SOFTWARE ENGINEERING

FALL 2016

SAN JOSÉ STATE UNIVERSITY

CMPE 273 – Enterprise Distributed Systems – Team Project

Airbnb Simulation with Auction – Report

Team Number 17

Team Member	SJSU ID
Maitray Shah	011432279
Parth Mukesh Upadhyay	011451219
Salmaan Pehlari	011409307
Shrey Bhatt	011489777
Vedant Shete	011412141

Contribution of Members

- **Maitray Shah:** Back End Development of Analytics Module, Mocha Testing, Front End Design of Analytics and Host Module

- **Parth Upadhyay:** Backend Development of Trips, Property Listing Module, Redis Cache Management, RabbitMQ Implementation, Integration of Back End and Front End, Google Maps API

- **Salmaan Pehlari:** Backend Development of Billing, Host, User Module, Dynamic Pricing Algorithm implementation, Mocha Testing, Integration of Back End and Front End, Redis Cache Management

- **Shrey Bhatt:** Database Design, Front End Design of Admin, User, Property, Trip and Profile Module, RabbitMQ implementation, Integration of Back End and Front End, JMeter Testing

- **Vedant Shete:** Backend Development of Admin Panel Module, Visualization of Analytics Graphs Using D3, JMeter Testing

Introduction

- **Goal**

The application is the prototype of the Airbnb. In this project we built Airbnb which enables people to list, find, then rent vacation homes.

Airbnb is a peer-to-peer accommodation market place that connects hosts (vendors of rooms/accommodations) and travelers via its website.

Users of the site must register and create a personal online profile before using the site. Every property is associated with a host whose profile includes recommendations by other users, reviews by previous guests, as well as a response rating.

Validation at every stage of the application is also implemented, and all exceptions have also been handled.

Server is also tested with several hundred concurrent users using JMeter, and the results have been compared and compiled in a graph to compare the effect of adding connection pooling in the system.

Testing of the Rest API's has also been done using Mocha

- **Purpose of System**

We learned and implemented the MEAN stack for web development and also the use of RabbitMQ using the AMQP protocol for scalability. We have used the Redis for cache management.

System Design

Technologies used

- Frontend : Angular.js
- Backend : Node.js
- UI/UX : CSS and Bootstrap
- Markup Language : Html

- Scripting Language : JavaScript
- Database : MongoDB
- Password Encryption : Bcrypt
- Authentication : Passport.js
- Messaging Protocol : AMQP
- Message broker Middleware : RabbitMQ
- SQL Caching : Redis
- Analytics : ELK stack
- Data Generation : Python , Faker

• The system is made using NodeJS, and express JS framework at the backend.

• MongoDB is used as a database for the application.

• Sessions have also been stored in MongoDB.

• RabbitMQ is used as a message passing mechanism for database access from client to server.

• Frameworks like bcrypt for password hashing, and Winston for log generation have also been used. For front-end, angularJS is used along with HTML/CSS.

• The user is prompted to login, or sign up on the front page. The user cannot access any pages without signing in. The user's password is encrypted and stored in the database. The user' log in time is also logged in the database.

• User's personal information is also stored in the database.

• User can then proceed to post and advertisement or buy from other user advertisements.

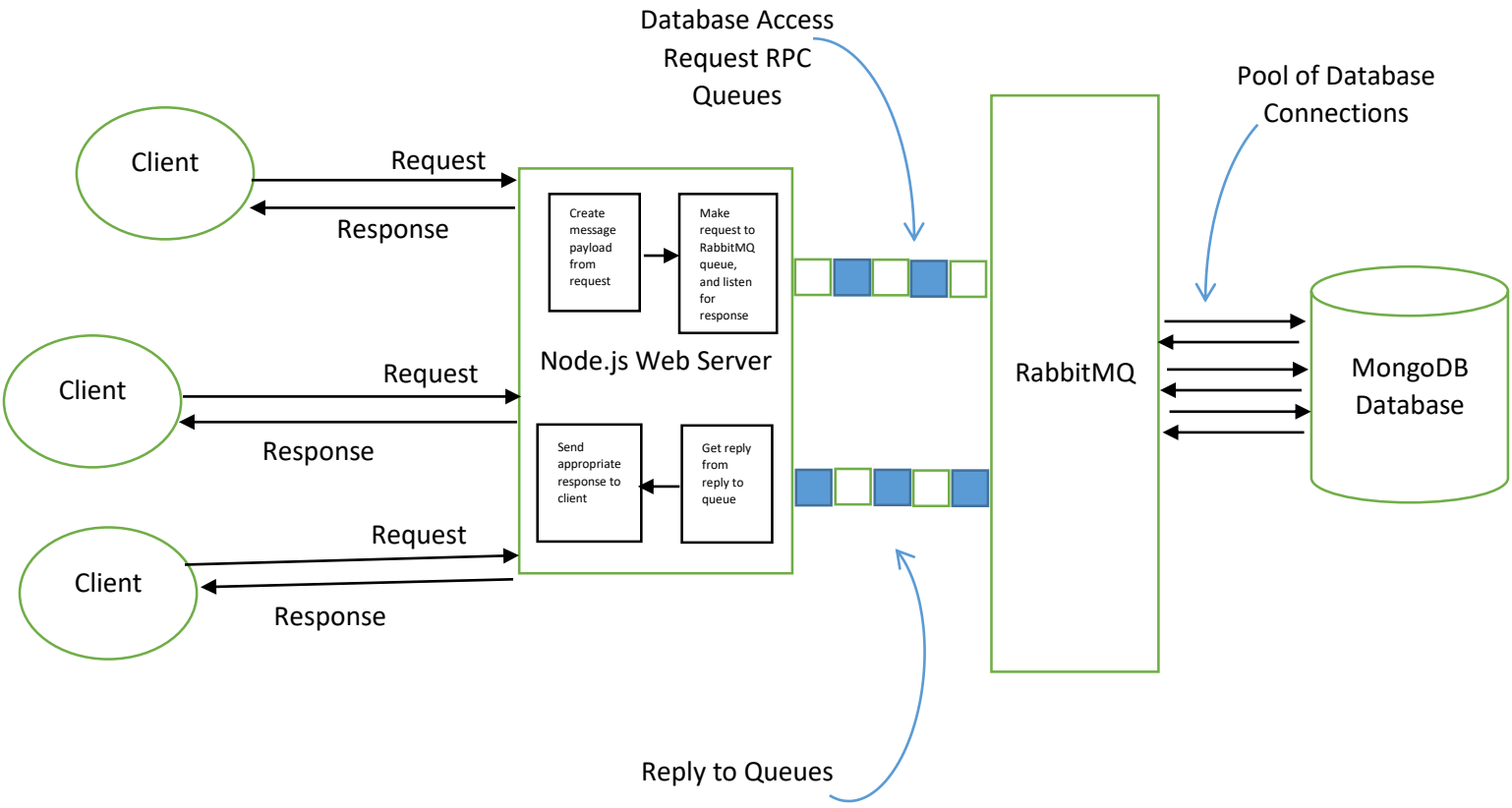
• User can also bid on other items, which takes place over a period of 4 days.

• Connection pooling has been implemented at the backend for the server to handle requests and send responses faster.

• The server tracks all activities of the user like which item seen, item added to cart, time of log in, etc.

• Server also takes of validation at every stage. Incorrect login, negative values for stock, and item quantities have been taken care of as well.

ARCHITECTURE DIAGRAM



Object Management Policy

1. Requirement gathering and Requirement Analysis:

In this phase, we performed requirement gathering and requirement analysis. As per the requirements of the project the three tiers needed in order to accomplish the Airbnb application are as following:

I) **Client tier**: The user will interact with the Airbnb application using the User Interface.

II) **Middle tier/ Message Oriented Middleware(MOM)** : This tier supports sending and receiving messages between distributed systems. The services have been implemented using RabbitMQ message broker which implements the Advanced Message Queuing Protocol(AMQP).

III) **Database tier**: This tier comprises of the database to store the objects of the system. We are using two databases for the implementation of this application.

i) MySQL : We are storing the data in MySQL if the application has high transactional data which requires ACID properties.

ii) MongoDB: In MongoDB we are storing the data which requires flexible schema management and the size of the data is comparatively large for the processing.

2.Database Design:

In the Airbnb application; we implemented different modules namely User, Host, Admin, Analysis, Listing, Billing and Trips. All of the modules has an associated schema which is responsible for representing how particular data should be stored in database.

3.User Interface Design and Backend Development:

On the basis of Airbnb application requirements, we have developed the User Interface and implemented different functionalities. The implementation of the maps has been done using google maps API. We have used RabbitMQ to increase the reliability when the number of requests are more. We also have used Redis for SQL cache management.

4.Testing:

We have frequently tested all the APIs of the Airbnb application using Jmeter and Mocha and recorded results for the same. We have made changes accordingly if the performance was not up to the mark. We have designed Mocha test cases for each APIs.

How we handled the "Heavy Weight" resources.

The heavy weight resources in the project are images and videos of properties, photos of users and photos uploaded by user during property and trip review. We have all these heavy weight resources inside MongoDB as reading the same from the MySQL will take much more lookup time. As we have stored the images and videos in the MongoDB, retrieving them will be much more faster compared to MySQL. We also implemented redis cache to make sure the same object isn't processed many times by the server.

The policy we used to decide when to write the data into the database

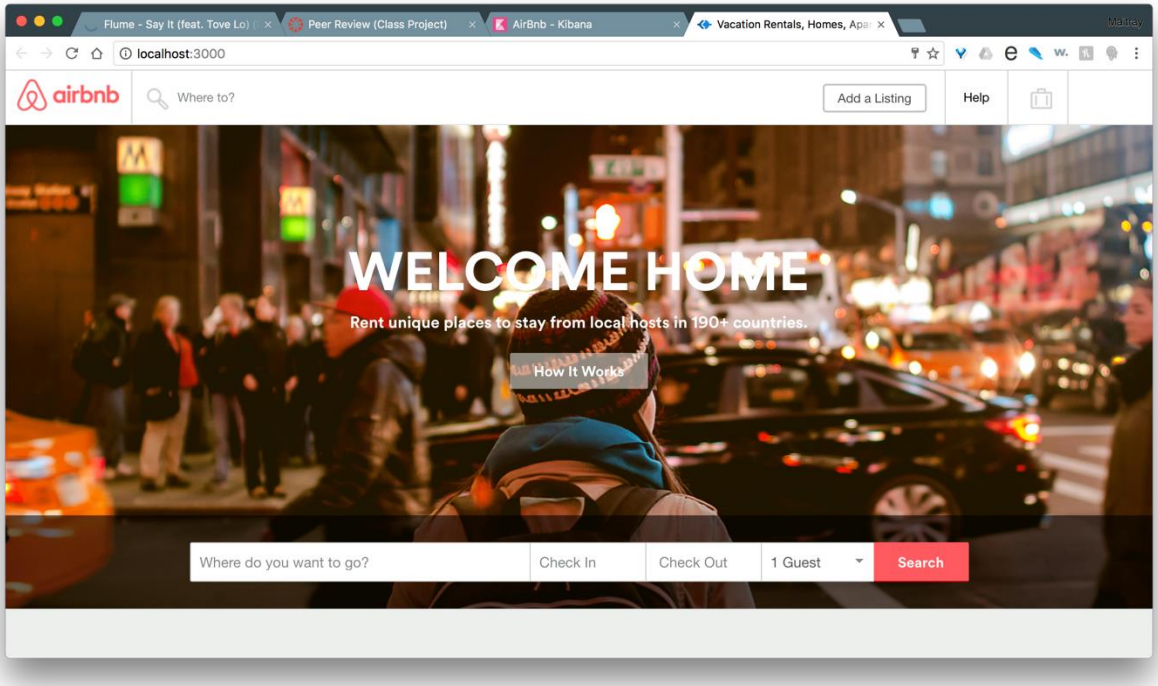
We had decided to insert the data into database optimally. For example; in the scenario like Adding a listing, we are using Mongo session to store the data provided by the Host in multiple screens and we are storing this data into the database only after all the data entry operations of all the screens get completed. Example: Add listing module has 6 screens. If we save the data in to Database in every screen then if the user terminates data entry operation in any of the next screens, then the previous entry operations need to be rolled back.

Therefore, we are storing this kind of data in session in each screen and saving the data only once in the last screen. This will reduce multiple inserts that needs to be done in previous screens and any further updates in the following screens.

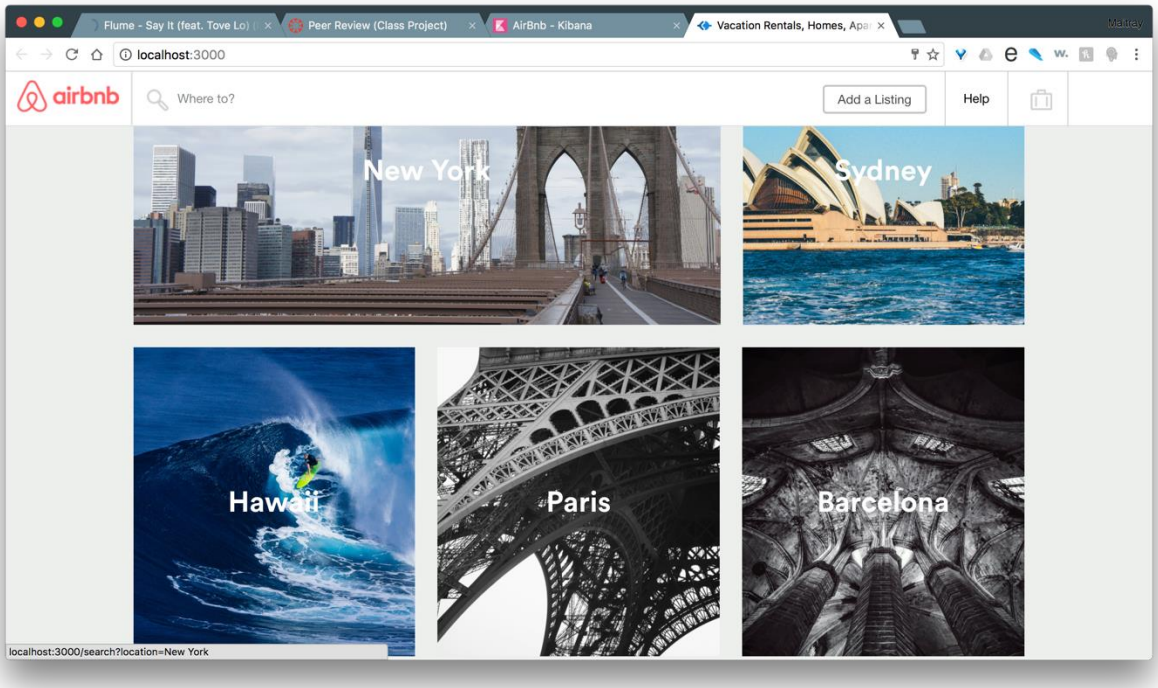
Modules

- **Homepage**

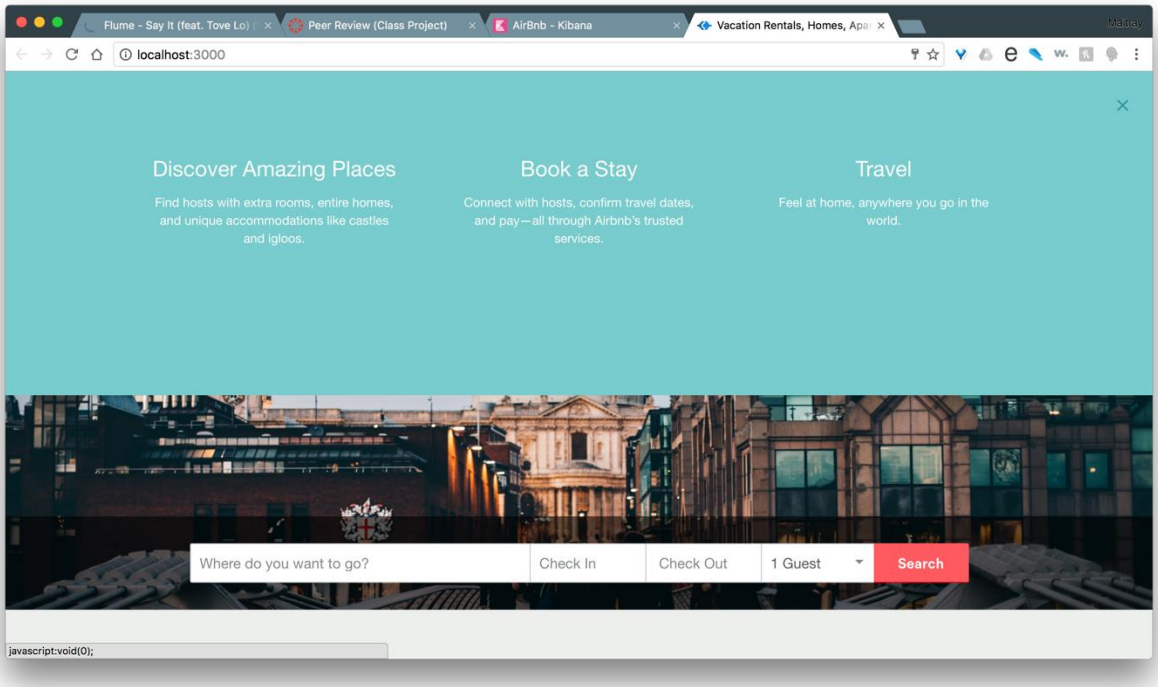
Home Page of the application. Shows all links for navigation and search for cities. Also shows different cities to navigate and search properties.



