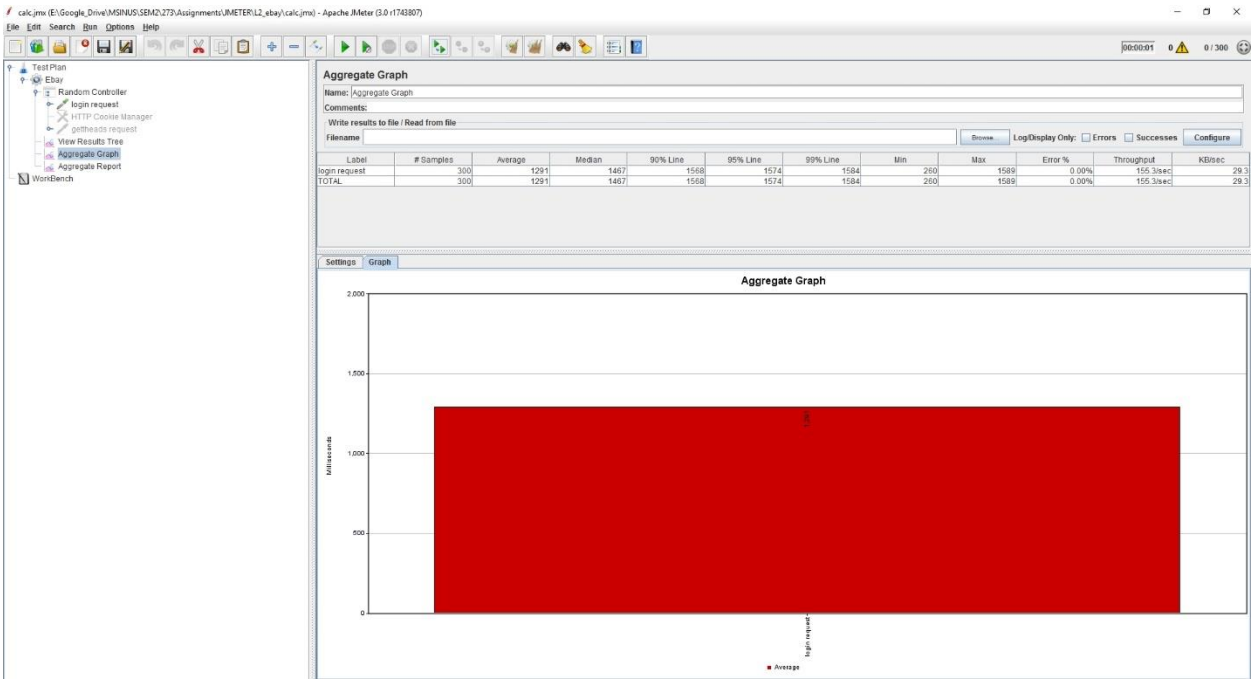


Image description: Number of concurrent users versus average response time in milliseconds – for 300 users

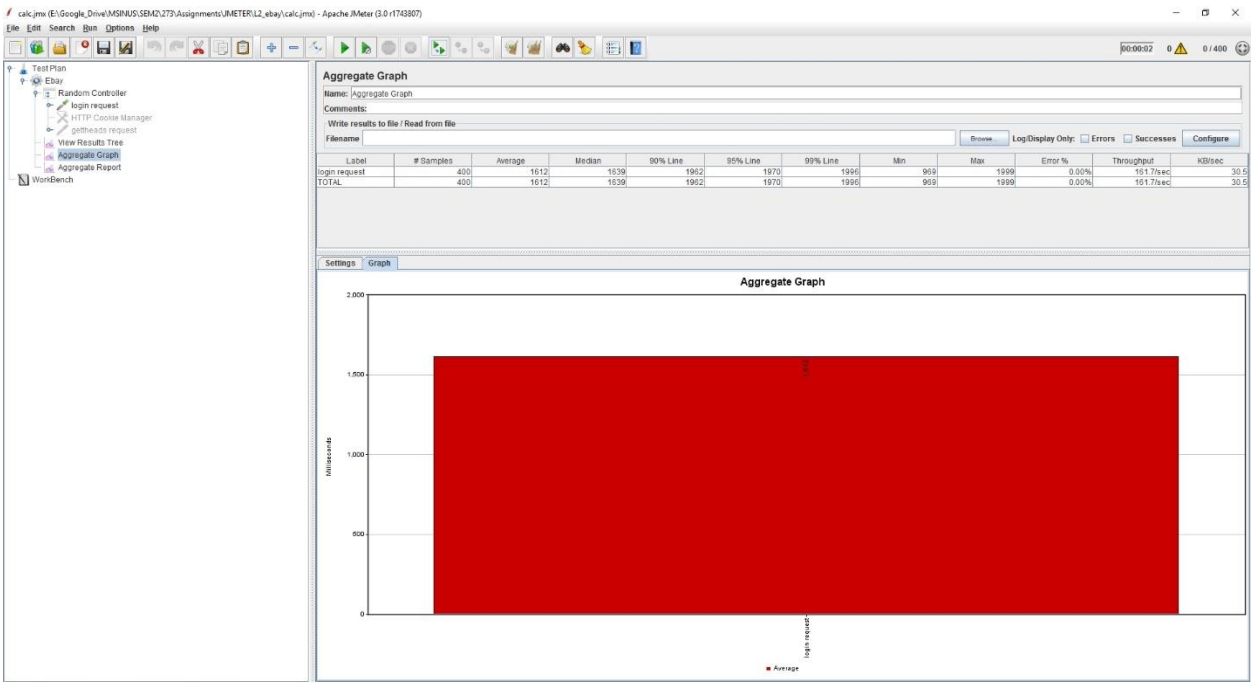


Image description: Number of concurrent users versus average response time in milliseconds – for 400 users