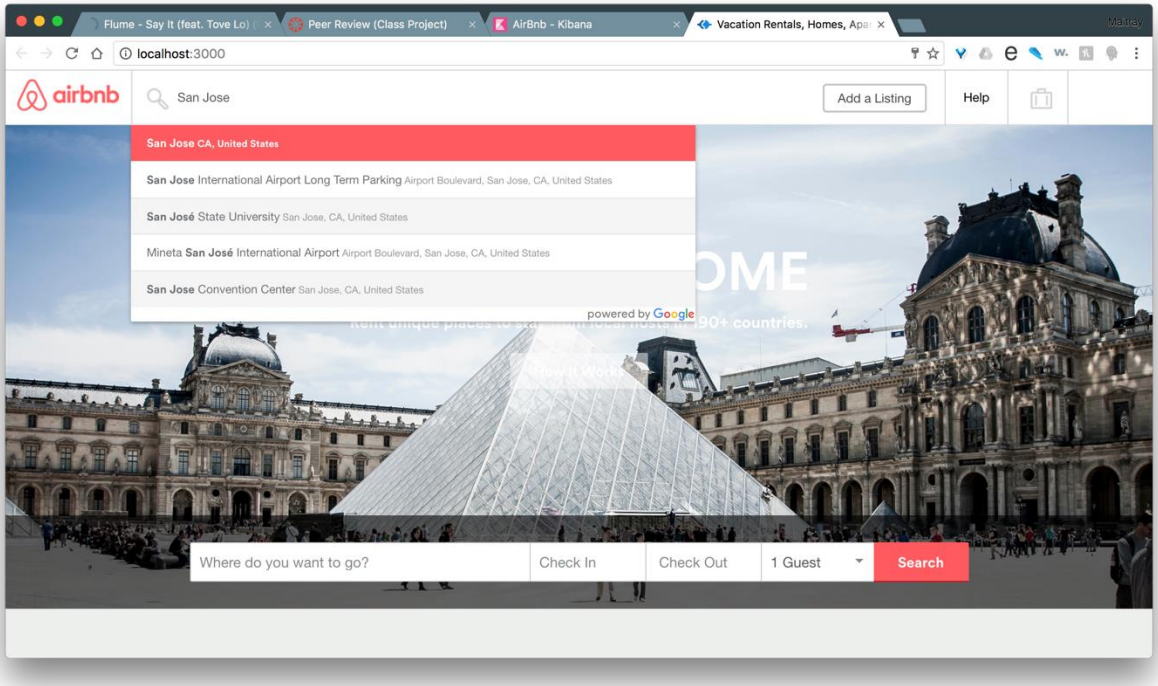
If you scroll down then you will see the Cities in the homepage



On clicking the How In Works button, This window will popup

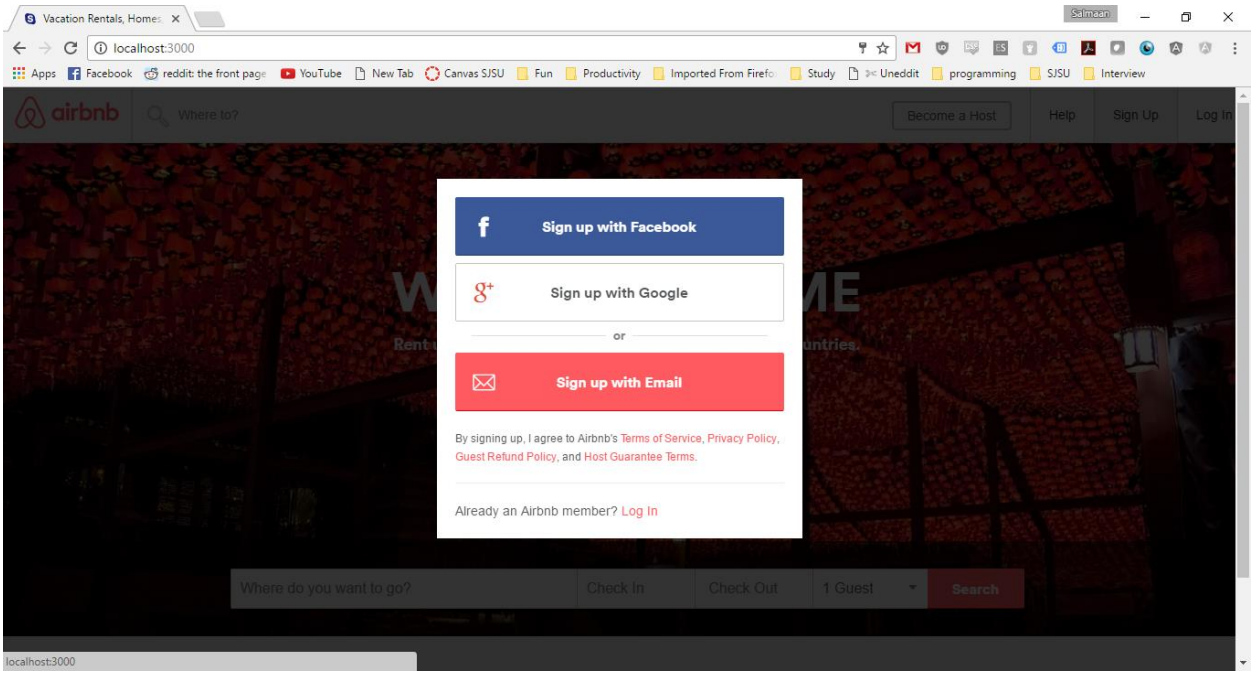


Search Navbar in the homepage



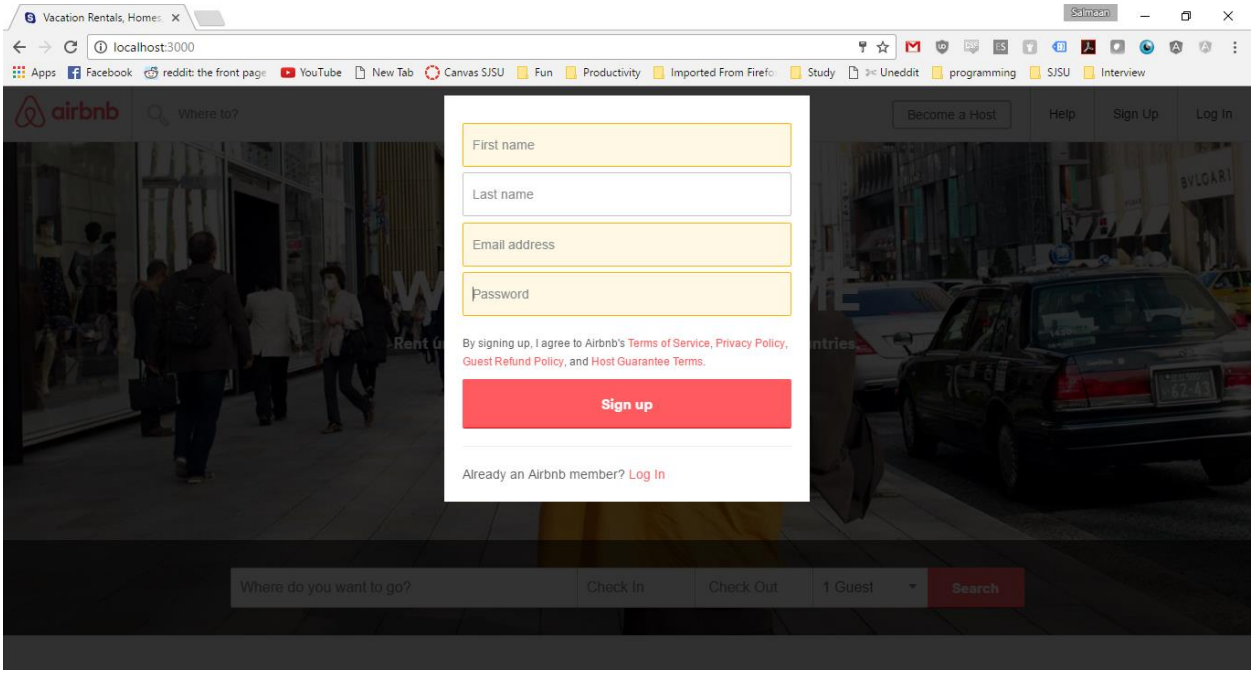
Registration Page

You can SignUp using email using the opened modal.

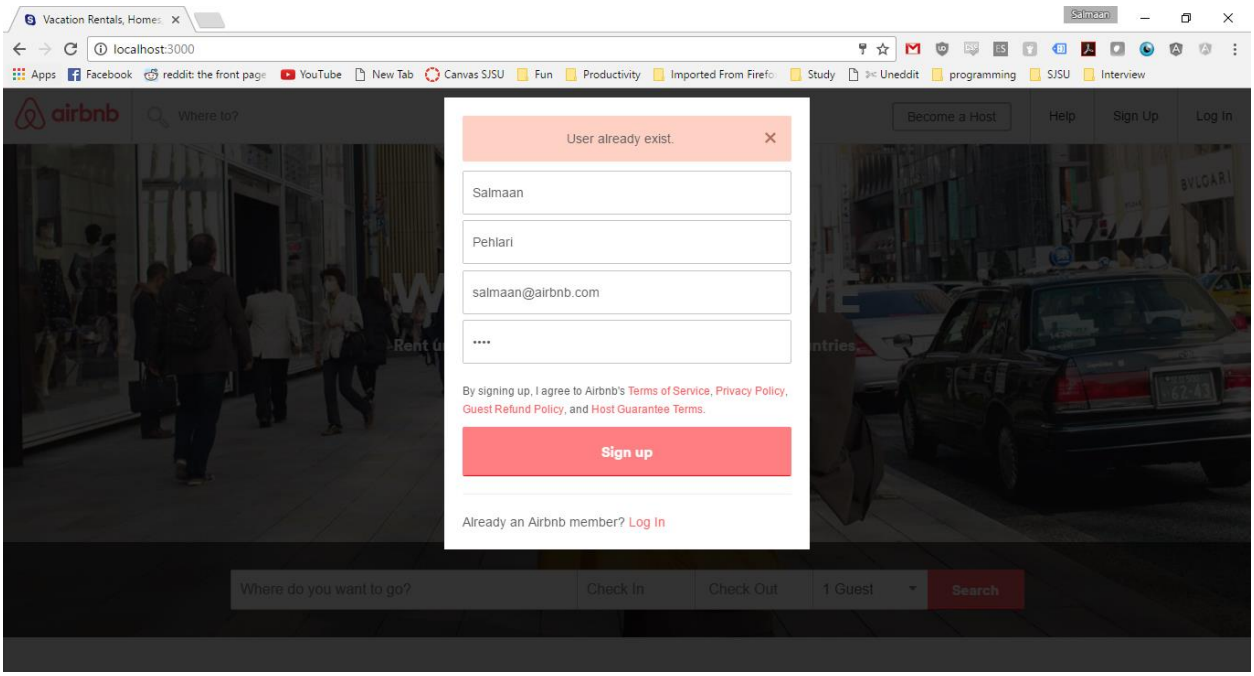


Signup with Email

If you click on sign up with Email then the following screen will popup**.**

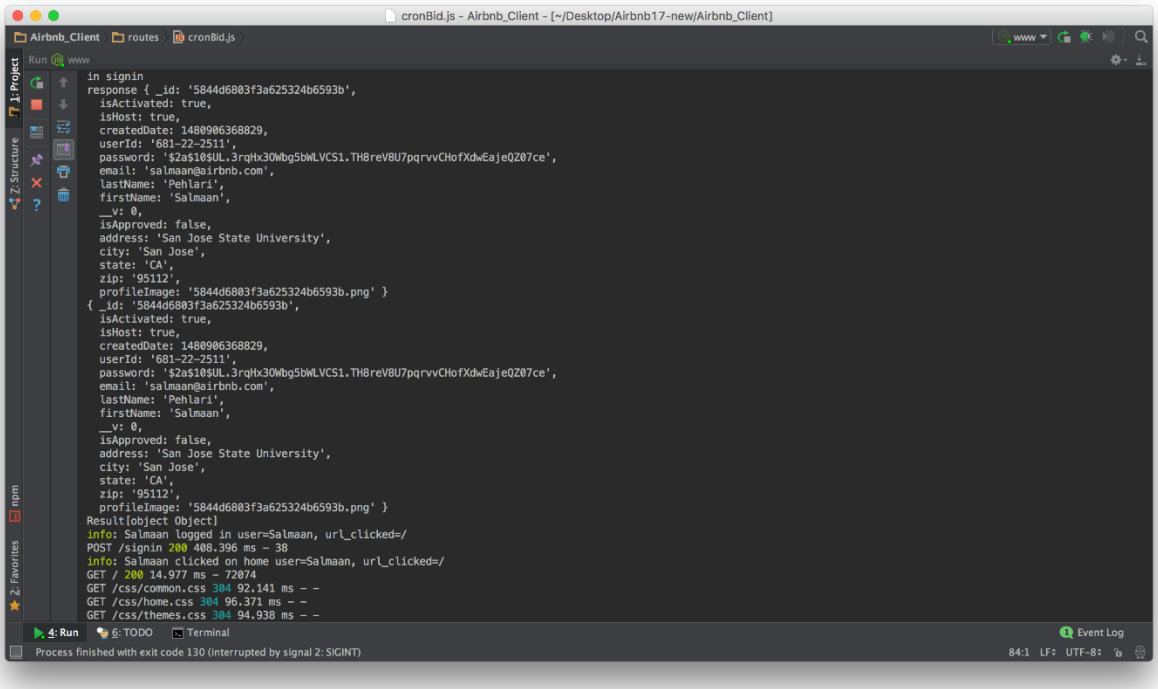


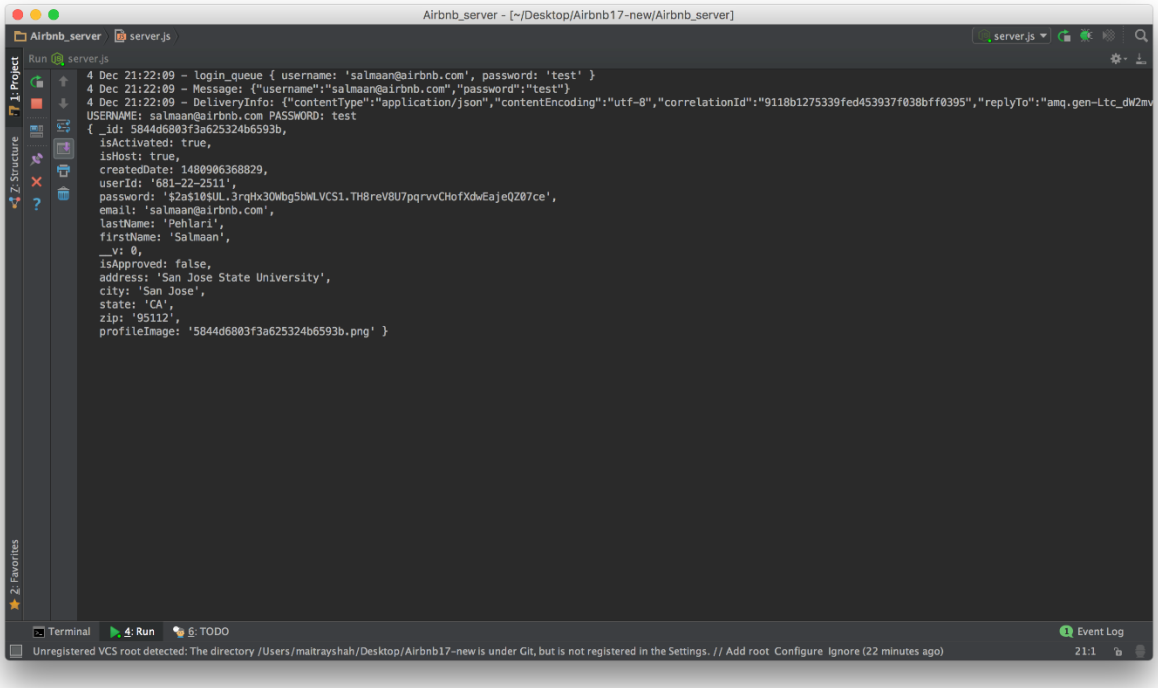
Validation in Sign Up if User Already exists



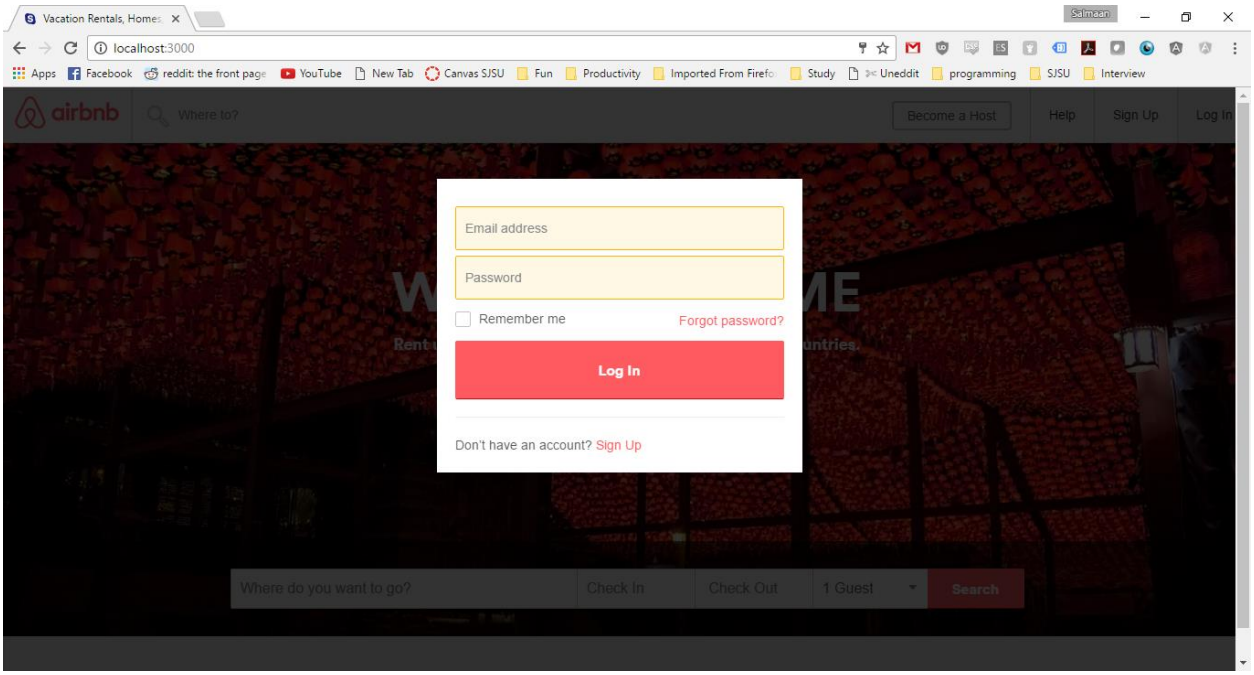
SERVER CONSOLE AND CLIENT CONSOLE SCREENSHOTS