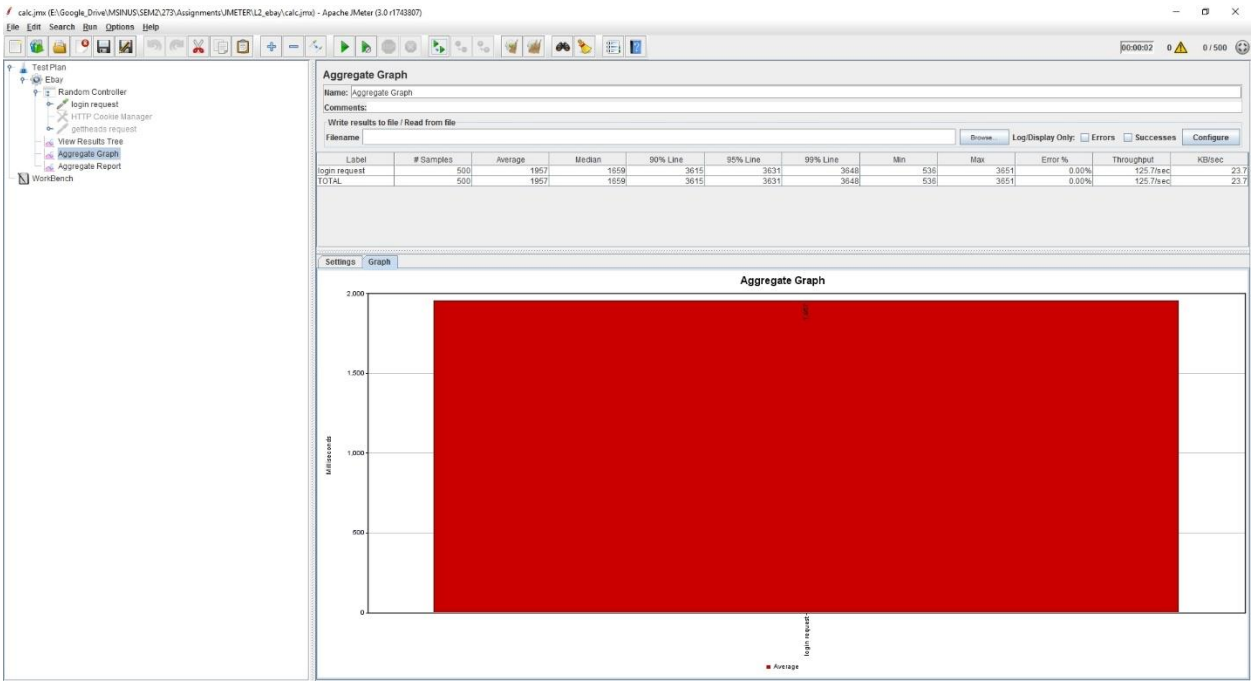


Image description: Number of concurrent users versus average response time in milliseconds – for 500 users

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

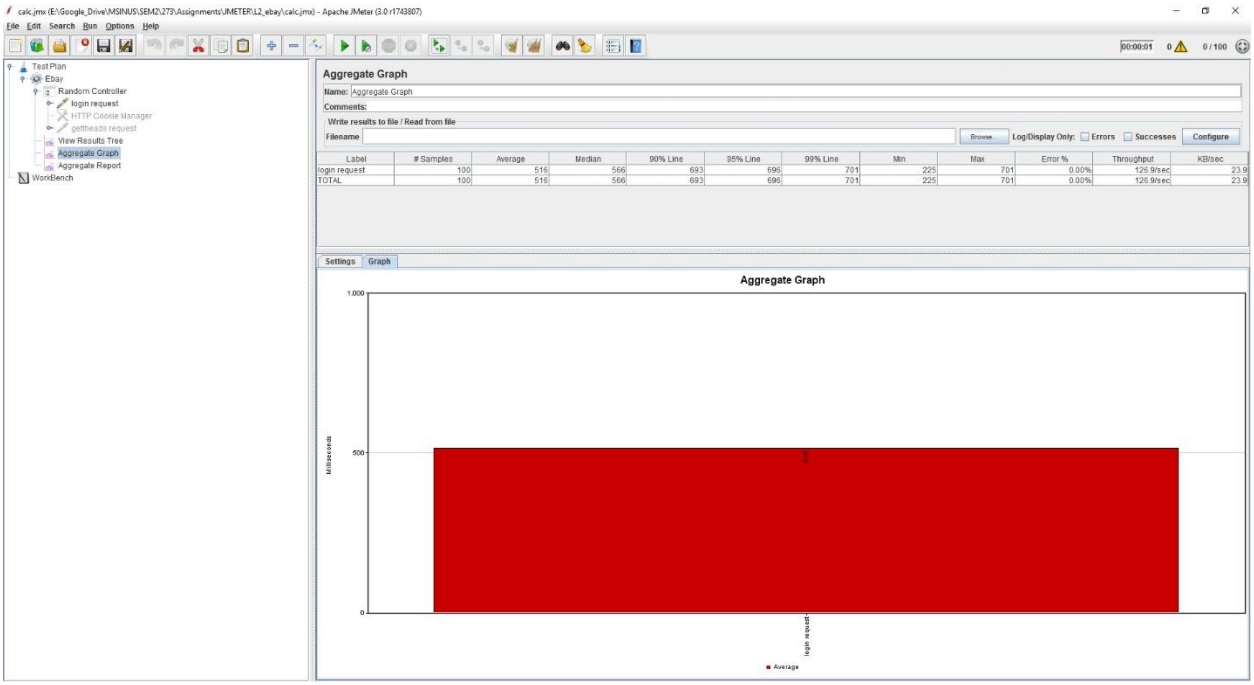


Image description: Number of concurrent users versus average response time in milliseconds – for 100 users

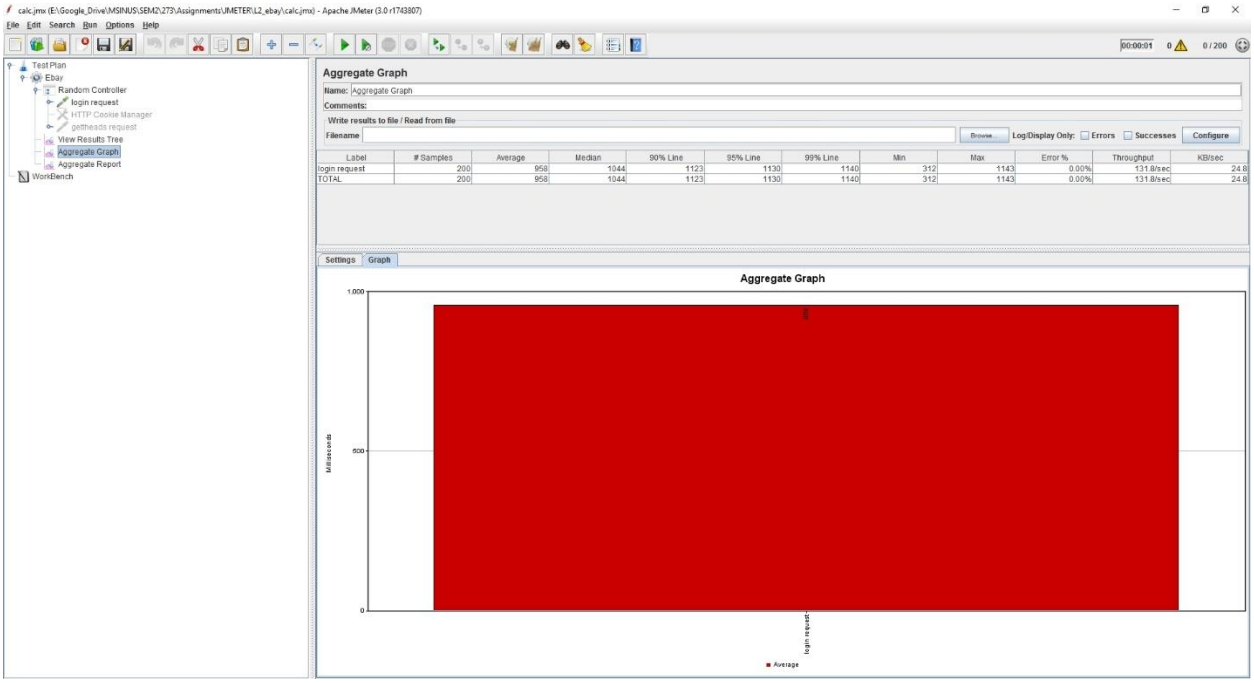


Image description: Number of concurrent users versus average response time in milliseconds – for 200 users