1. Sign In Call

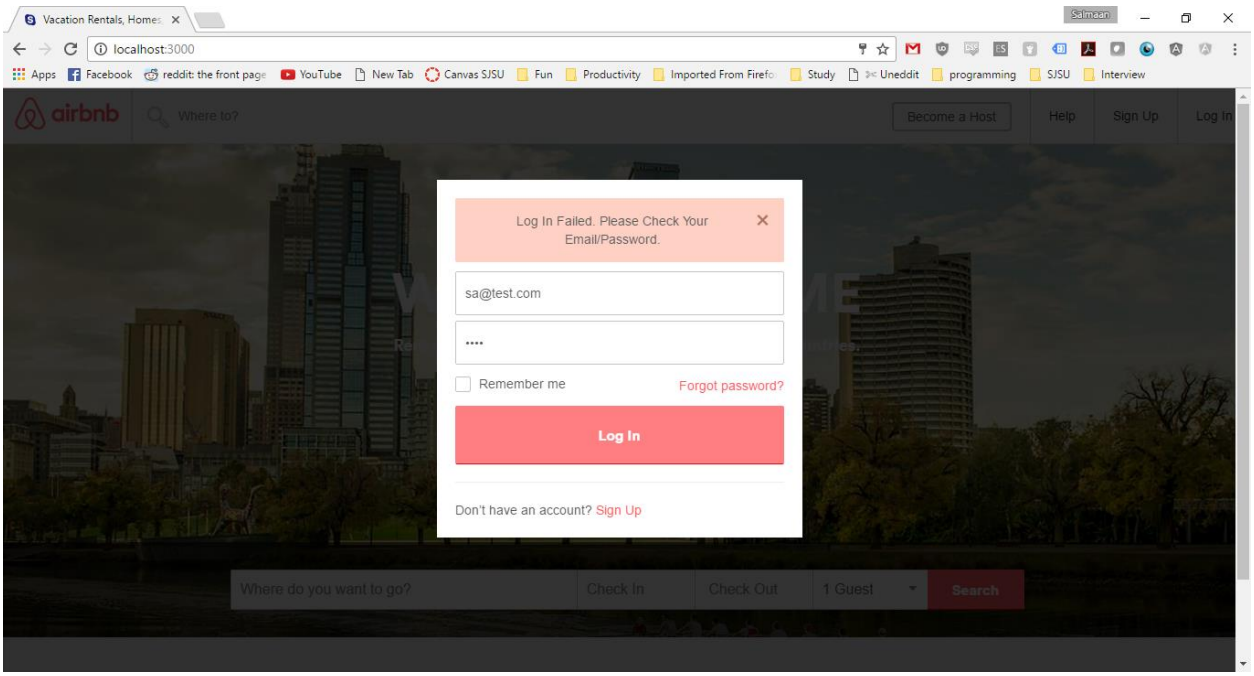




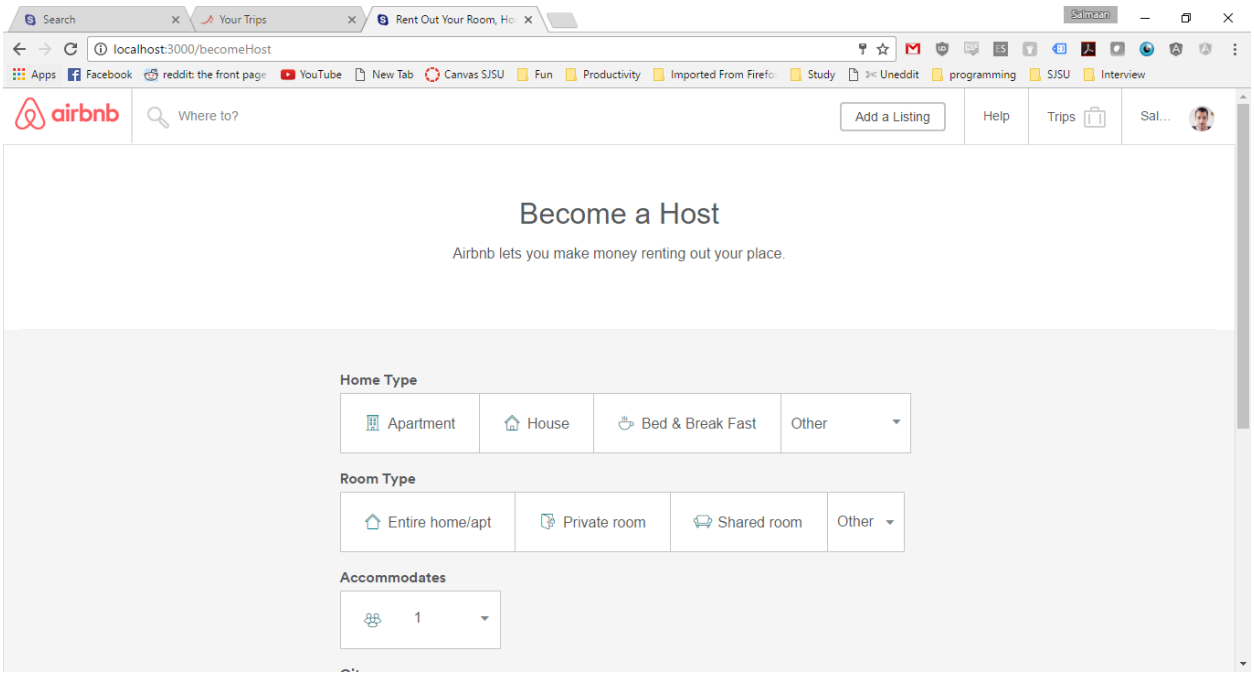
Login Page Screenshots

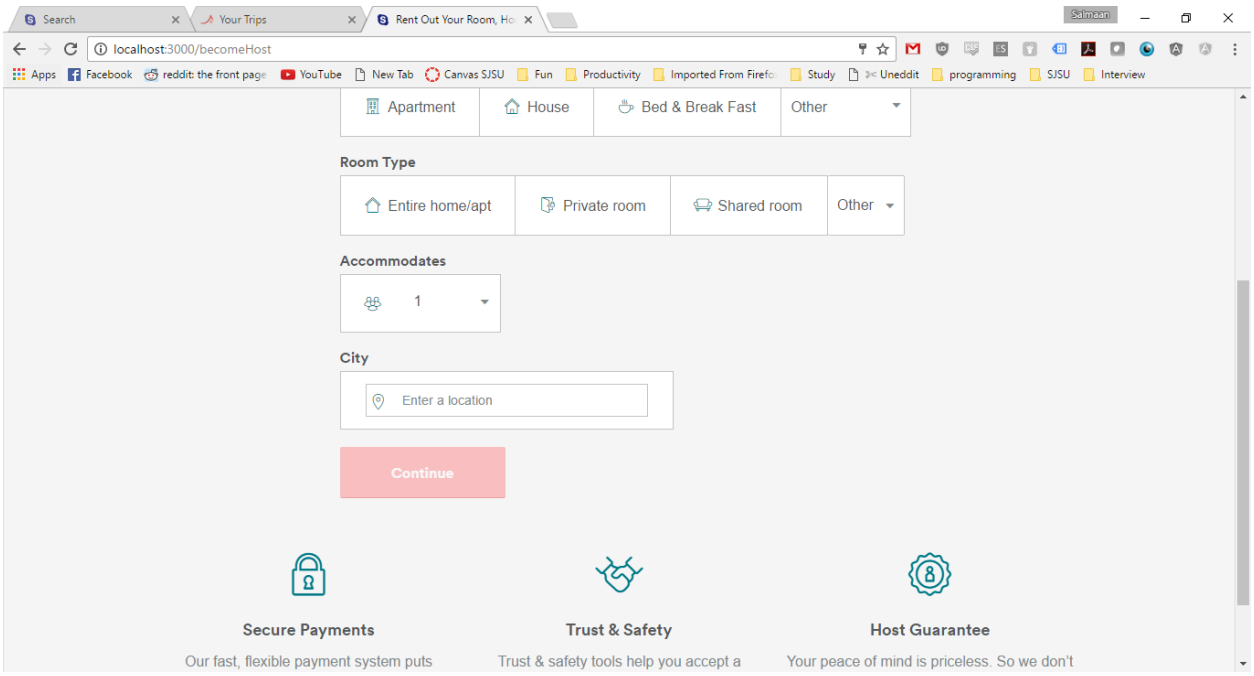


Validation in Login Page

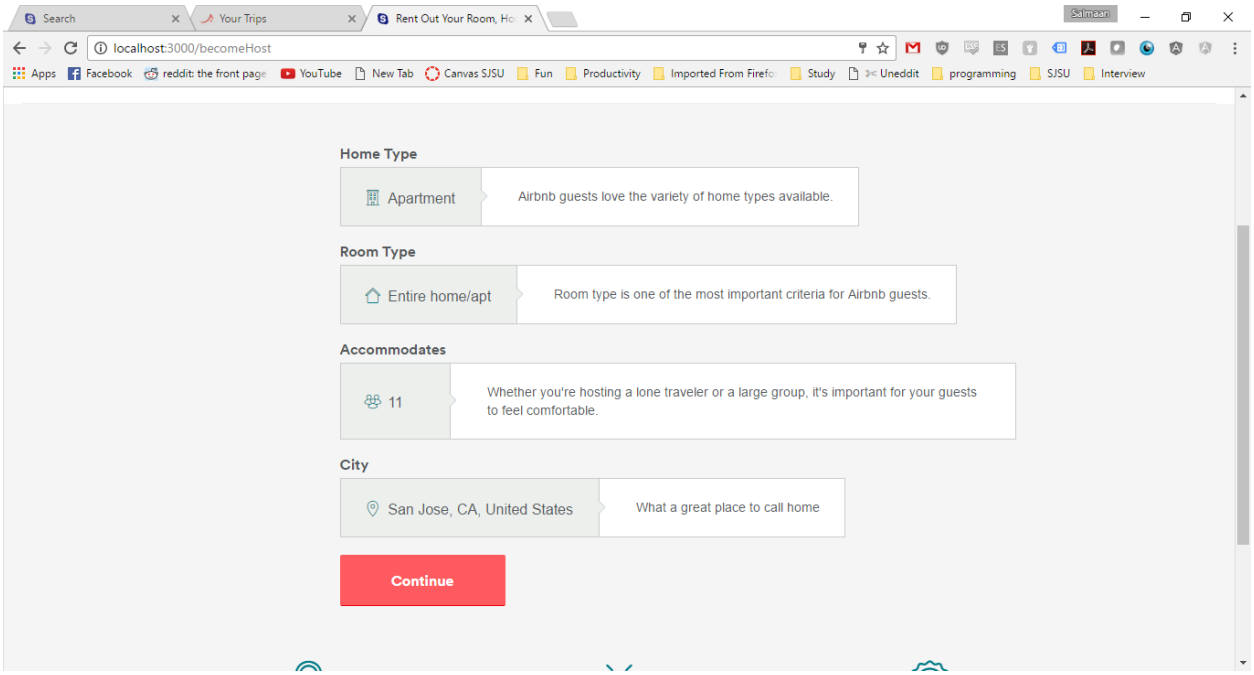


On clicking the Add a Listing button, the following page will appear

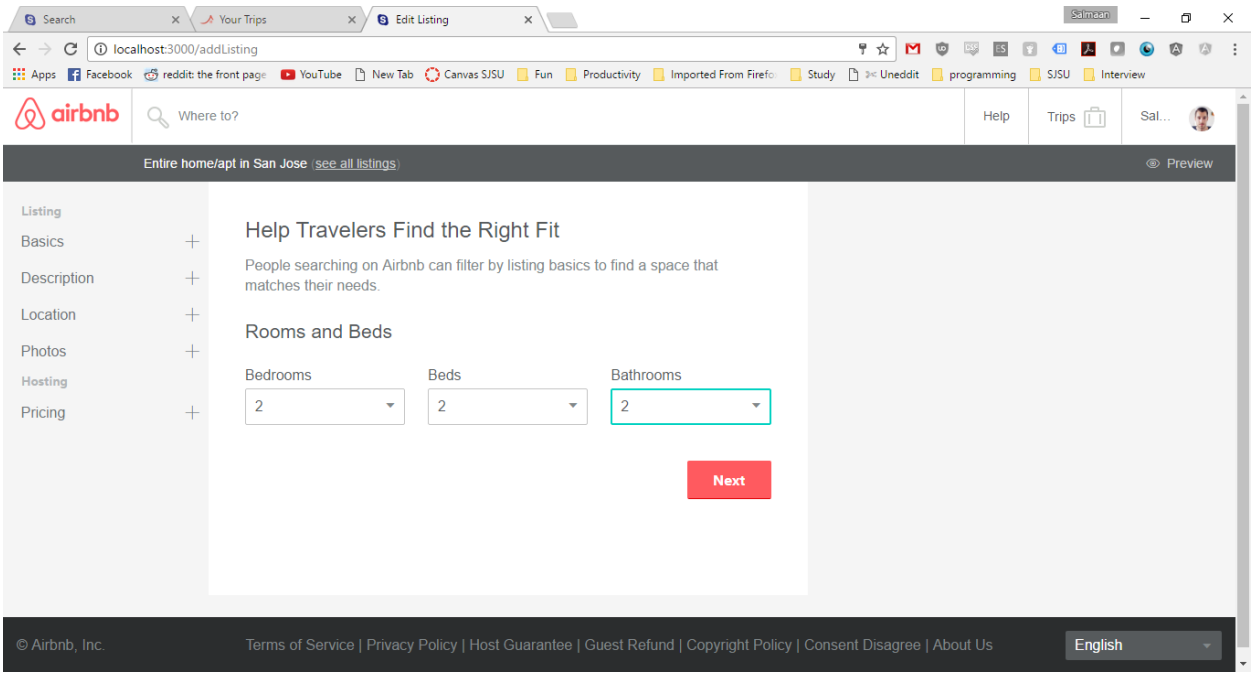




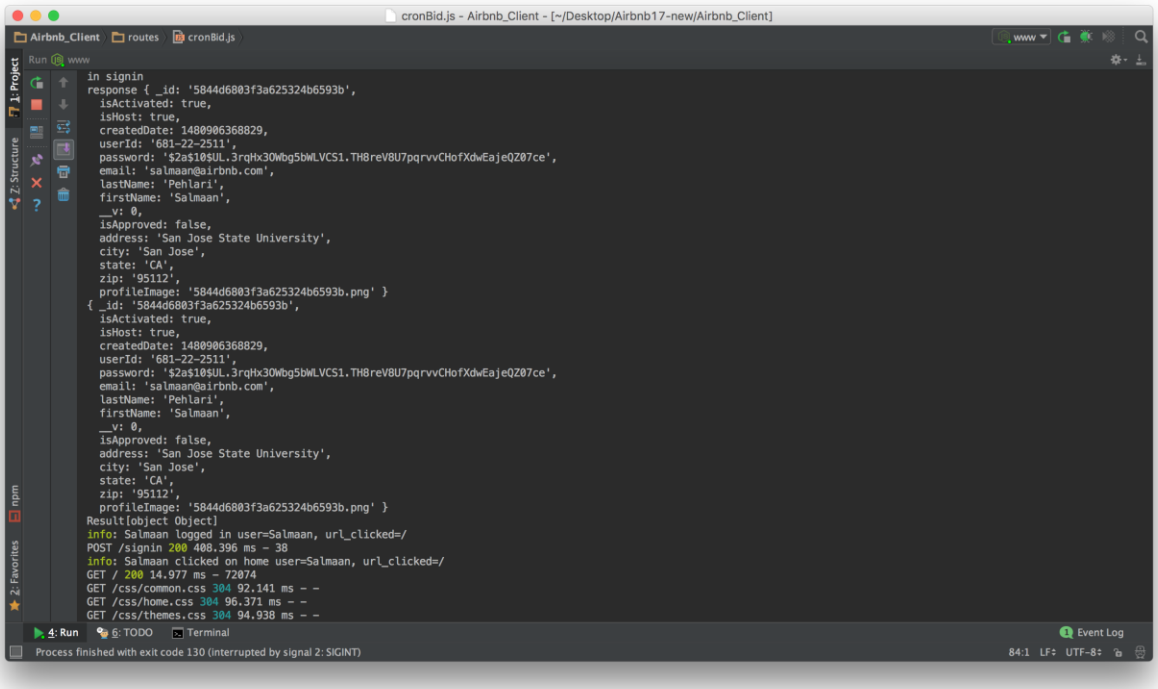
Example of entering the details

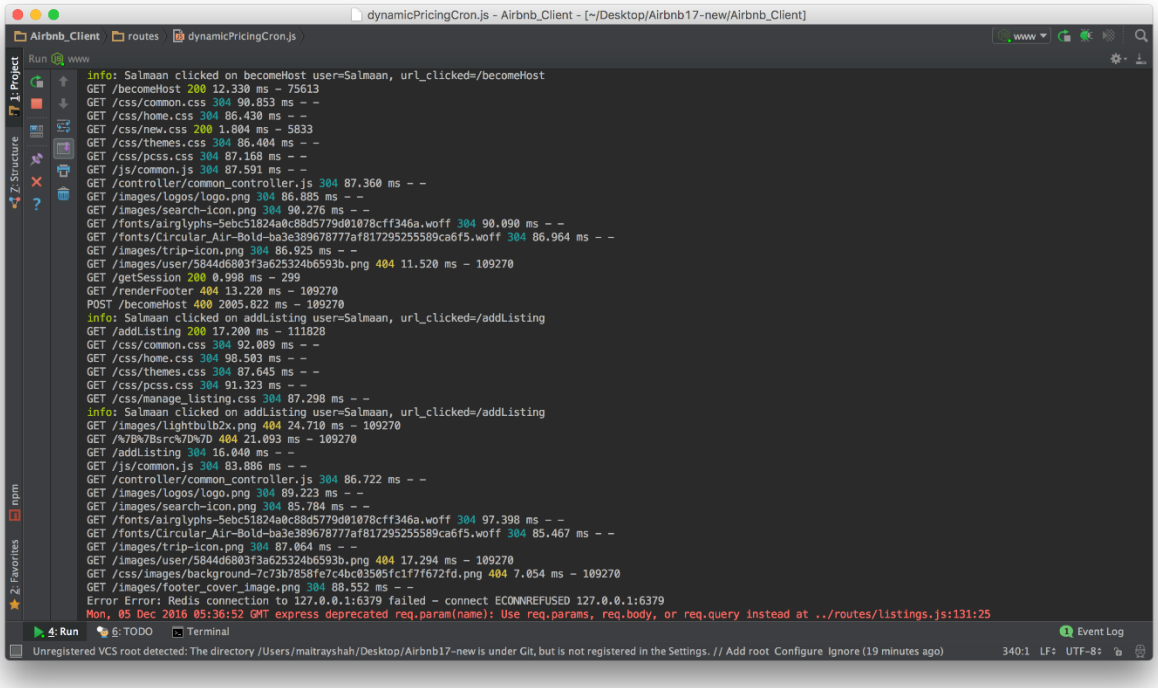


Basic Tab in Adding a Listing

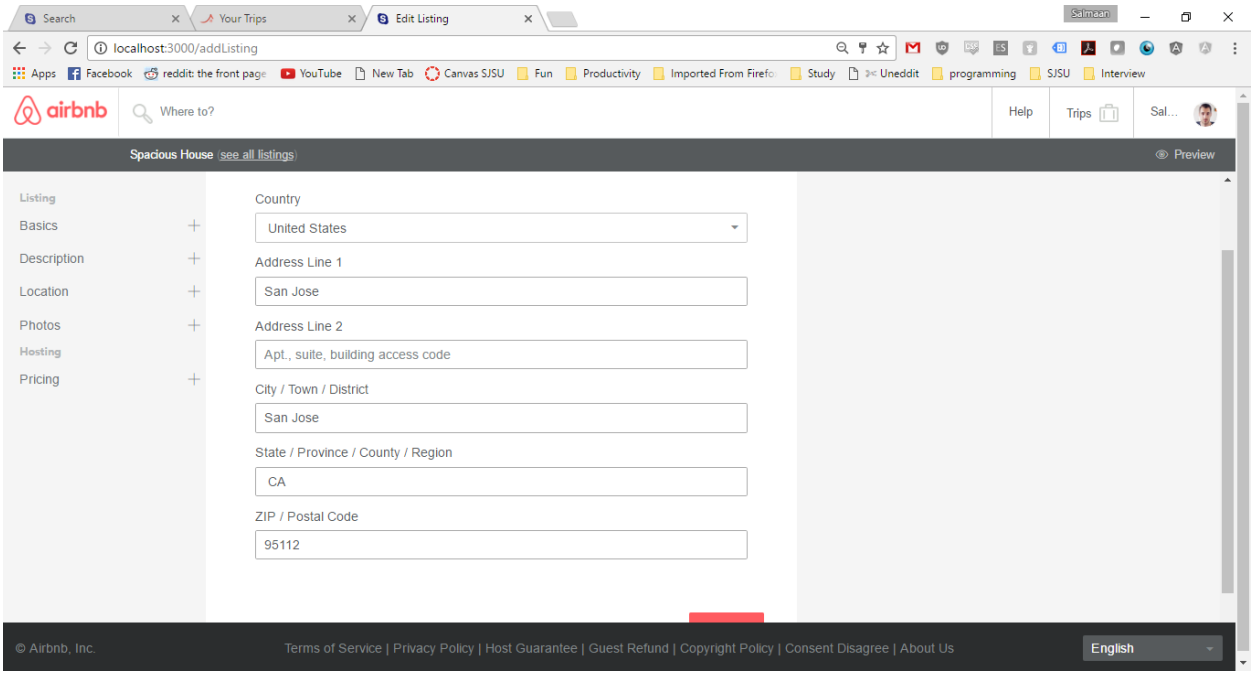