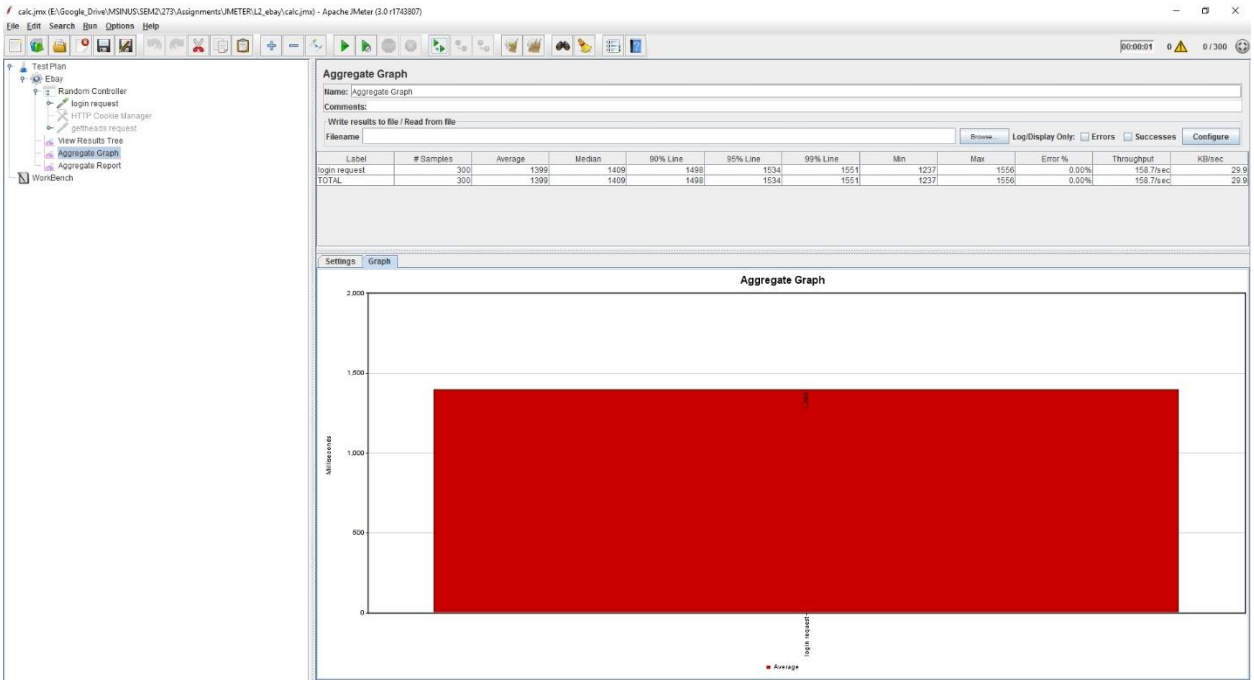


Image description: Number of concurrent users versus average response time in milliseconds – for 300 users

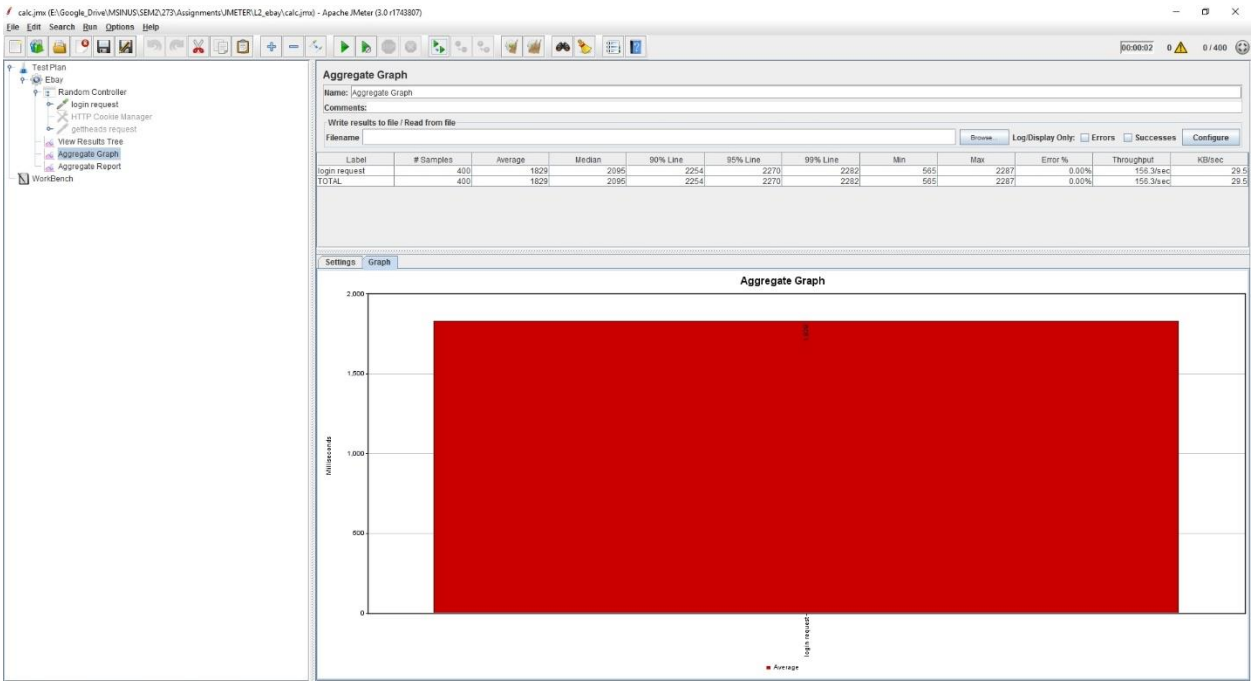


Image description: Number of concurrent users versus average response time in milliseconds – for 400 users