SERVER AND CLIENT CONSOLE Description tab in Adding a Listing



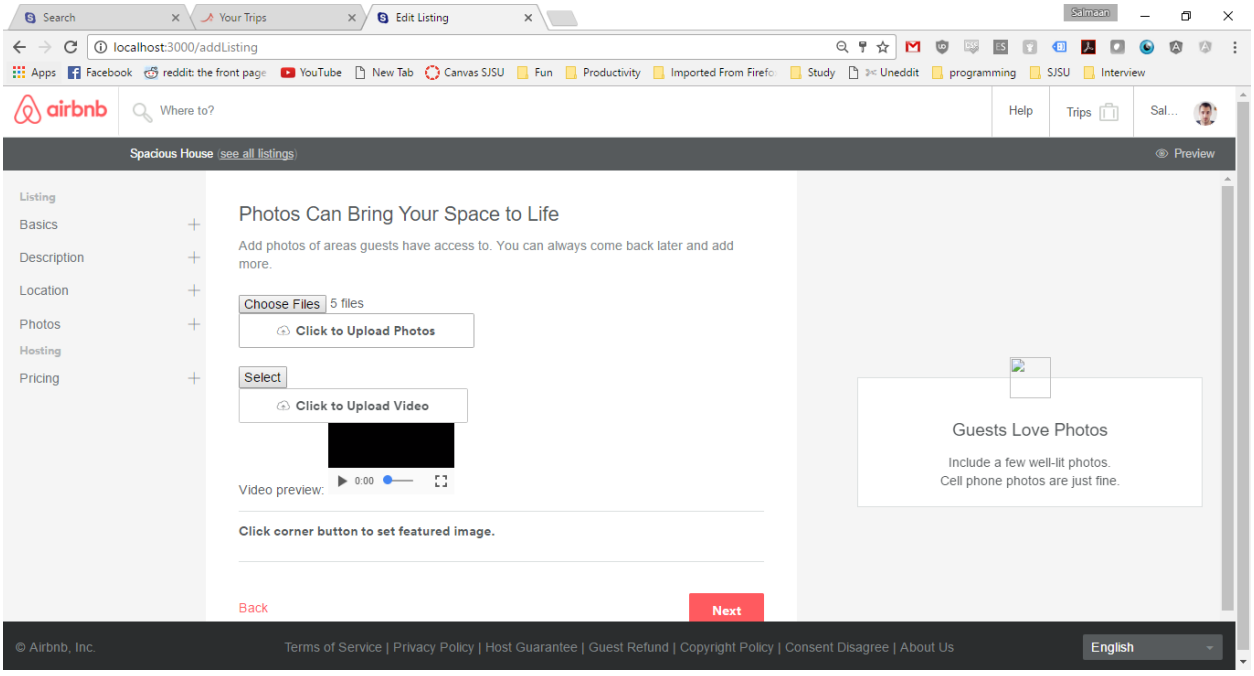


Location Tab in Adding a Listing

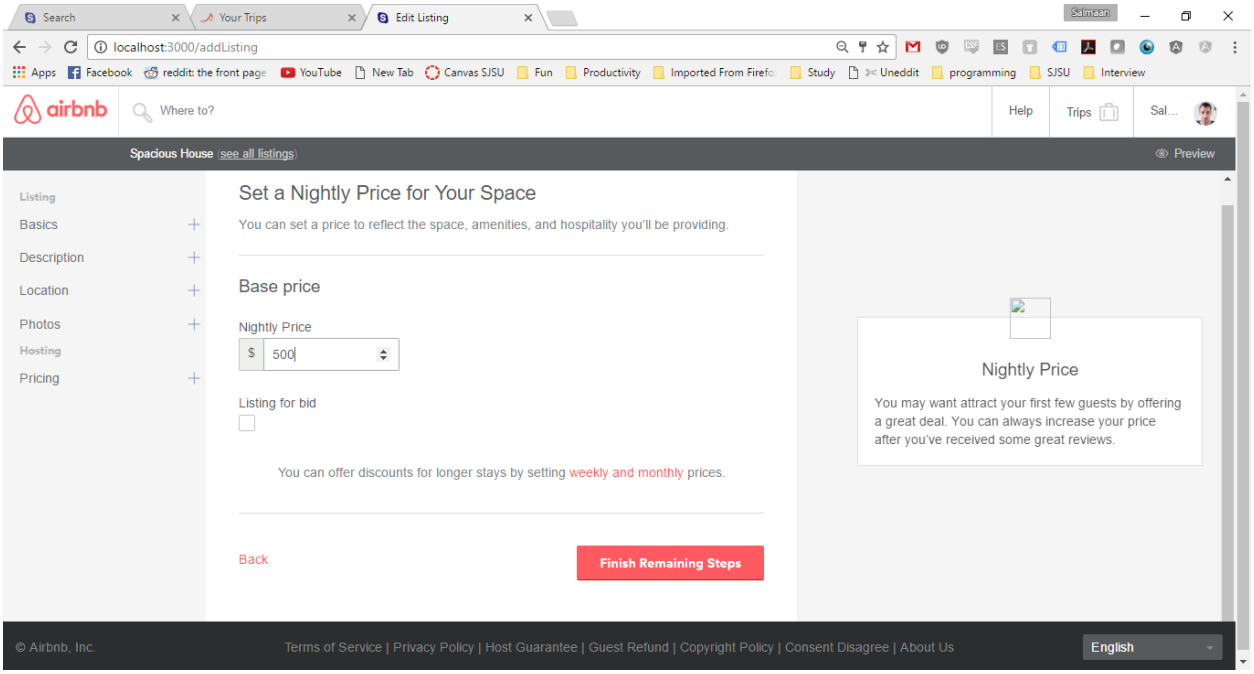


Photos Tab in Adding a Listing

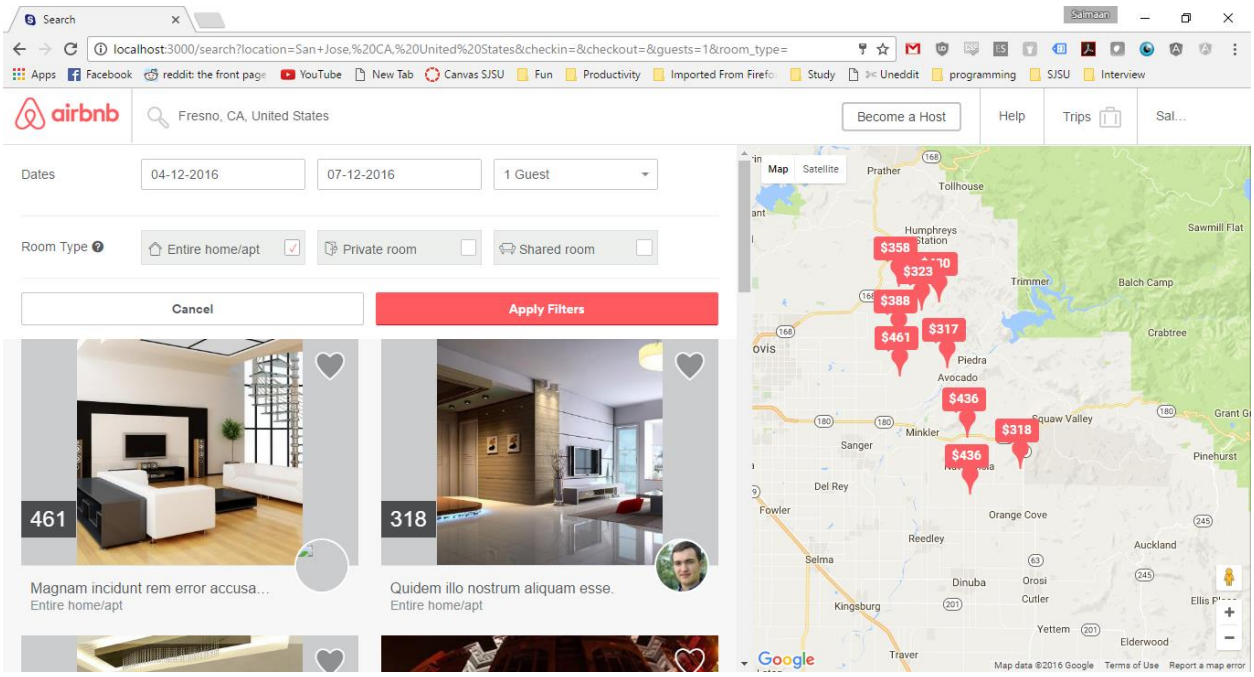
Host can add multiple images and video in this screen



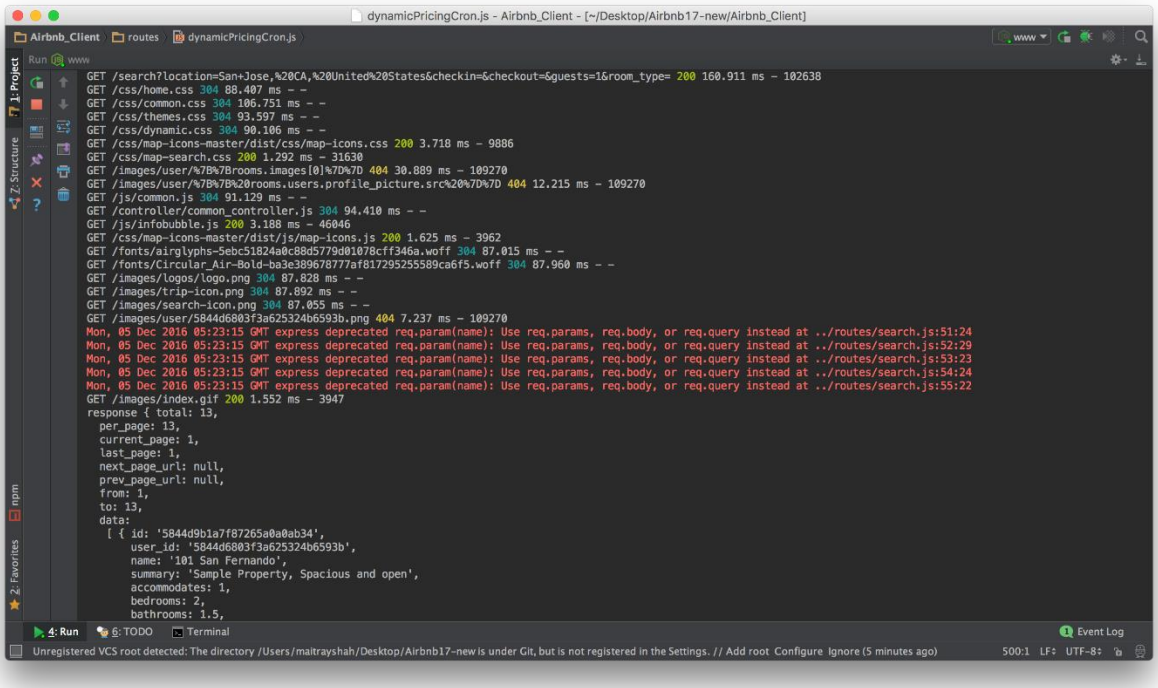
Pricing Tab in Adding a Listing

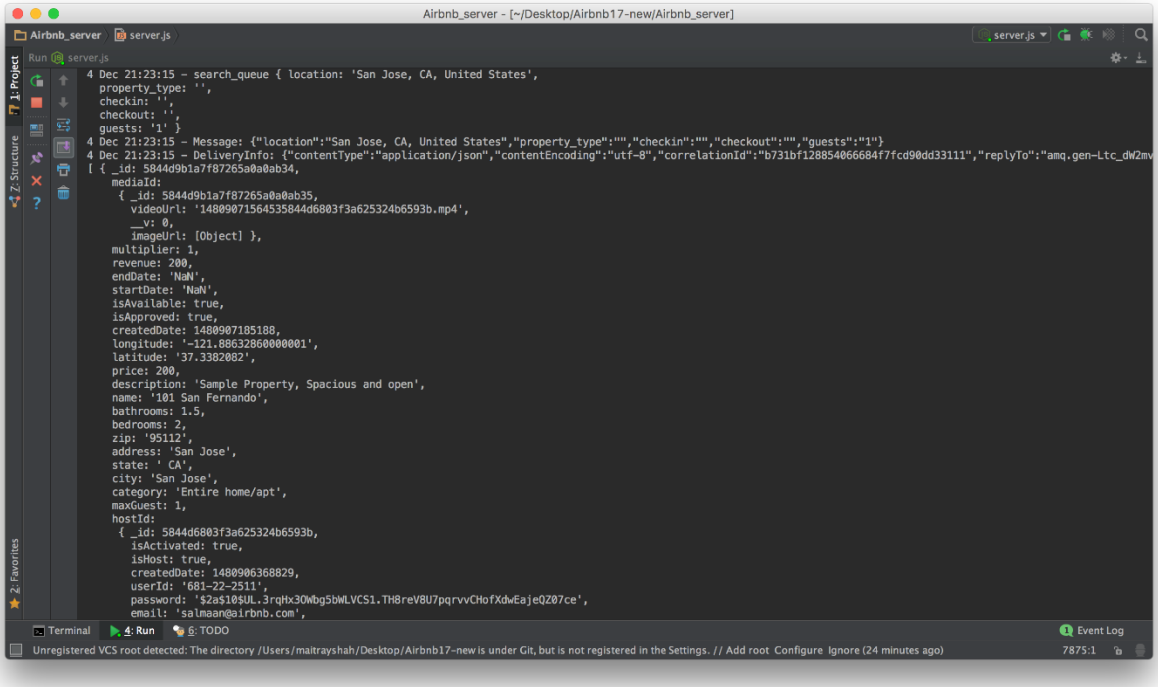


After searching a property in Search bar and applying filters the following screen will appear Pinpoint pricing is available in the Maps.