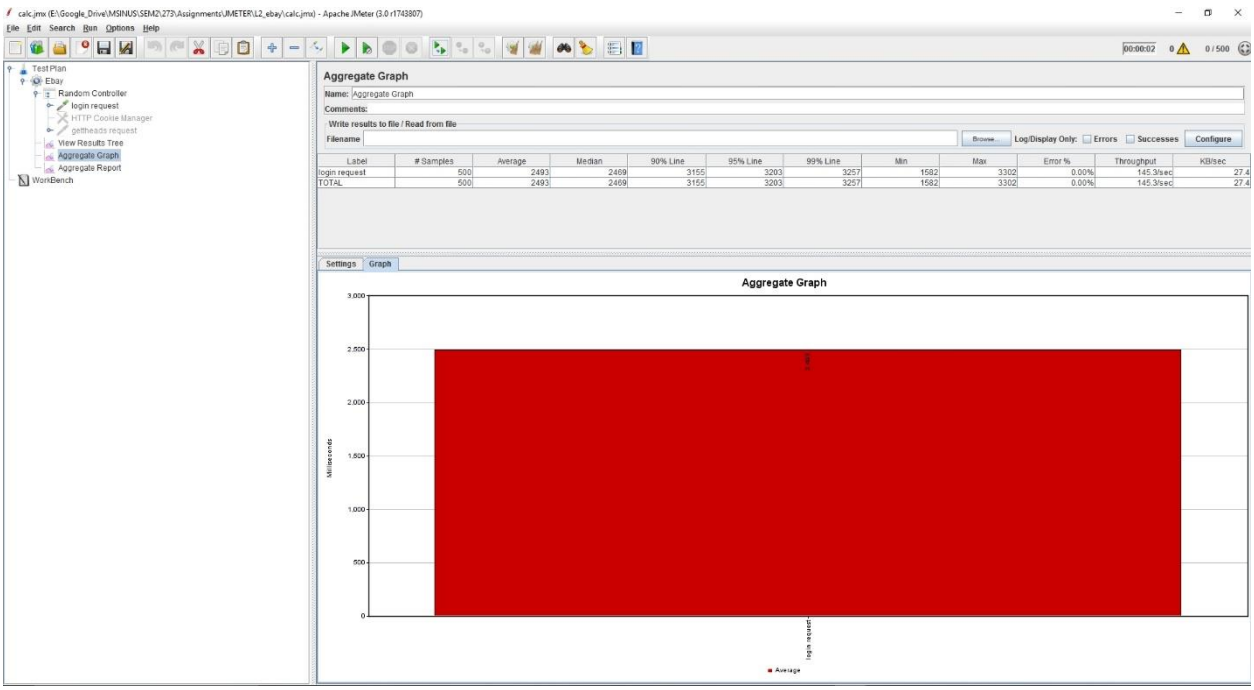


Image description: Number of concurrent users versus average response time in milliseconds – for 500 users

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

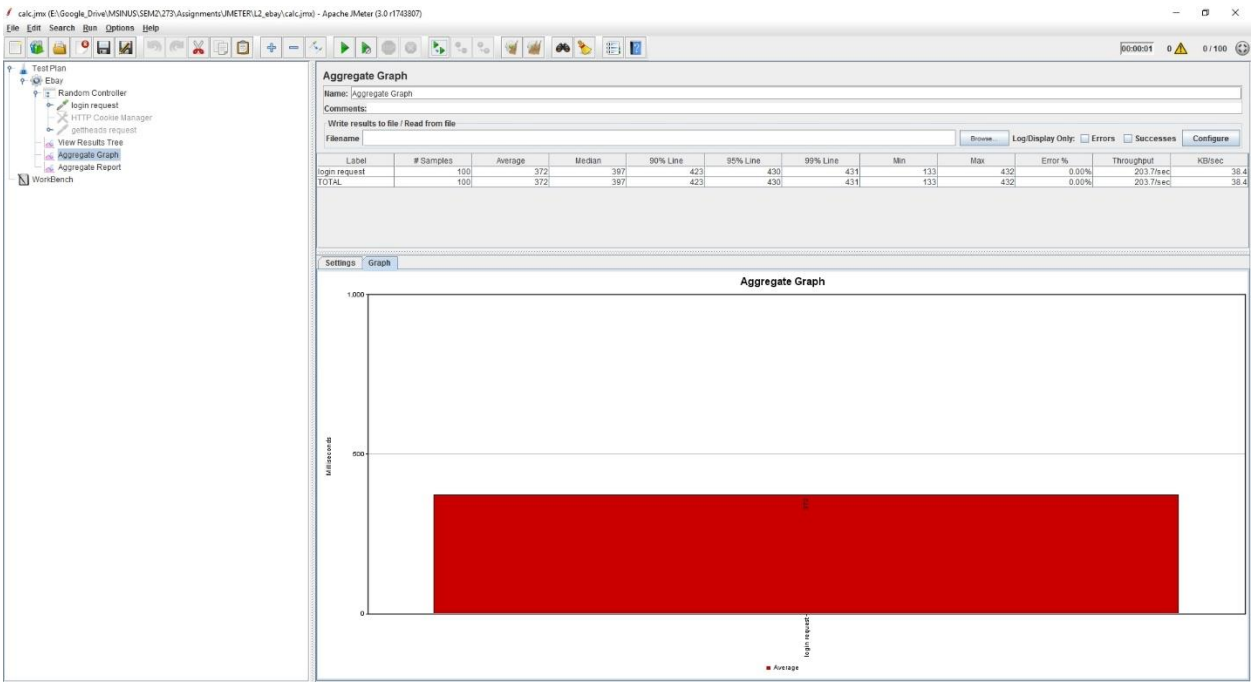


Image description: Number of concurrent users versus average response time in milliseconds – for 100 users

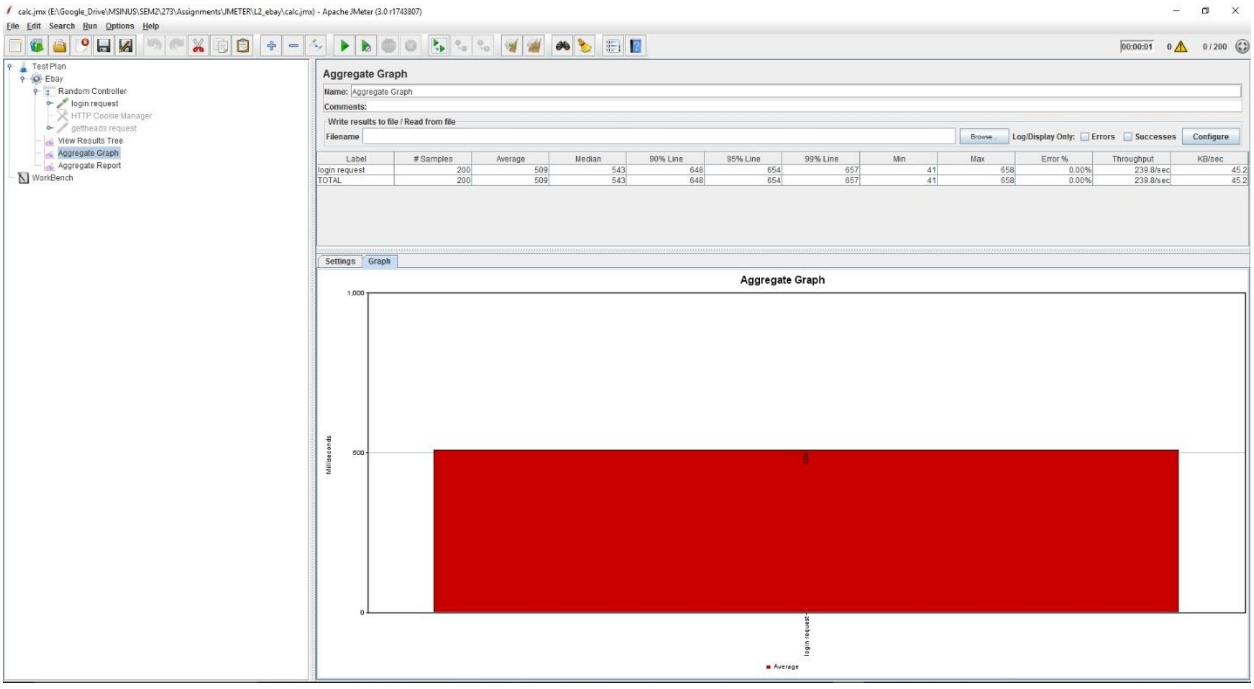


Image description: Number of concurrent users versus average response time in milliseconds – for 200 users

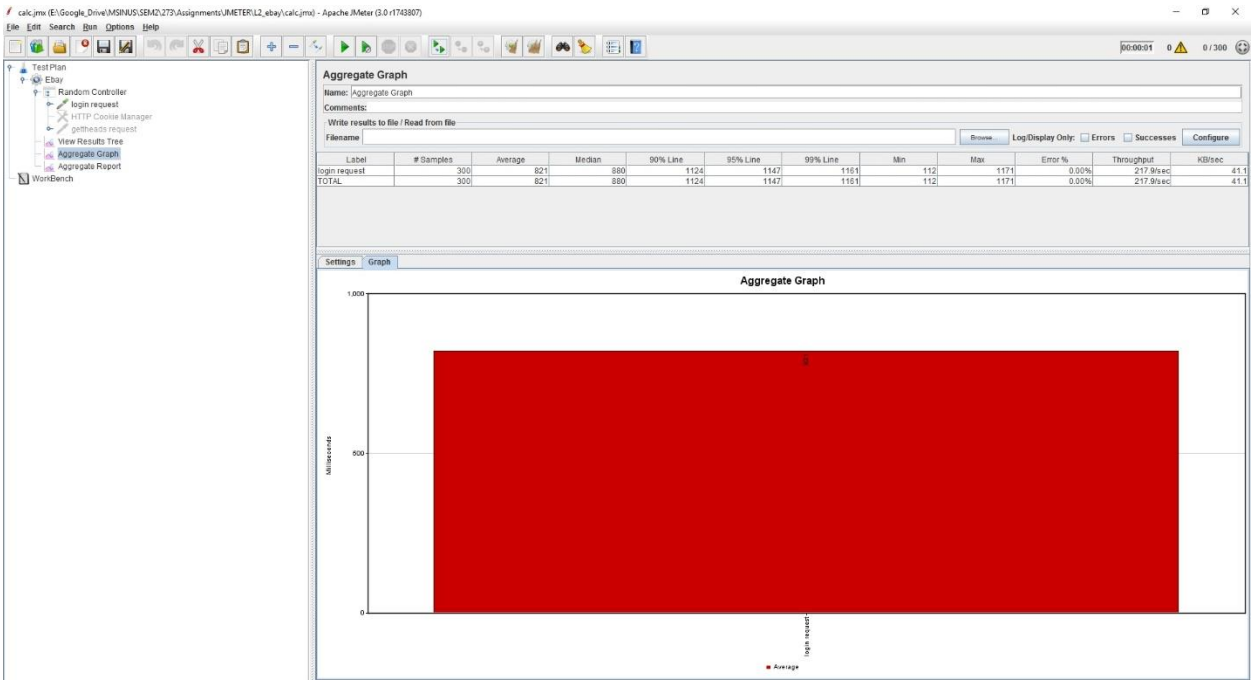


Image description: Number of concurrent users versus average response time in milliseconds – for 300 users

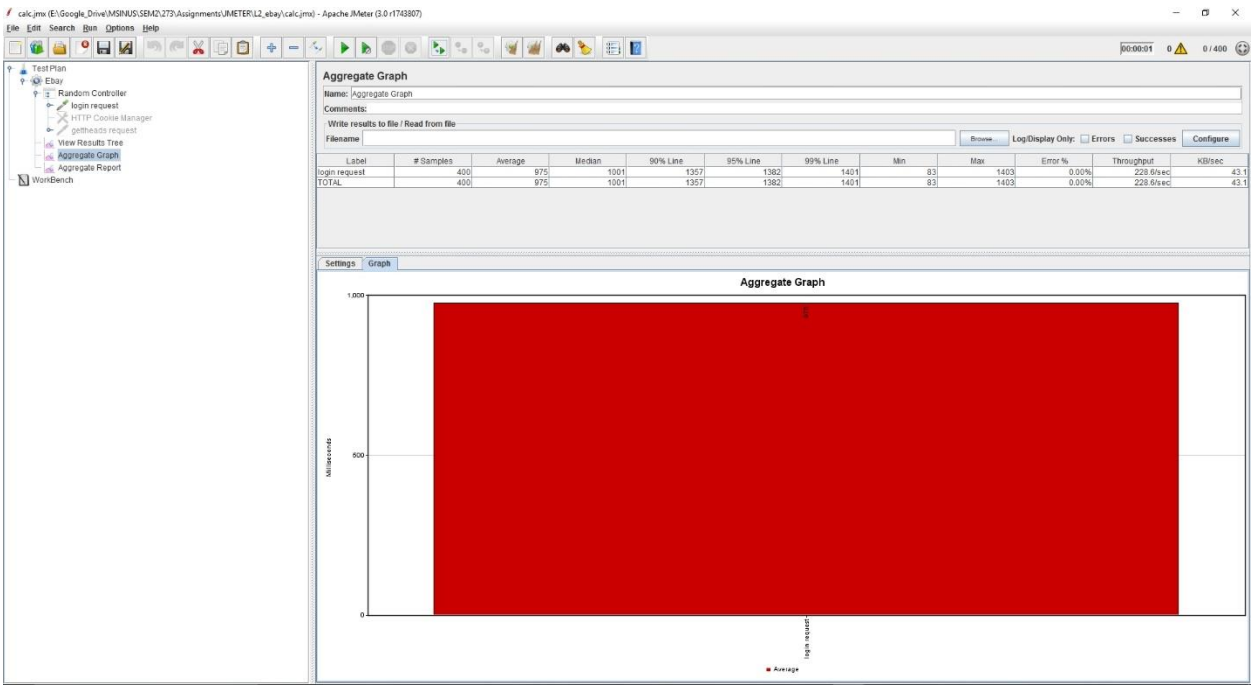


Image description: Number of concurrent users versus average response time in milliseconds – for 400 users