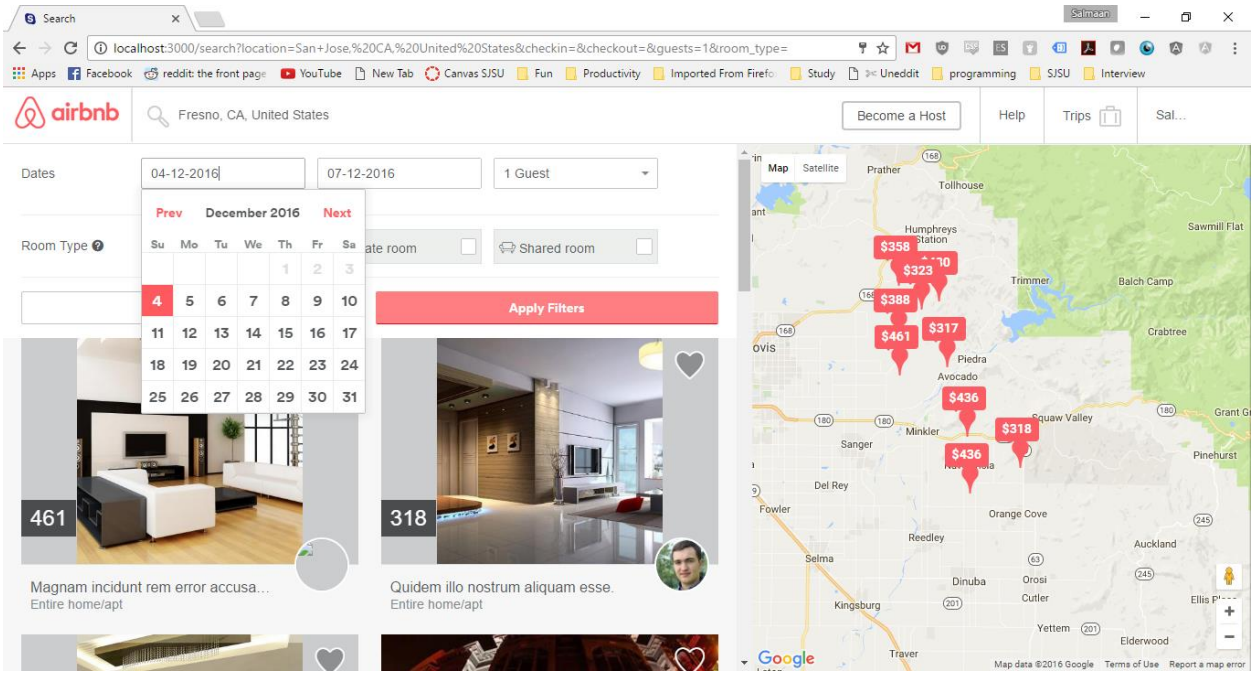


SERVER AND CLIENT CONSOLE FOR SEARCH QUERY

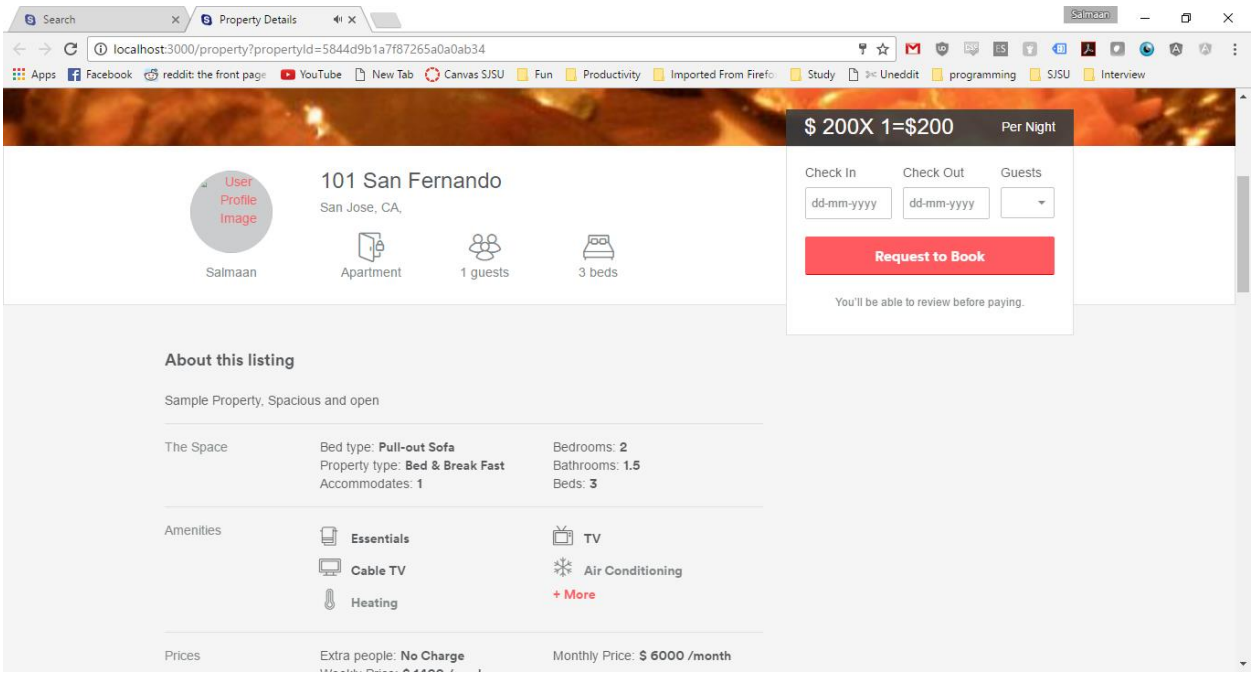


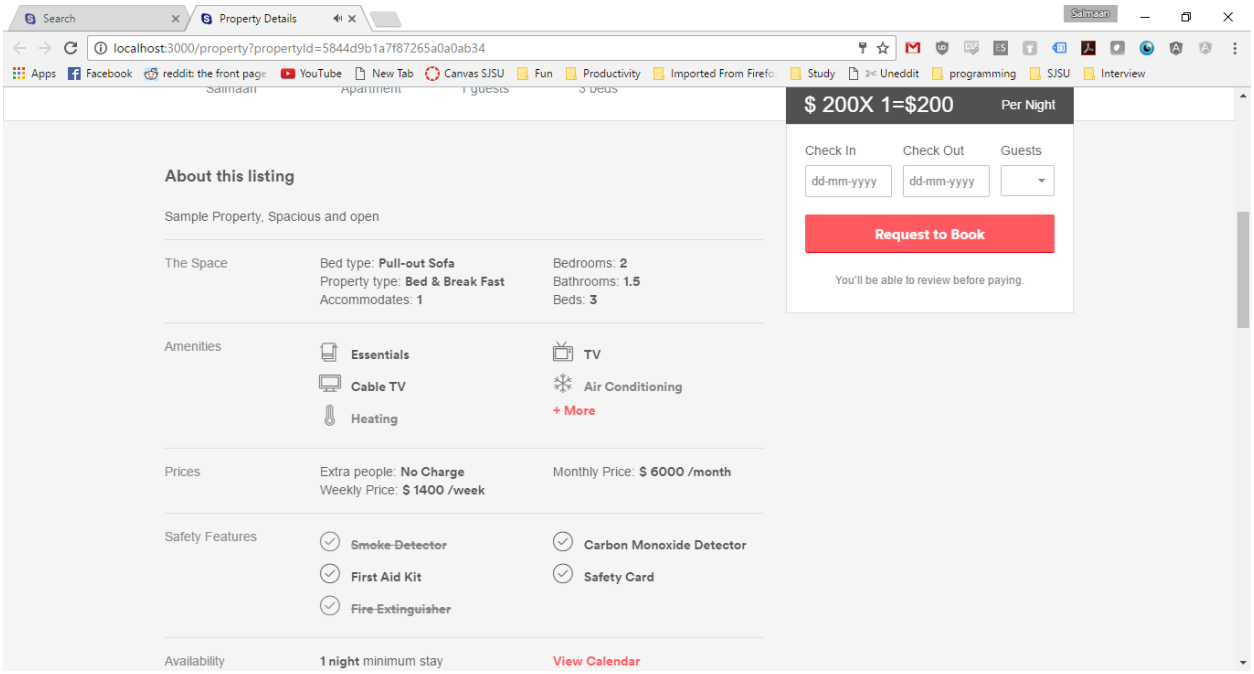


Adding Check In and Check Out Dates in filters

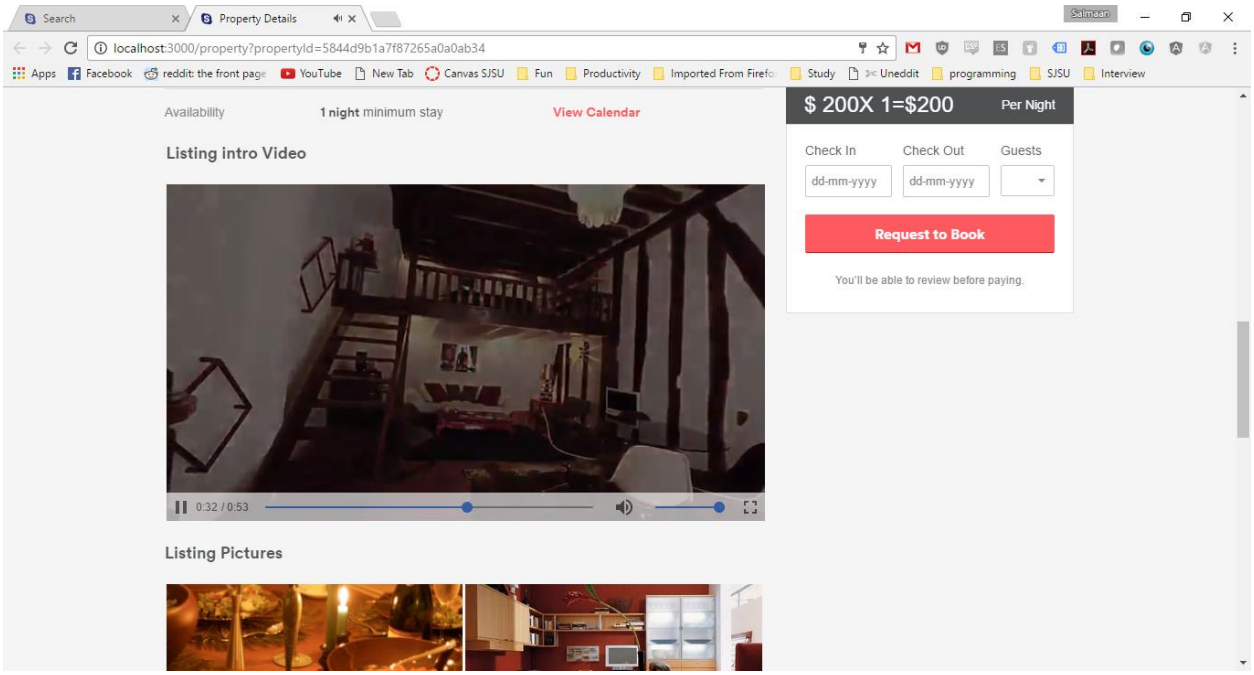


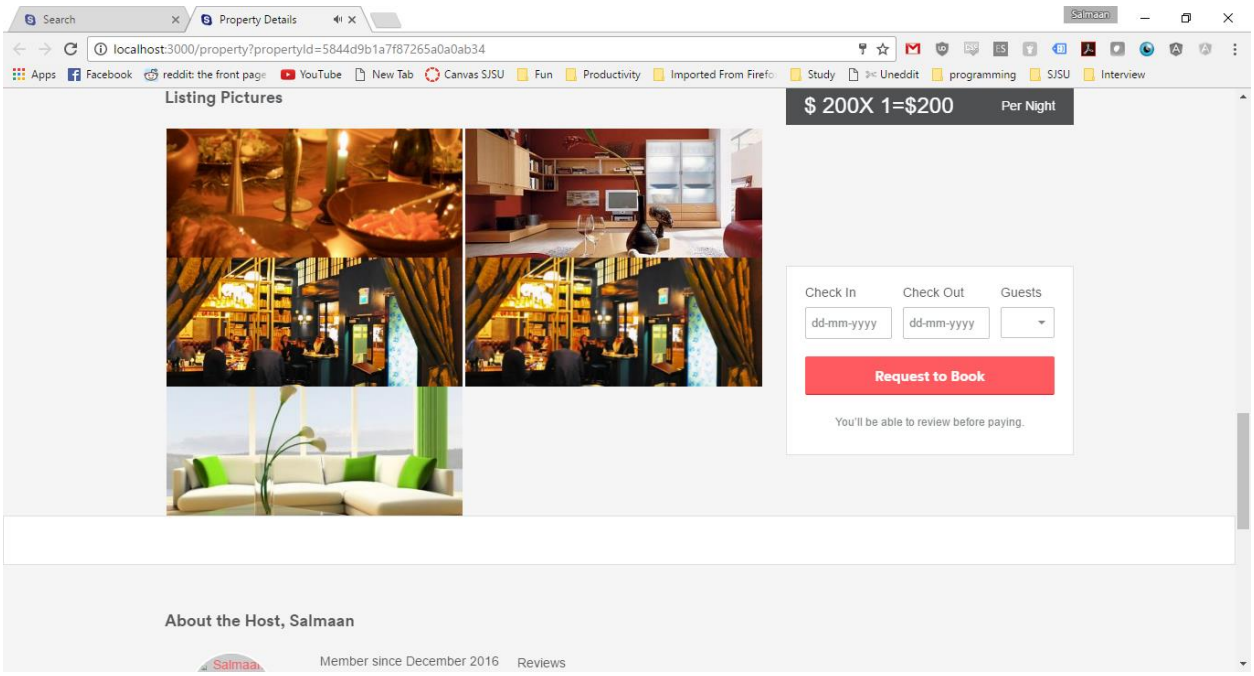
On clicking the property Image The Property Details page will appear


Search
Property Details
Salmaan
localhost:3000/property?propertyId=5844d9b1a7f87265a0a0ab34
Apps
Facebook
reddit: the front page
YouTube
New Tab
Canvas SJSU
Fun
Productivity
Imported From Firefo
Study
Uneddit
programming
SJSU
Interview
$ 200X 1=$200
Per Night
Check In
Check Out
Guests
dd-mm-yyyy
dd-mm-yyyy
Request to Book
You'll be able to review before paying.
User Profile Image
Salmaan
101 San Fernando
San Jose, CA,
Apartment
1 guests
3 beds
About this listing
Sample Property, Spacious and open
The Space
Bed type: Pull-out Sofa
Property type: Bed & Break Fast
Accommodates: 1
Bedrooms: 2
Bathrooms: 1.5
Beds: 3
Amenities
Essentials
Cable TV
Heating
TV
Air Conditioning
+ More
Prices
Extra people: No Charge
Monthly Price: $ 6000 /month

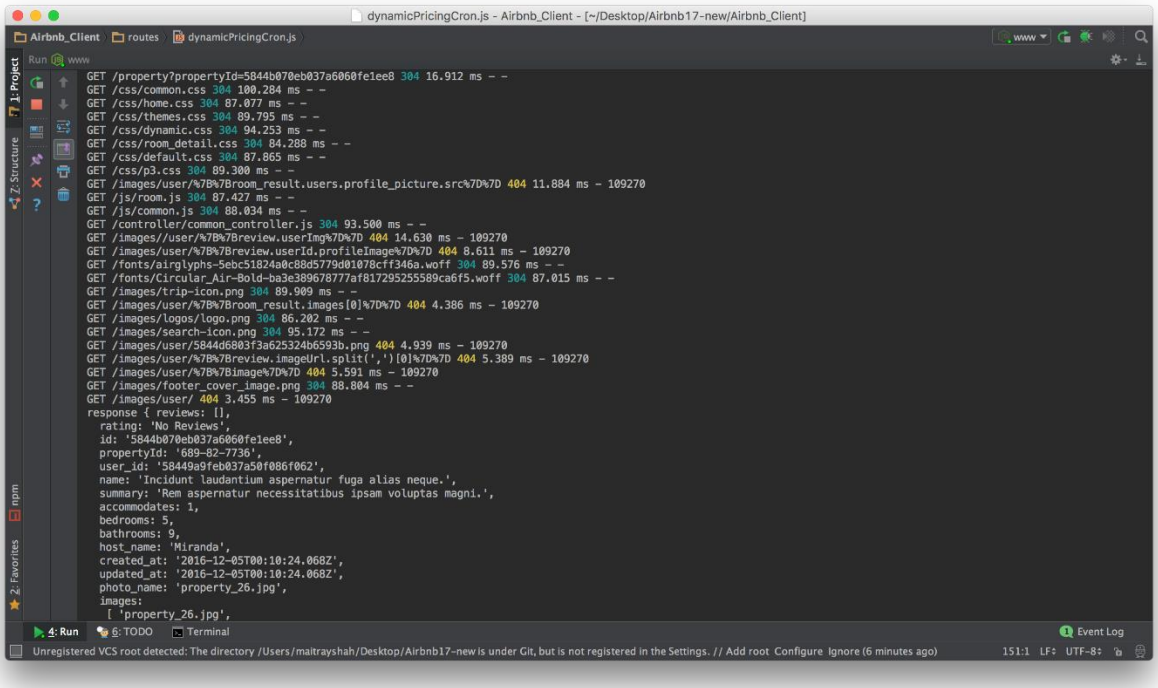

Search
Property Details
Salmaan
localhost:3000/property?propertyId=5844d9b1a7f87265a0a0ab34
Apps
Facebook
reddit: the front page
YouTube
New Tab
Canvas SJSU
Fun
Productivity
Imported From Firefo
Study
Uneddit
programming
SJSU
Interview
$ 200X 1=$200
Per Night
Check In
Check Out
Guests
dd-mm-yyyy
dd-mm-yyyy
Request to Book
You'll be able to review before paying.
About this listing
Sample Property, Spacious and open
The Space
Bed type: Pull-out Sofa
Property type: Bed & Break Fast
Accommodates: 1
Bedrooms: 2
Bathrooms: 1.5
Beds: 3
Amenities
Essentials
Cable TV
Heating
TV
Air Conditioning
+ More
Prices
Extra people: No Charge
Weekly Price: $ 1400 /week
Monthly Price: $ 6000 /month
Safety Features
Smoke Detector
First Aid Kit
Fire Extinguisher
Carbon Monoxide Detector
Safety Card
Availability
1 night minimum stay
View Calendar

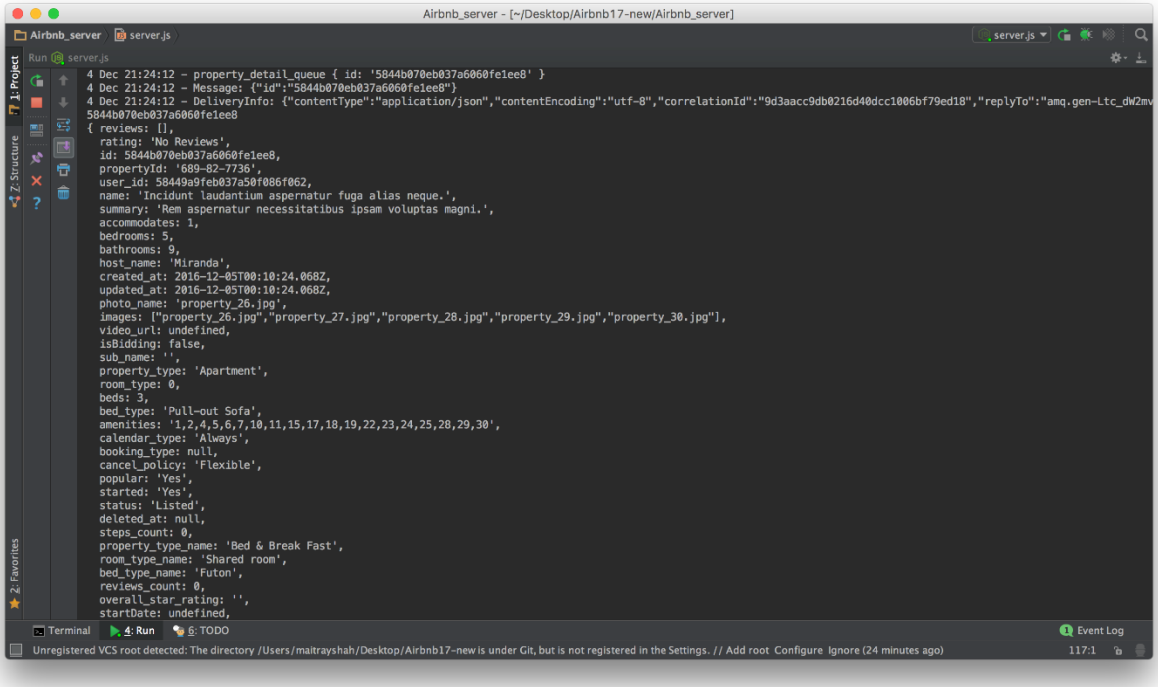
Photos and Video in Property Detail page





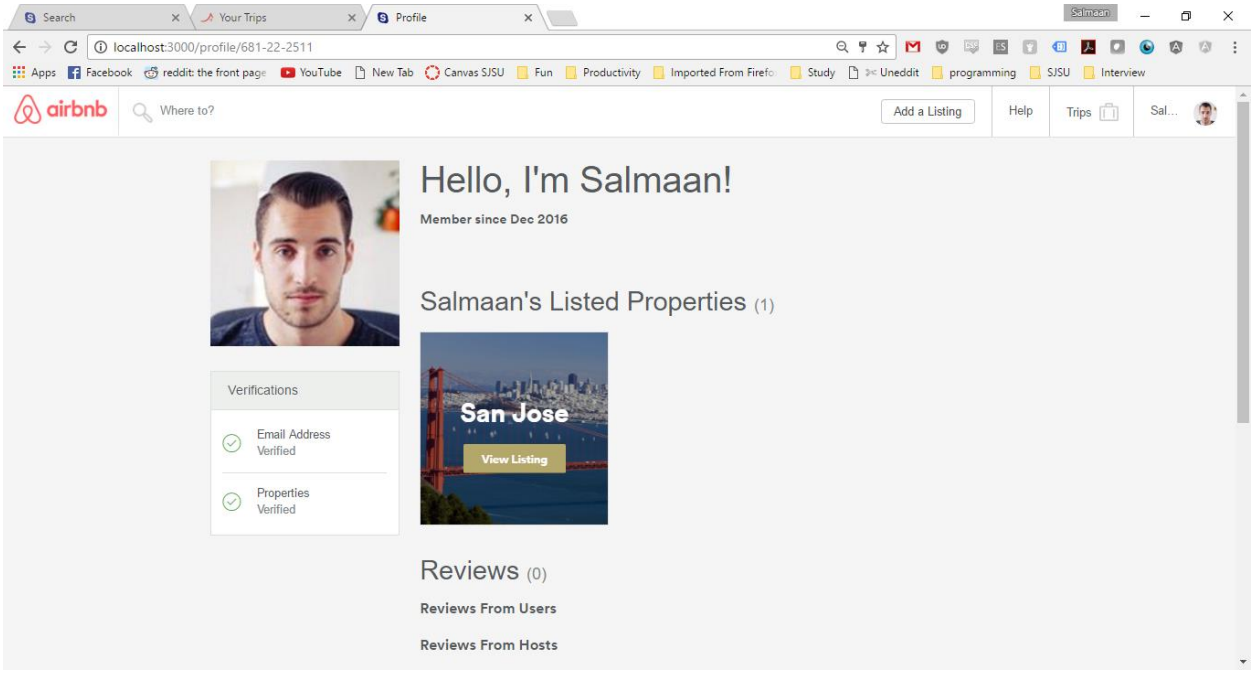
SERVER AND CLIENT CONSOLE

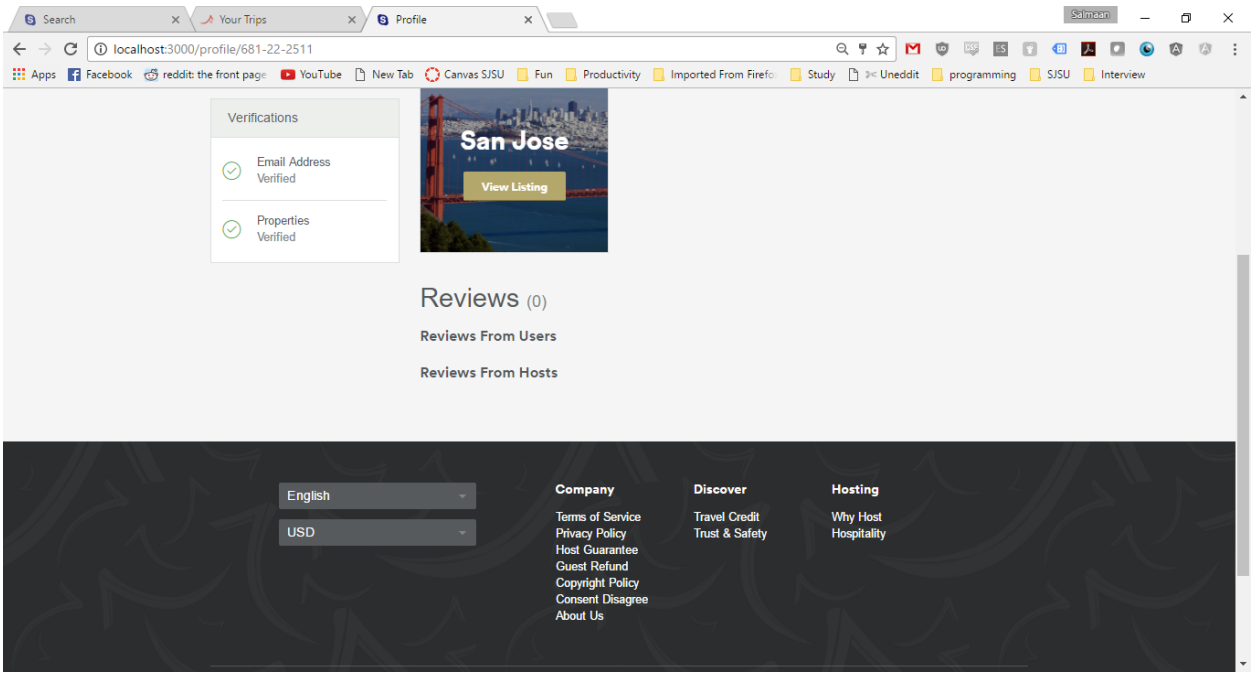




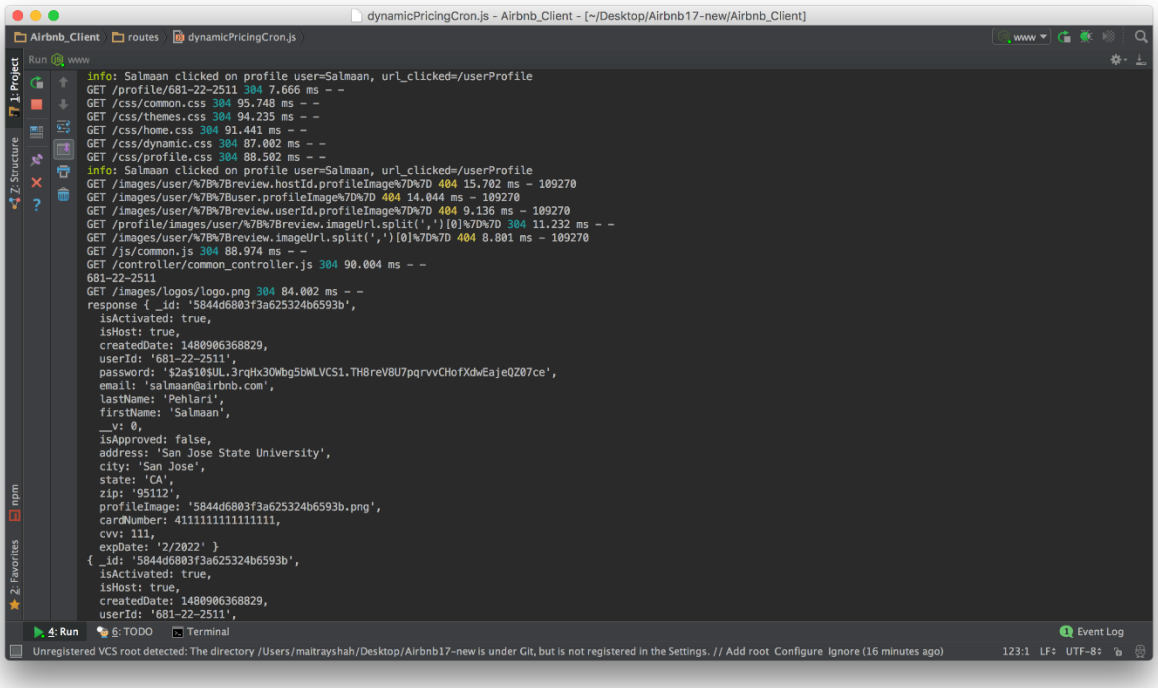
Profile page

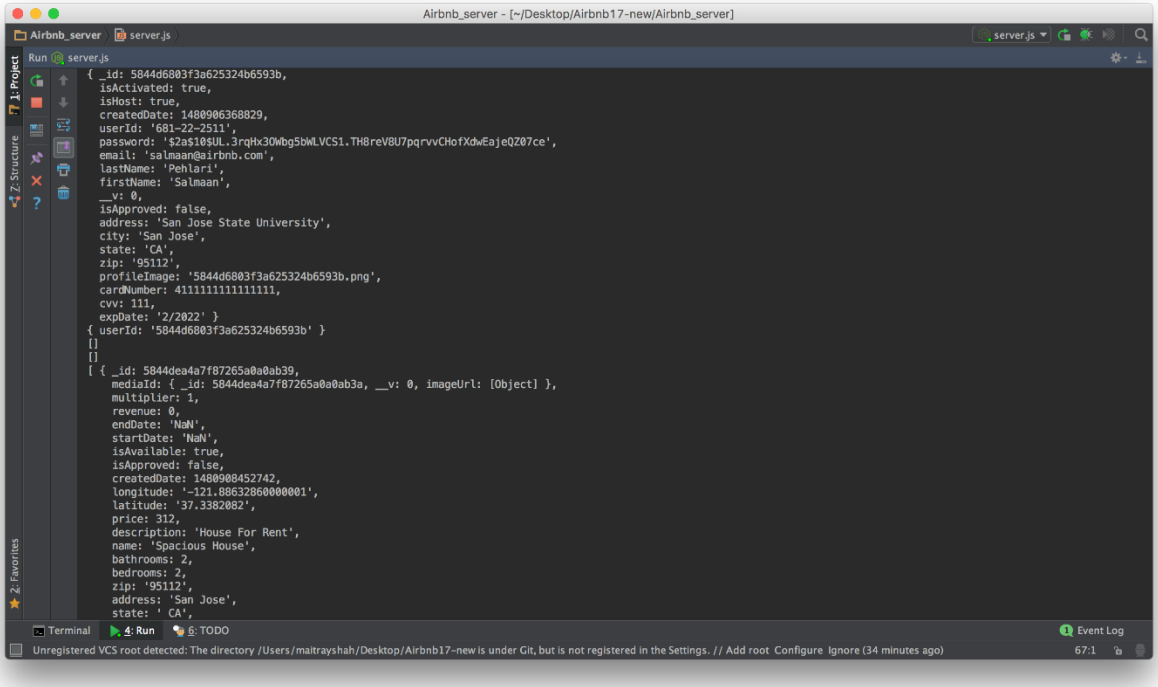
Profile Page will display User is Listed Properties