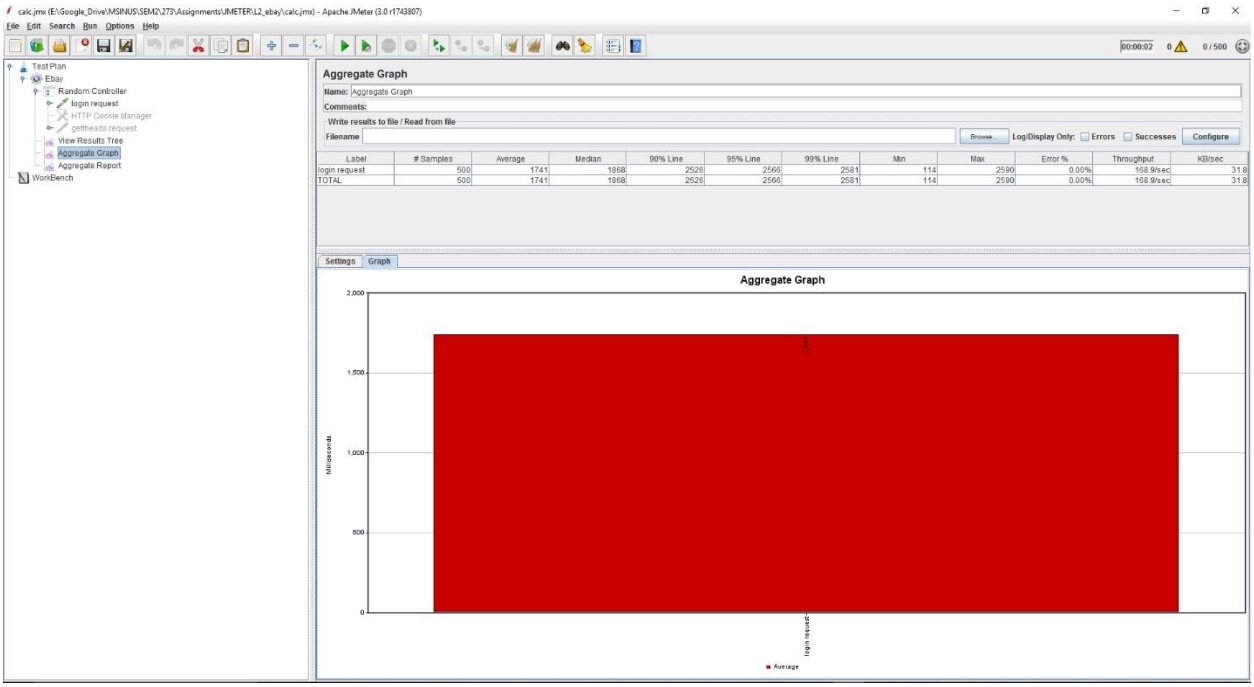


Image description: Number of concurrent users versus average response time in milliseconds – for 500 users

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

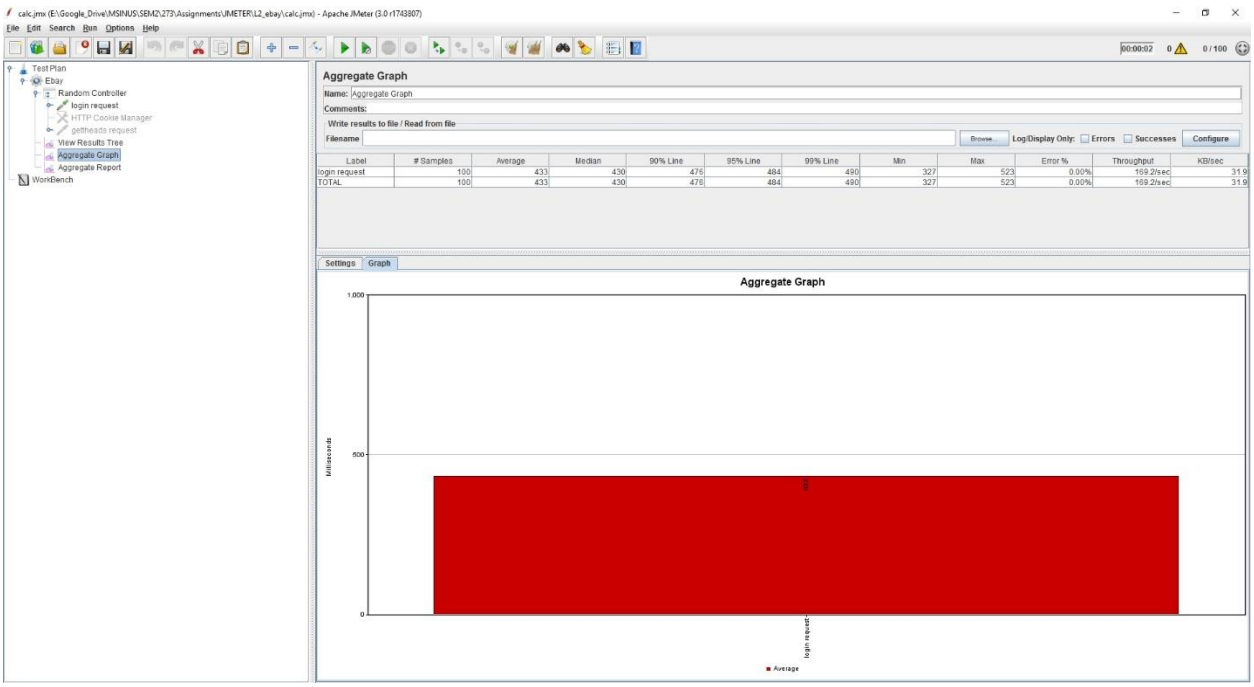


Image description: Number of concurrent users versus average response time in milliseconds – for 100 users

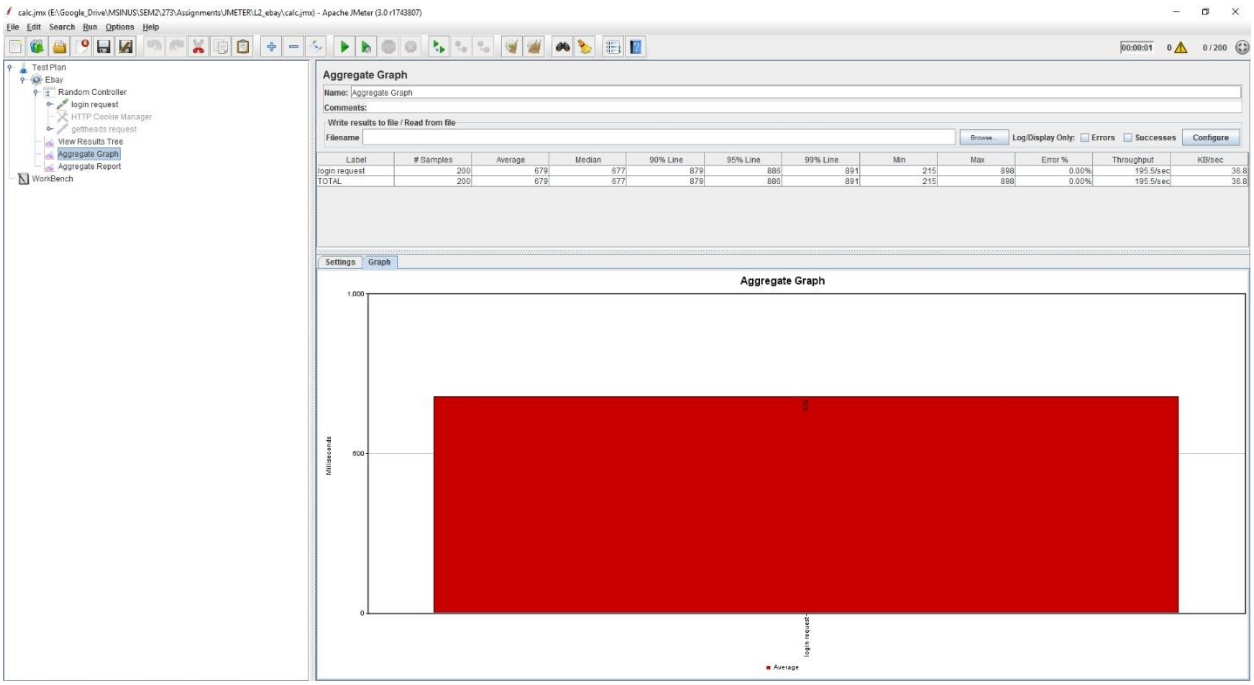


Image description: Number of concurrent users versus average response time in milliseconds – for 200 users

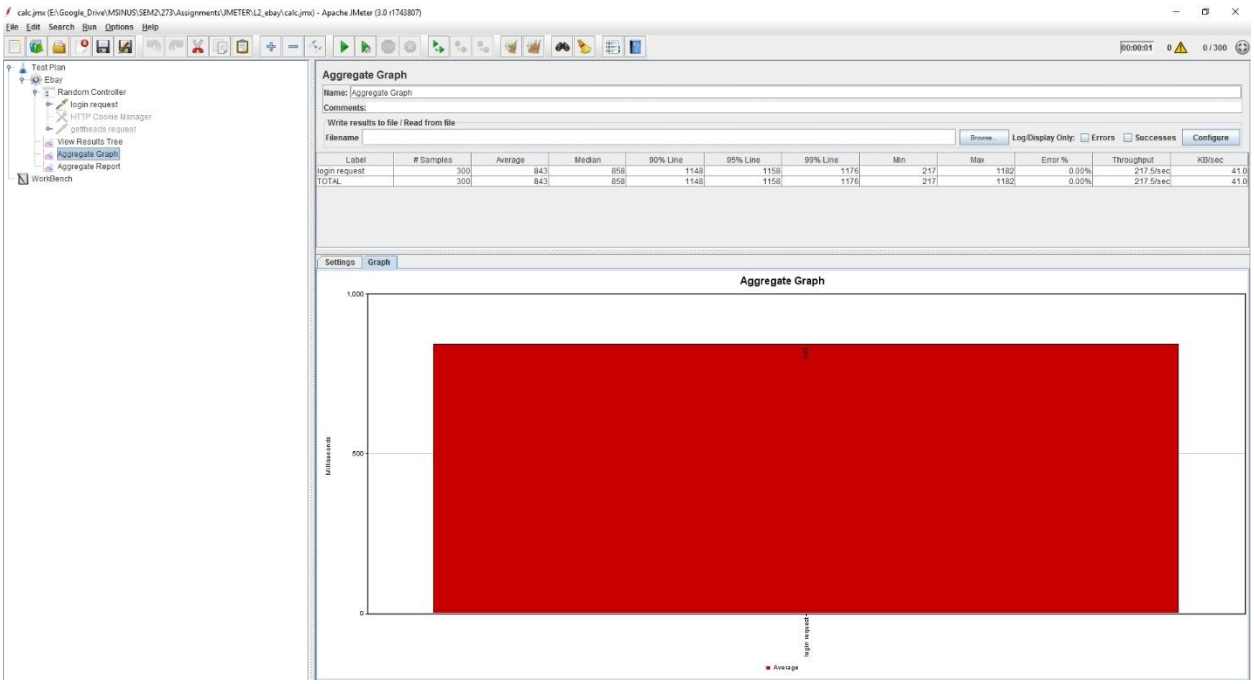


Image description: Number of concurrent users versus average response time in milliseconds – for 300 users

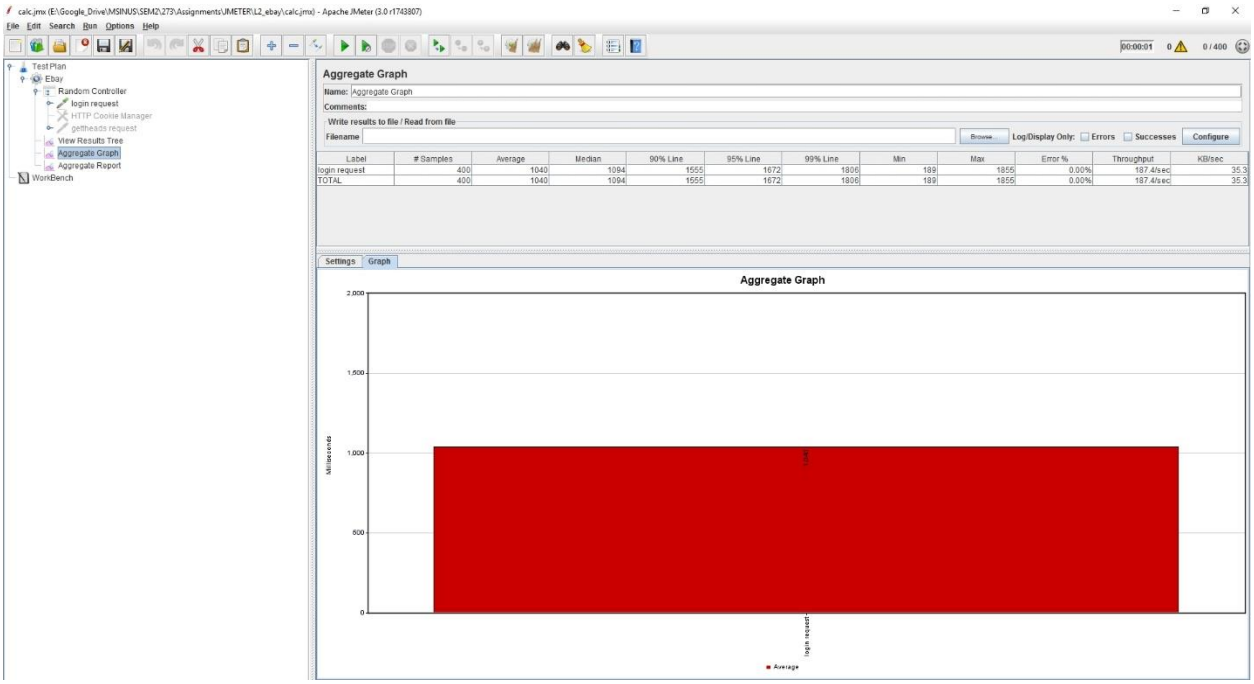


Image description: Number of concurrent users versus average response time in milliseconds – for 400 users