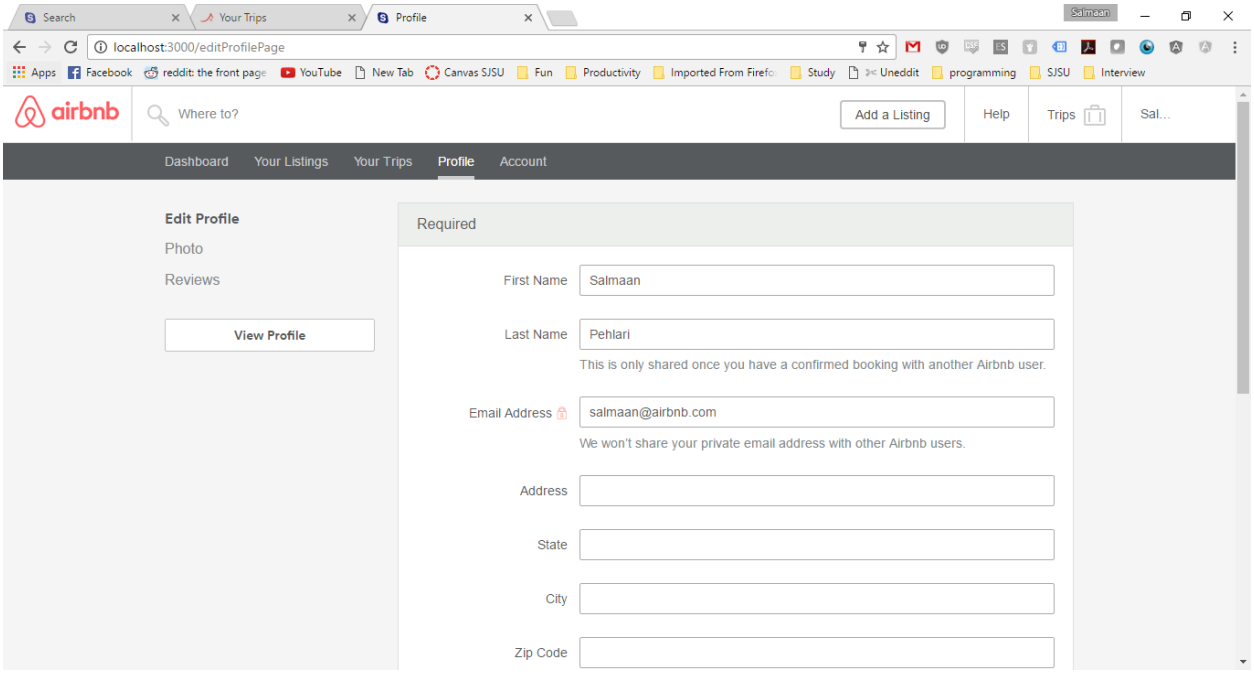


CLIENT AND SERVER CONSOLE CALL

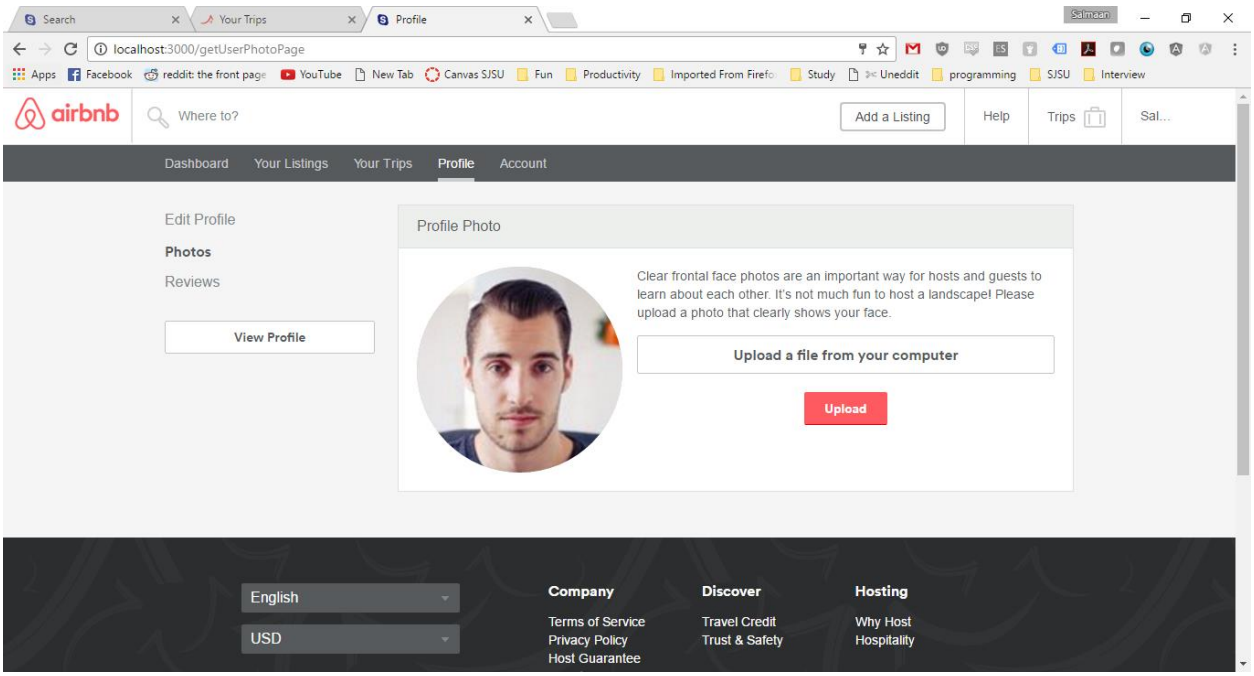




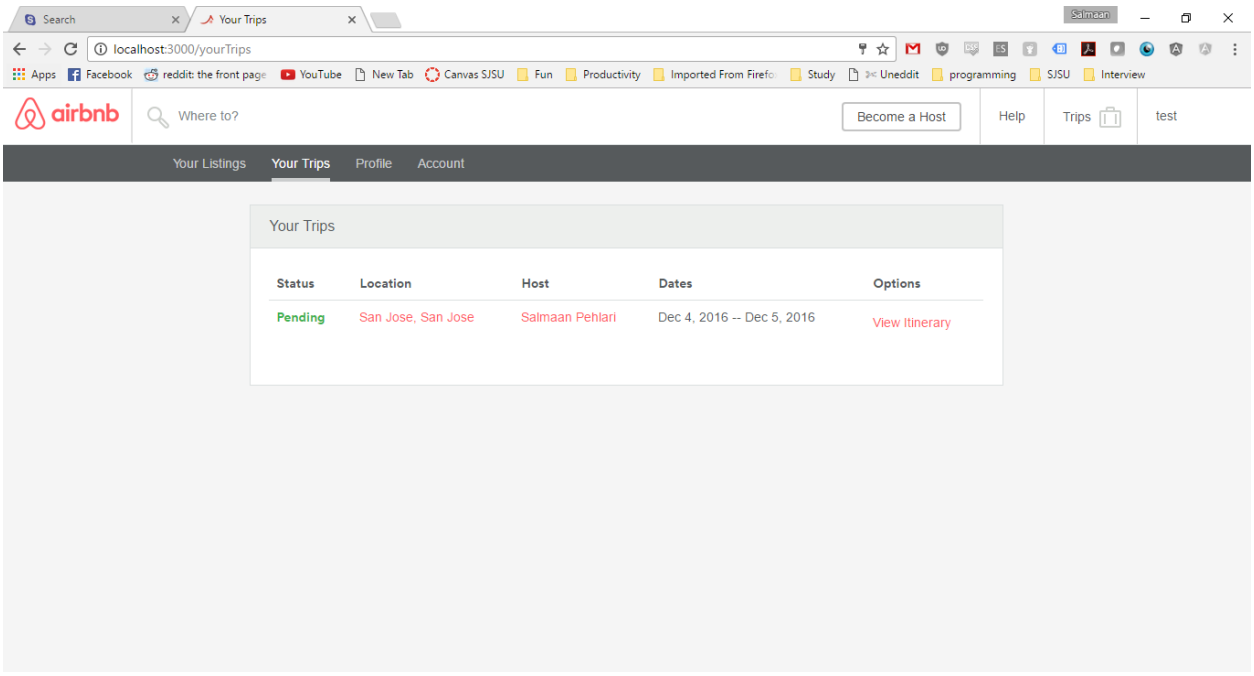
You can Edit the profile in the following screen



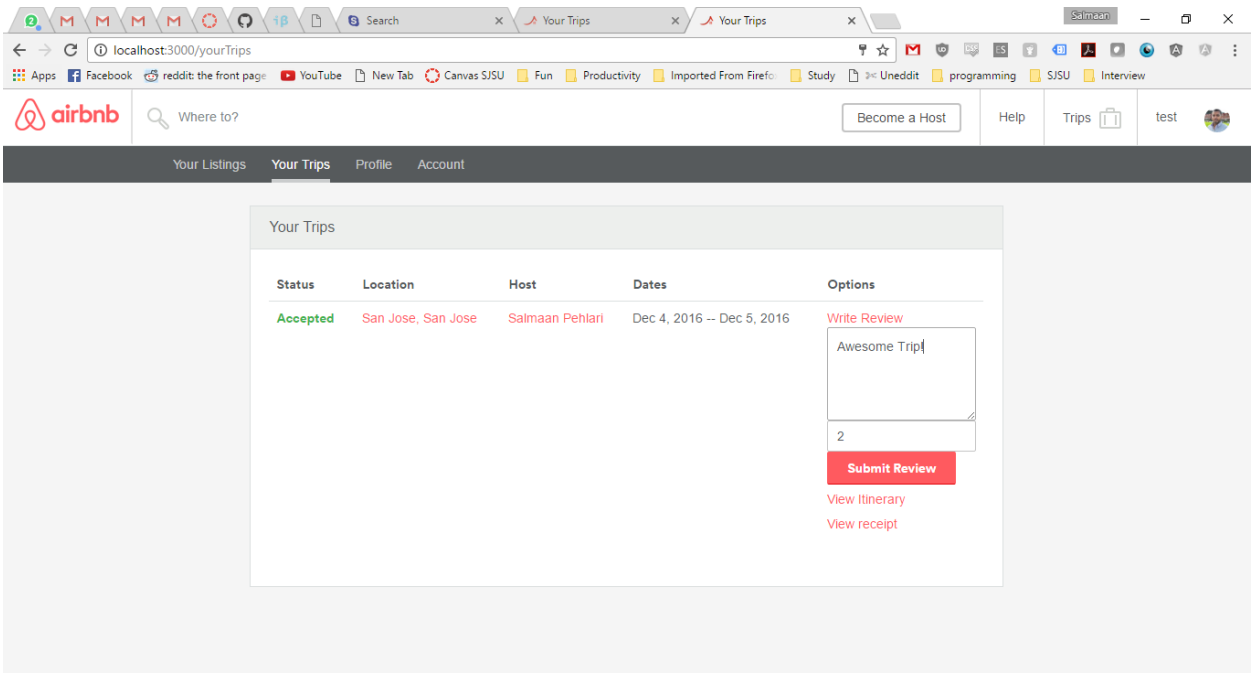
User can upload photos in edit profile page



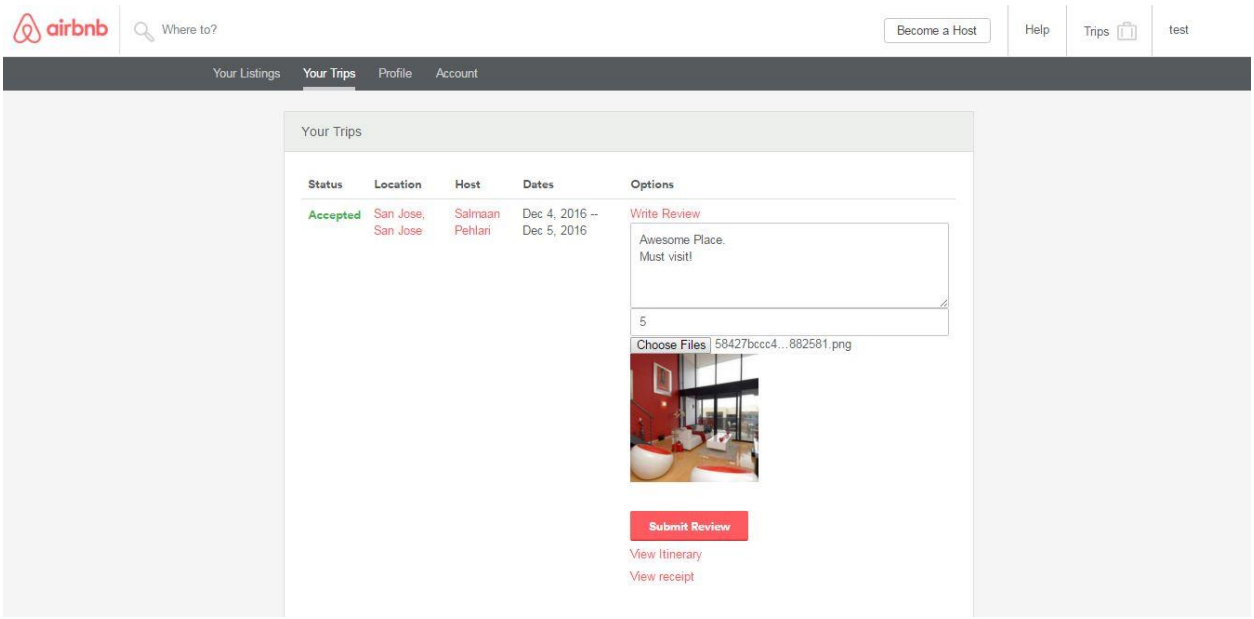
Your Trips Page



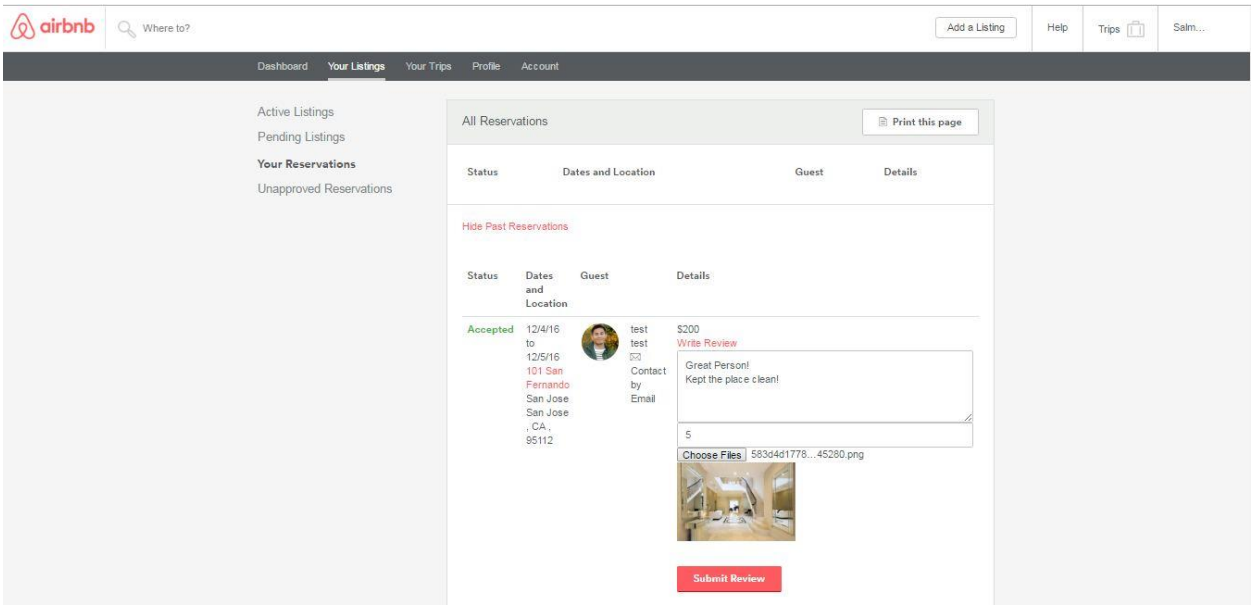
Reviews and ratings can be added after the trip in accepted