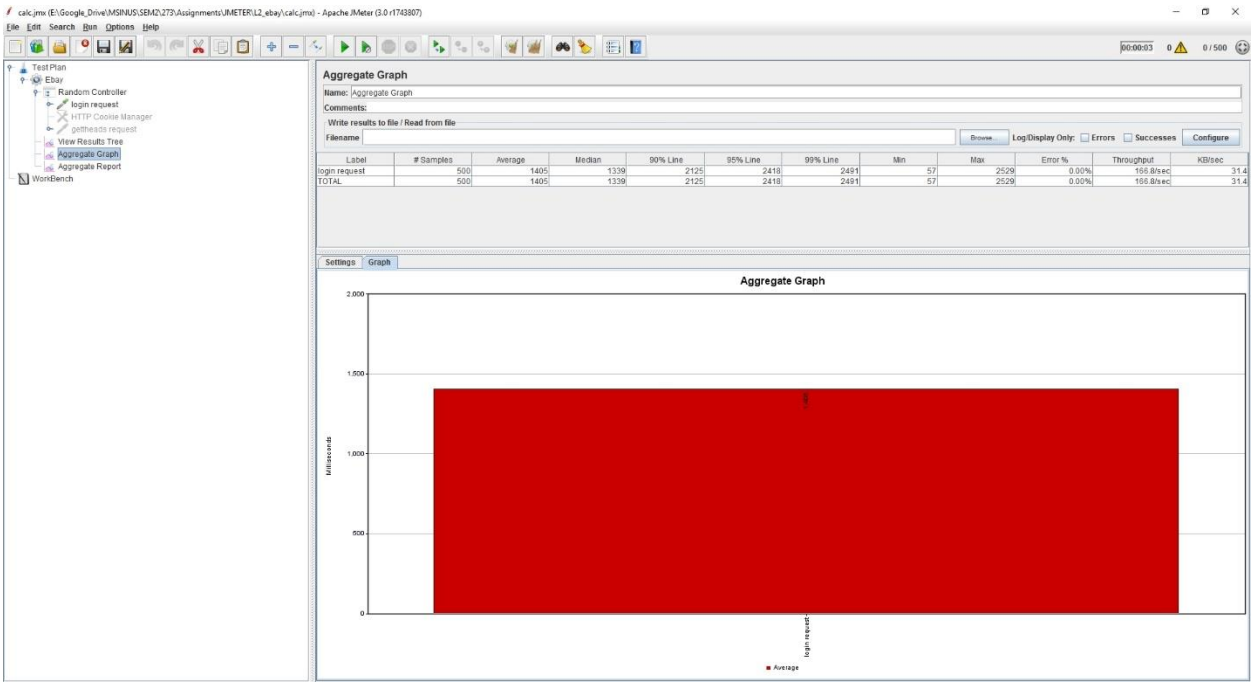
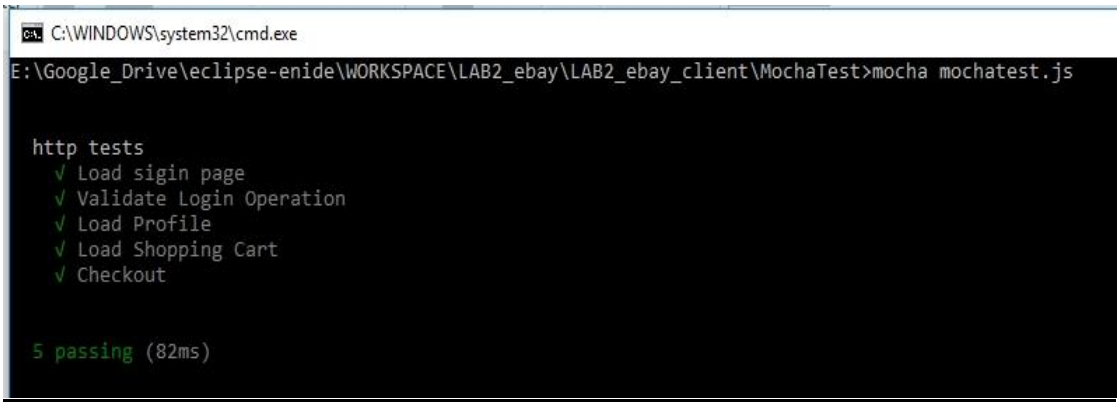


Image description: Number of concurrent users versus average response time in milliseconds – for 500 users

Testing Using mocha framework:




```
it('Load sigin 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.

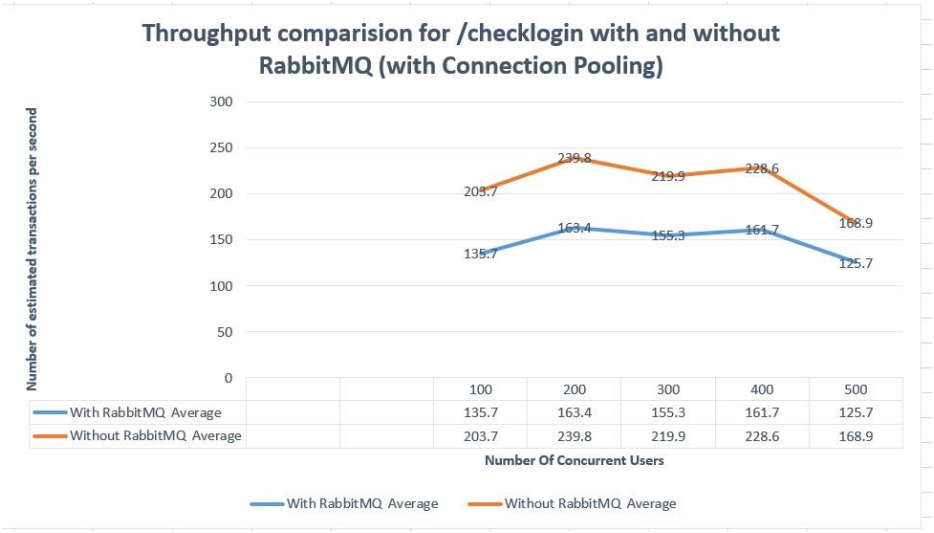


Image description: Throughput Comparison - number of concurrent users versus number of estimated transactions per second for sign- in operation with and without using RabbitMQ (with connection pool)

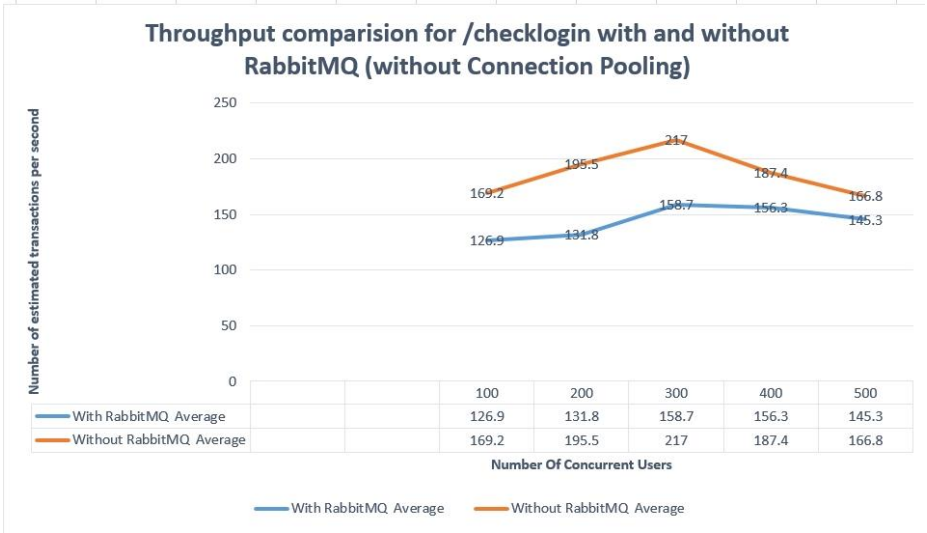


Image description: Throughput Comparison - number of concurrent users versus number of estimated transactions per second for sign- in operation with and without using RabbitMQ (without connection pool)

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.