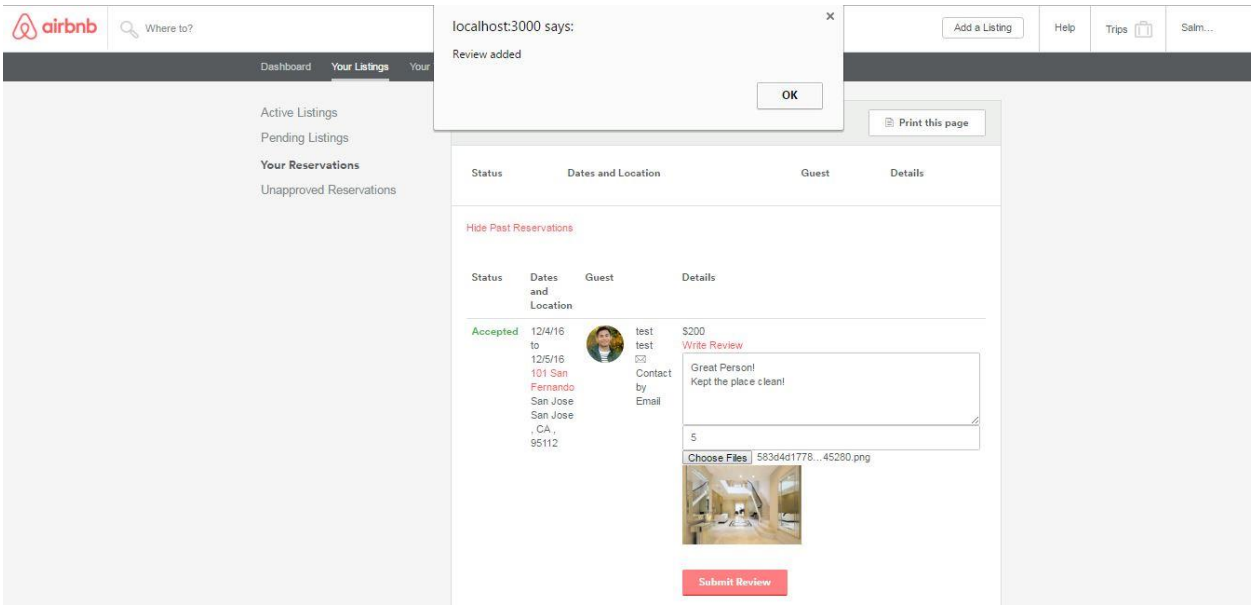


Add Host Review

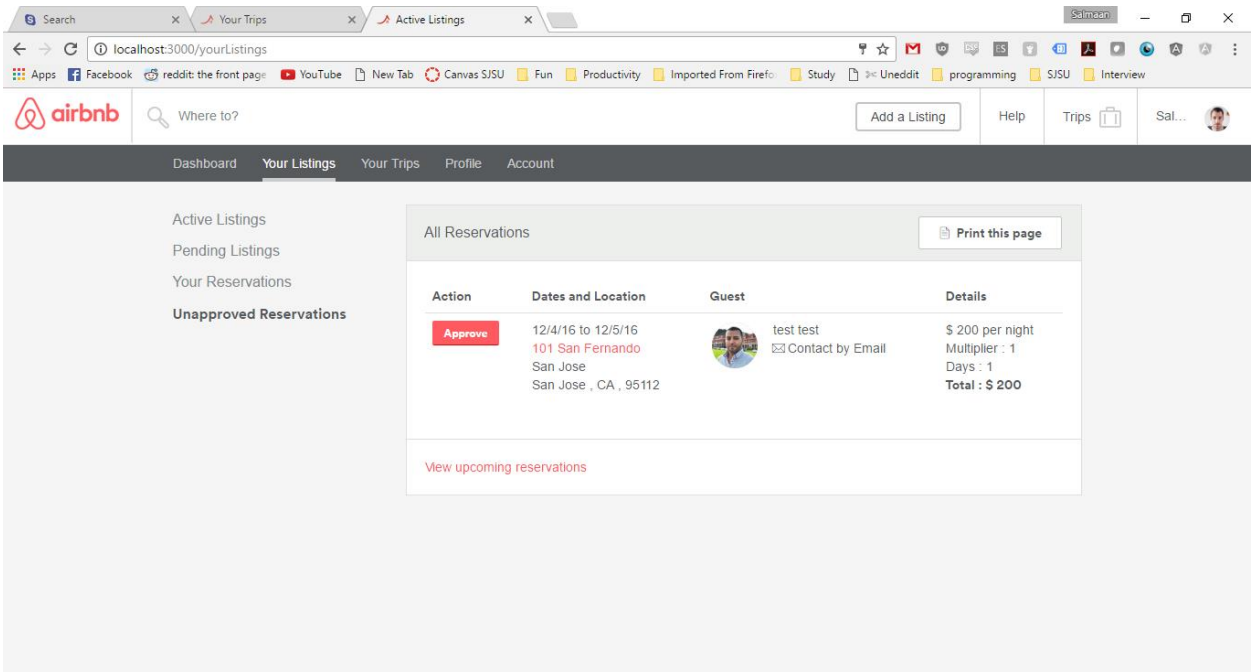


Add user review

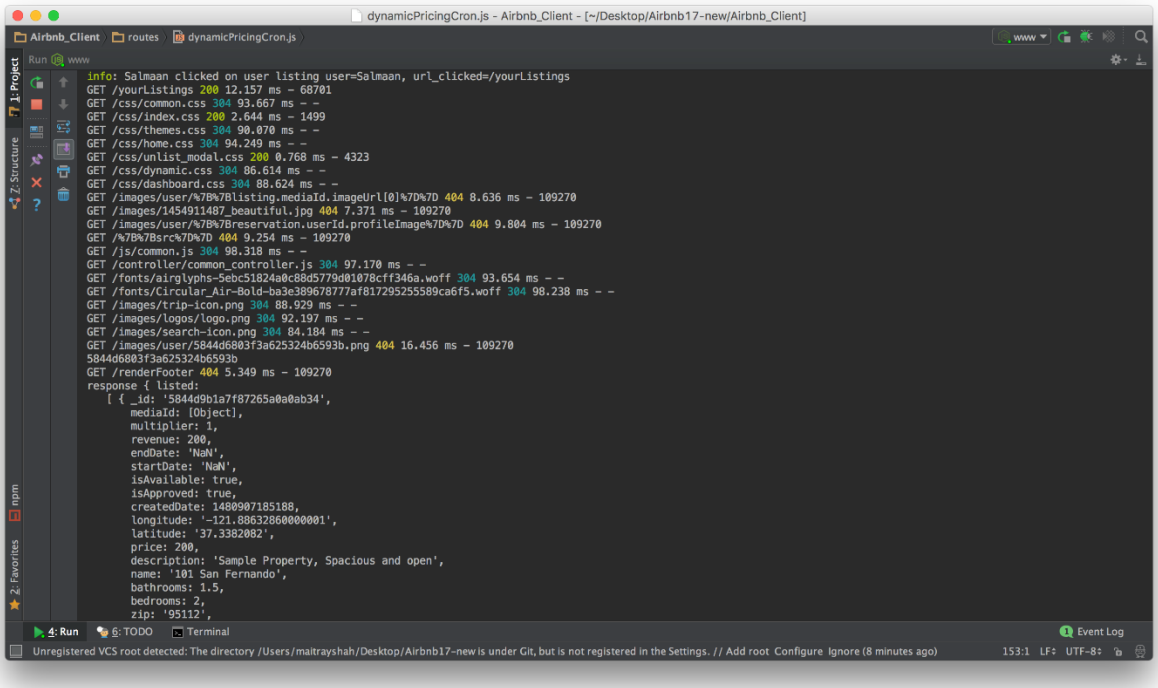


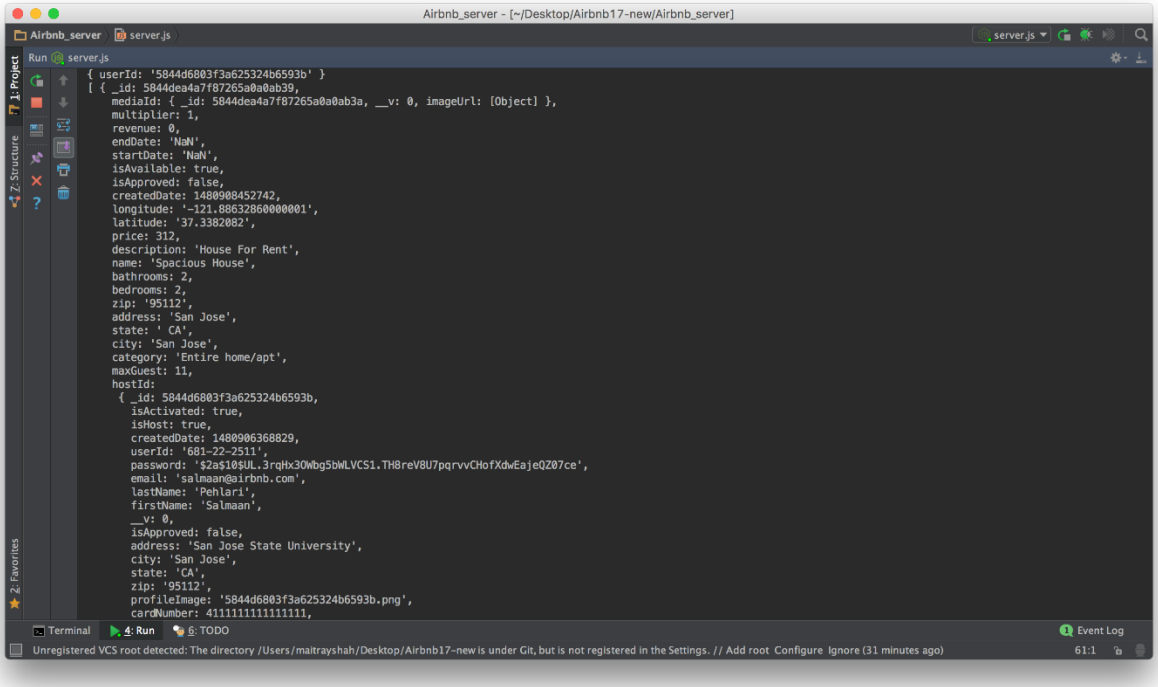


Your Listings Page

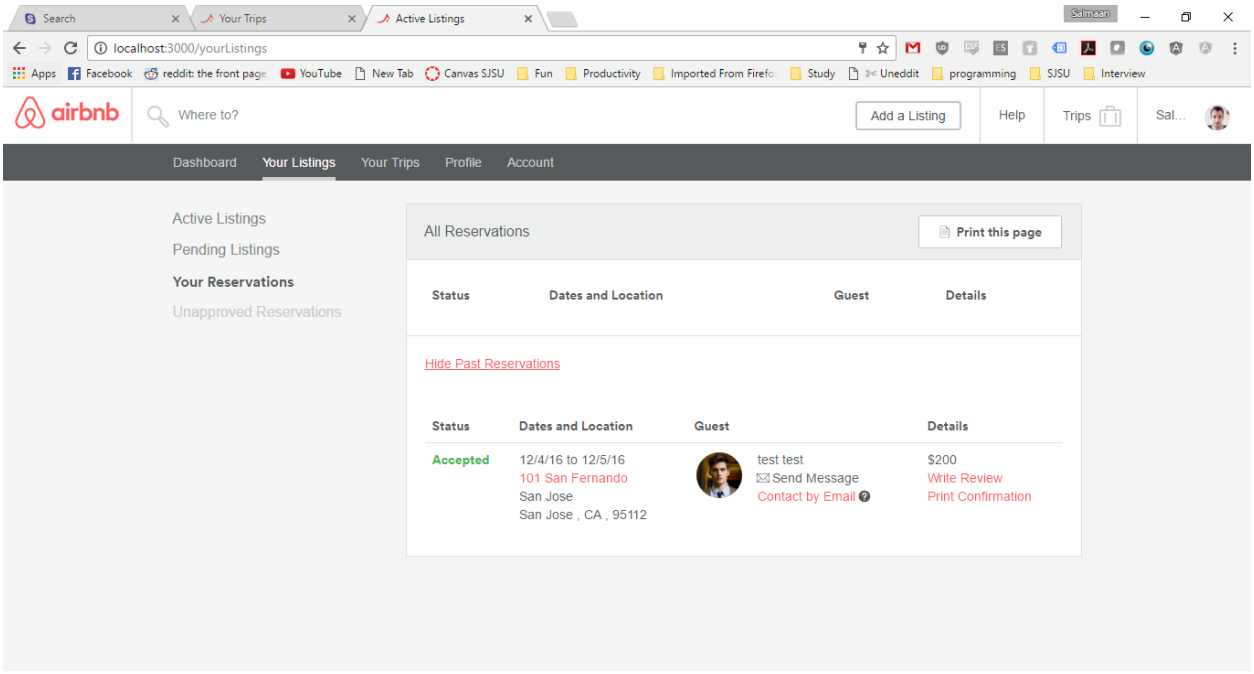


SERVER AND CLIENT CONSOLE CALL FOR USER LISTING



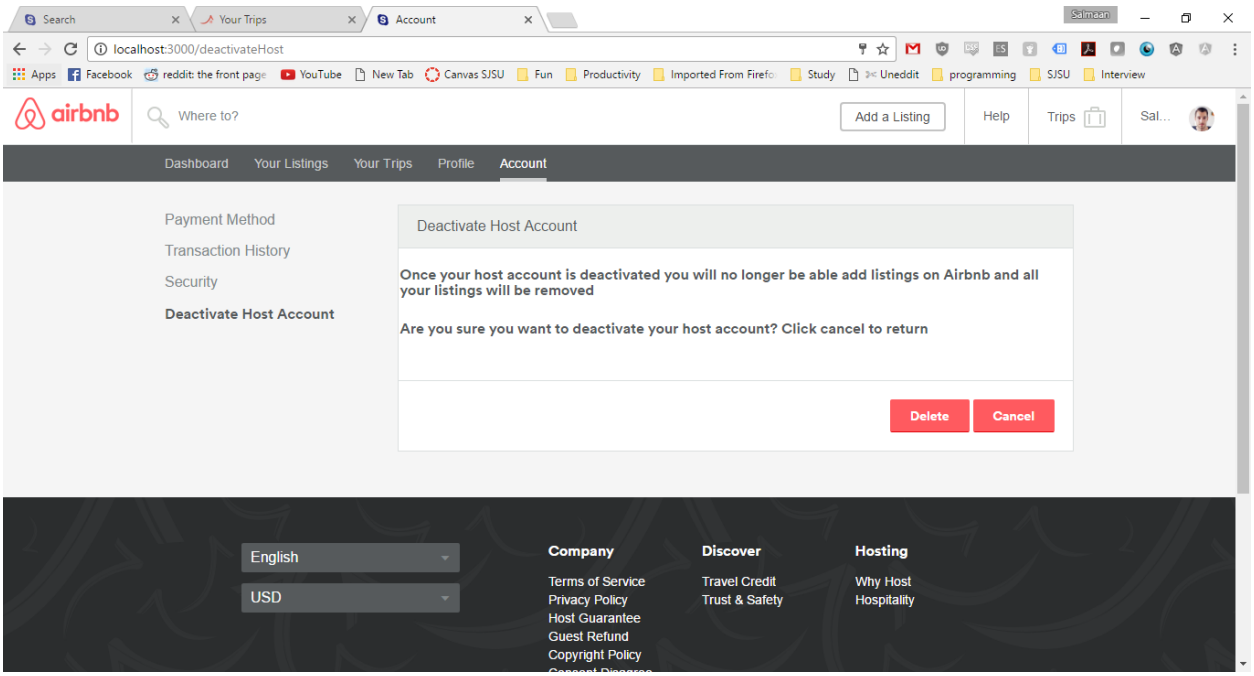


Your past reservations page

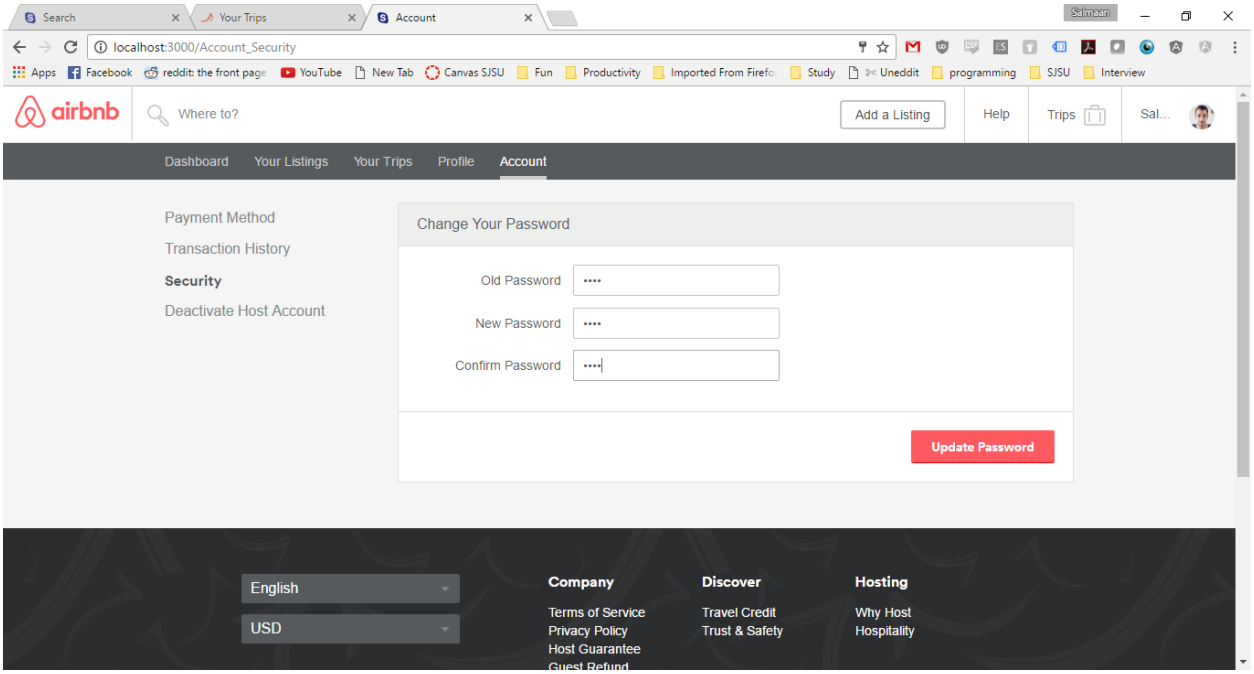


Delete Host Account Page

You can delete your host account

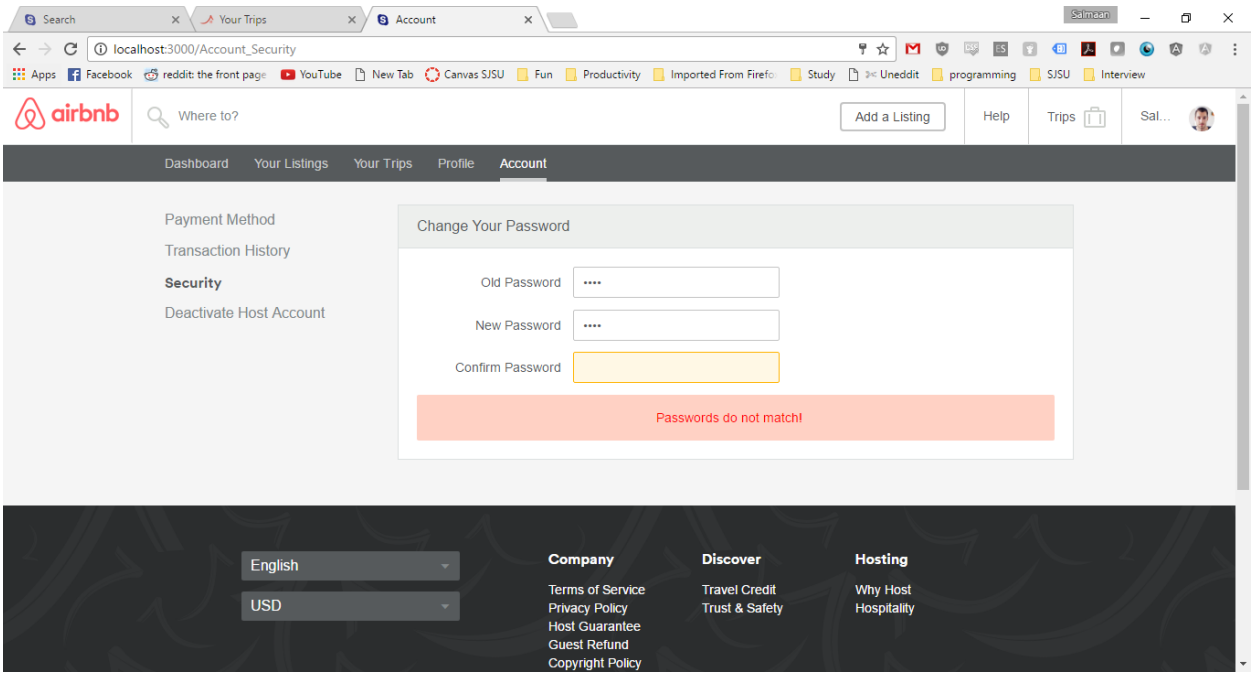


You can change your password in the security panel of the account tab as shown In the following screen



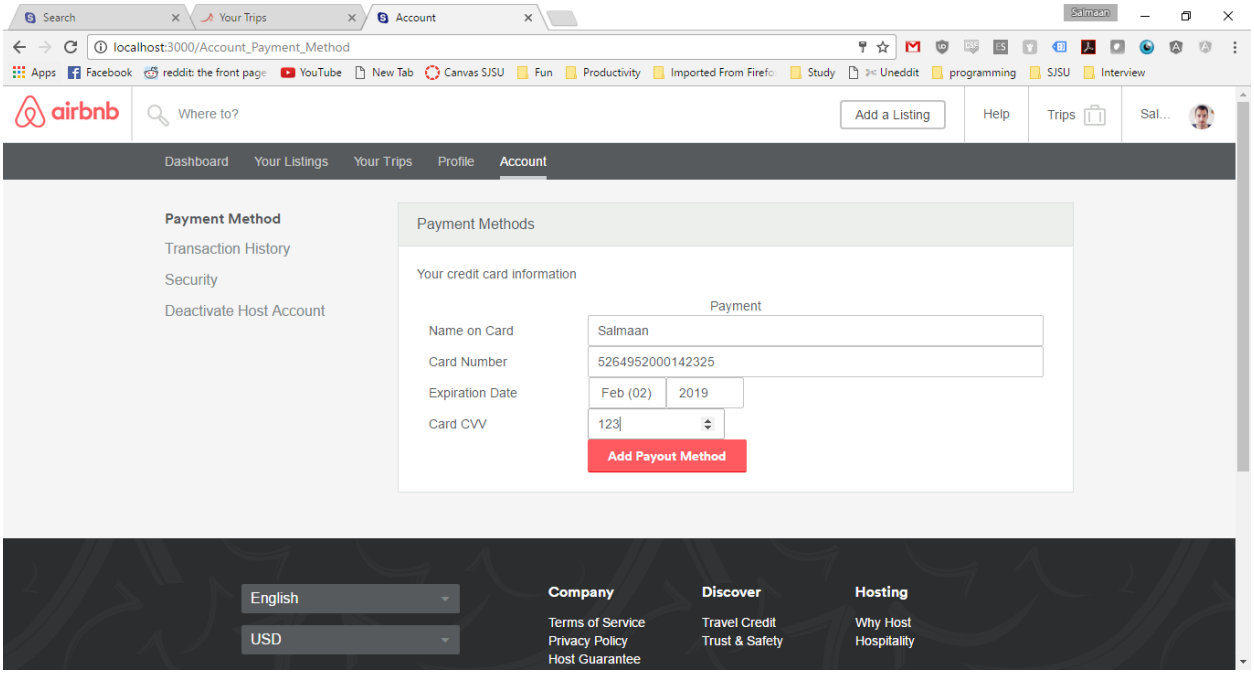
Validations in change password screen

If newly entered password and confirm passwords do not match then the following screen will be appeared

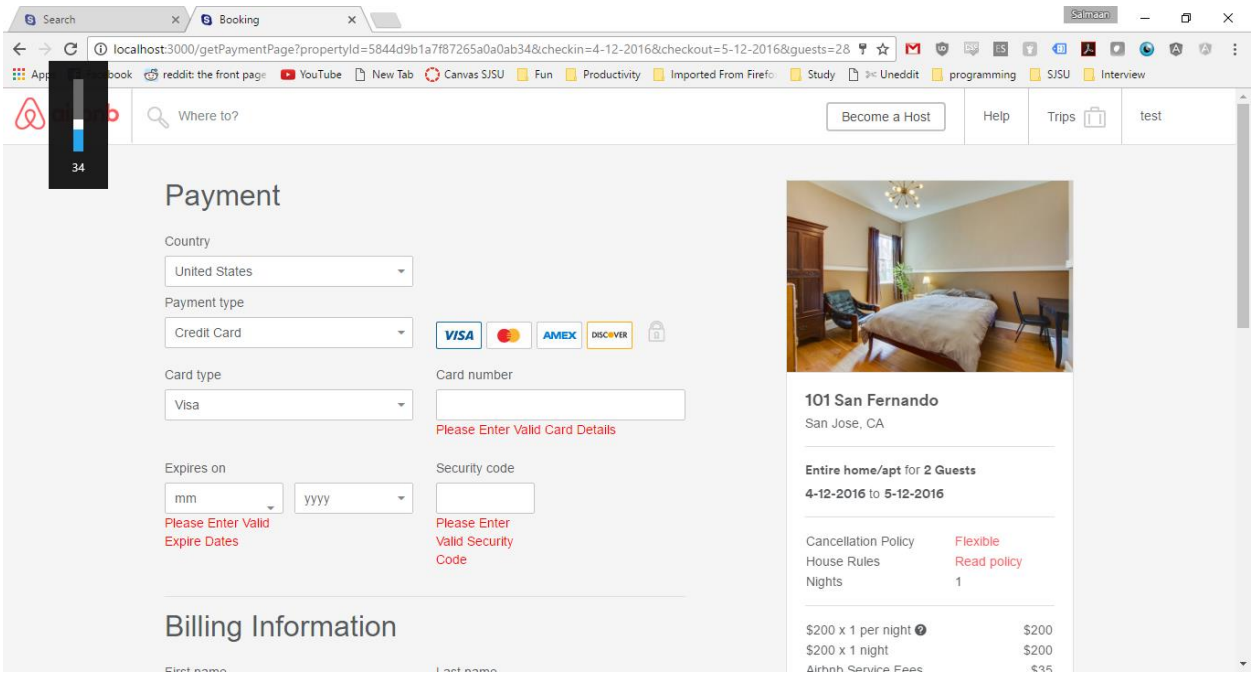


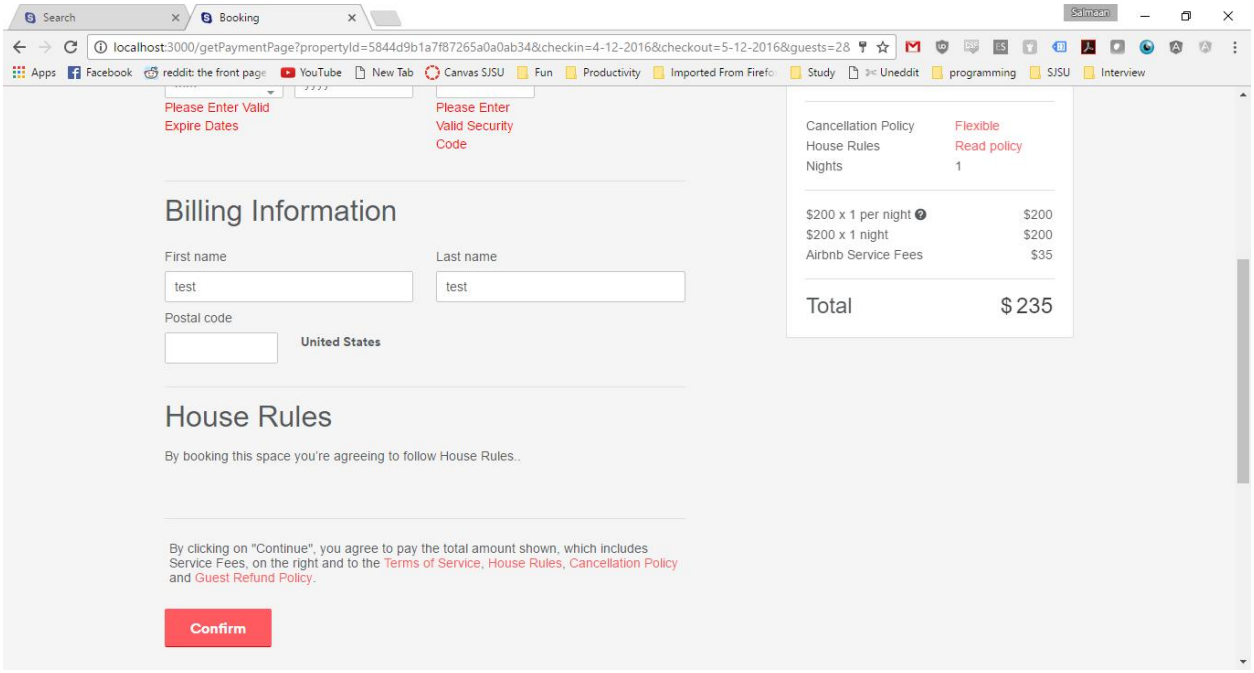
Add payment Method page

User can enter Credit card details in the following screen.

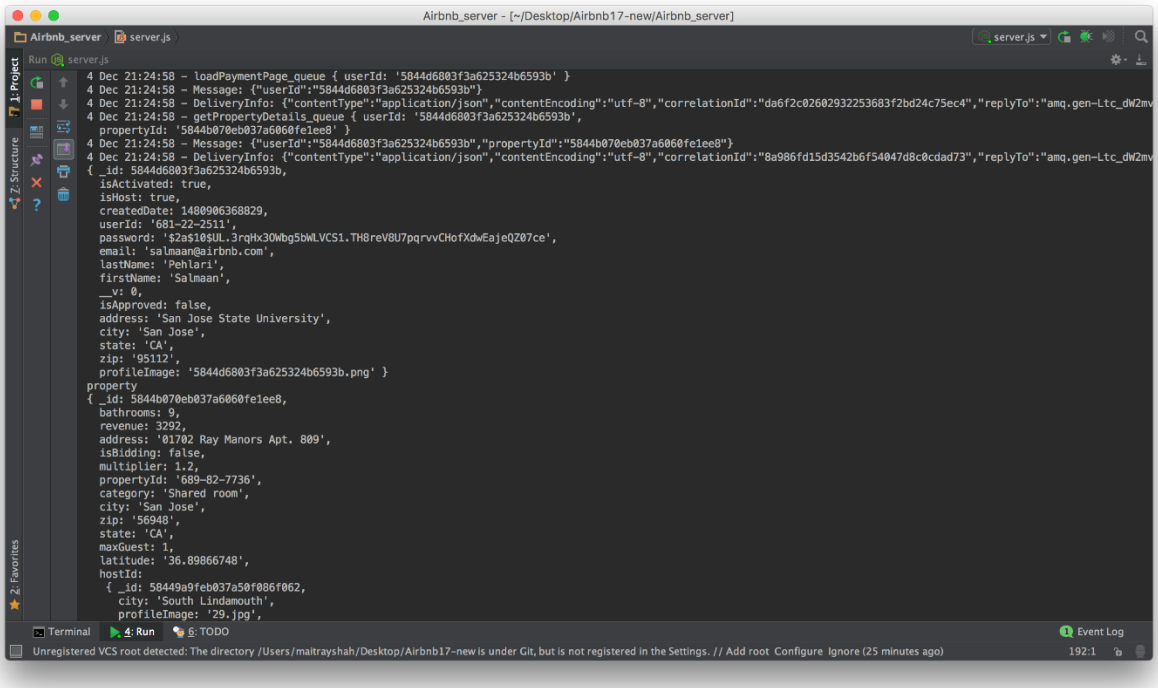


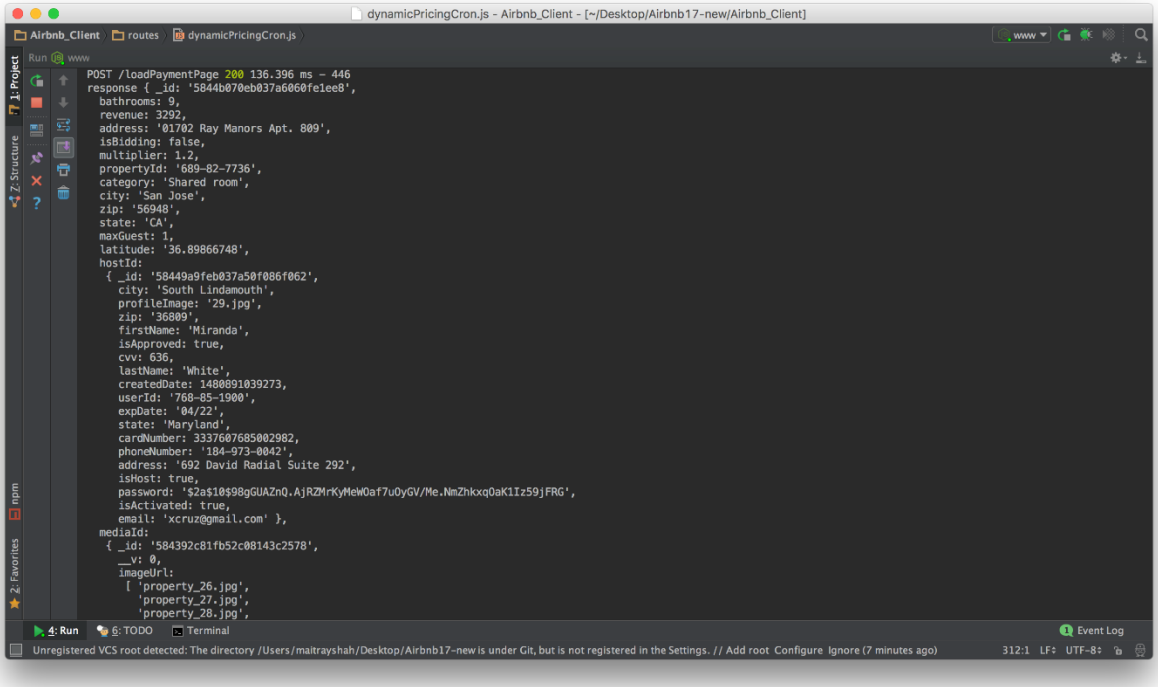
Payment Validation page



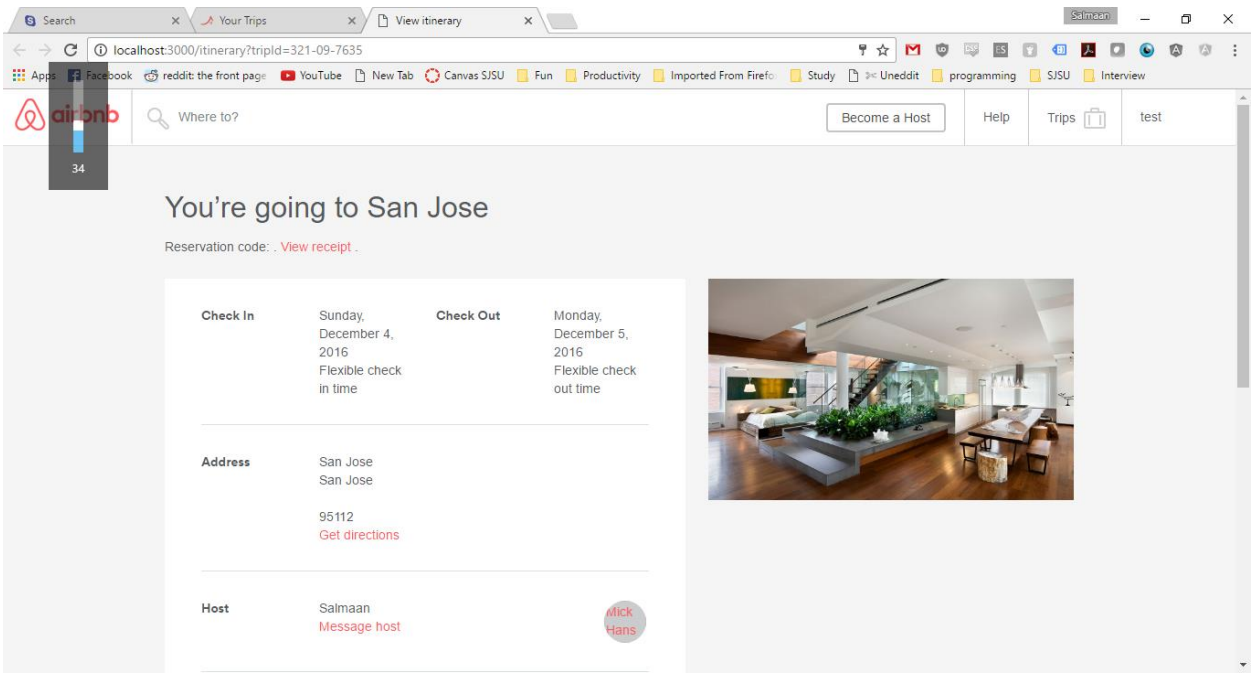

Search
Booking
Balmeet
localhost:3000/getPaymentPage?propertyId=5844d9b1a7f87265a0a0ab34&checkin=4-12-2016&checkout=5-12-2016&guests=28
Apps
Facebook
reddit: the front page
YouTube
New Tab
Canvas SJSU
Fun
Productivity
Imported From Firefo
Study
Uneddit
programming
SJSU
Interview
Please Enter Valid Expire Dates
Please Enter Valid Security Code
Billing Information
First name
test
Last name
test
Postal code
United States
House Rules
By booking this space you're agreeing to follow House Rules..
By clicking on "Continue", you agree to pay the total amount shown, which includes Service Fees, on the right and to the Terms of Service, House Rules, Cancellation Policy and Guest Refund Policy.
Confirm
Cancellation Policy
Flexible
House Rules
Read policy
Nights
1
$200 x 1 per night
$200
$200 x 1 night
$200
Airbnb Service Fees
$35
Total
$ 235

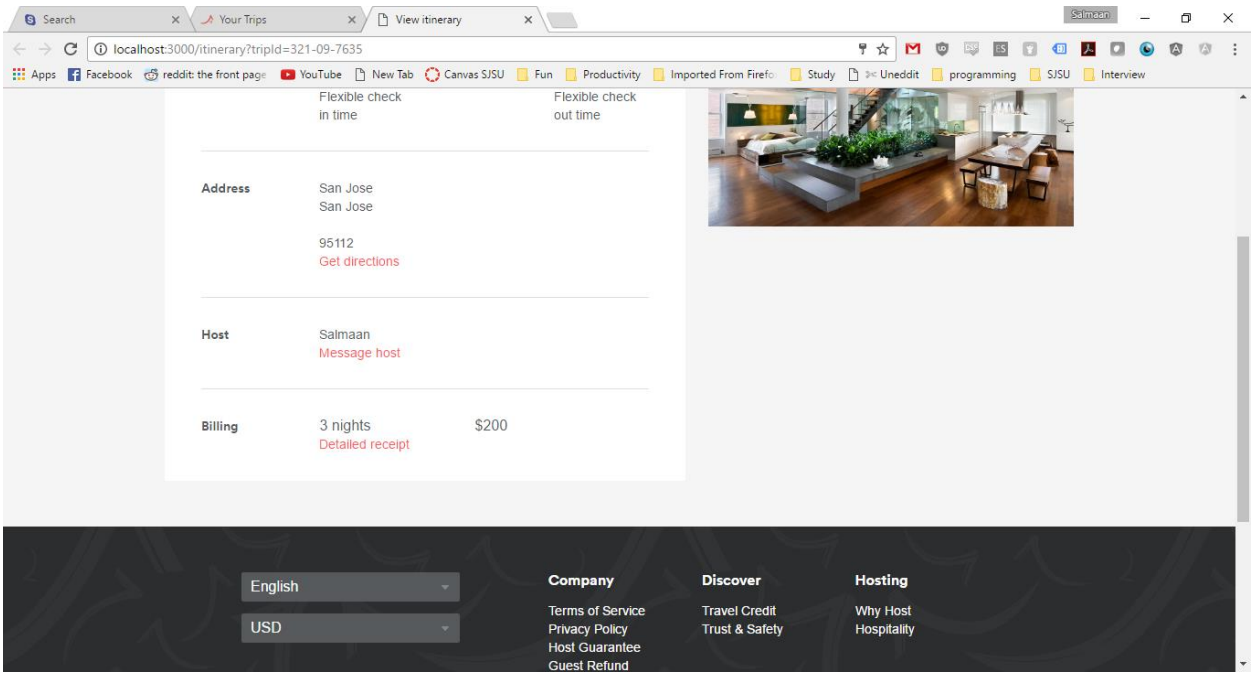
CLIENT AND SERVER CONSOLE – PAYMENT PAGE



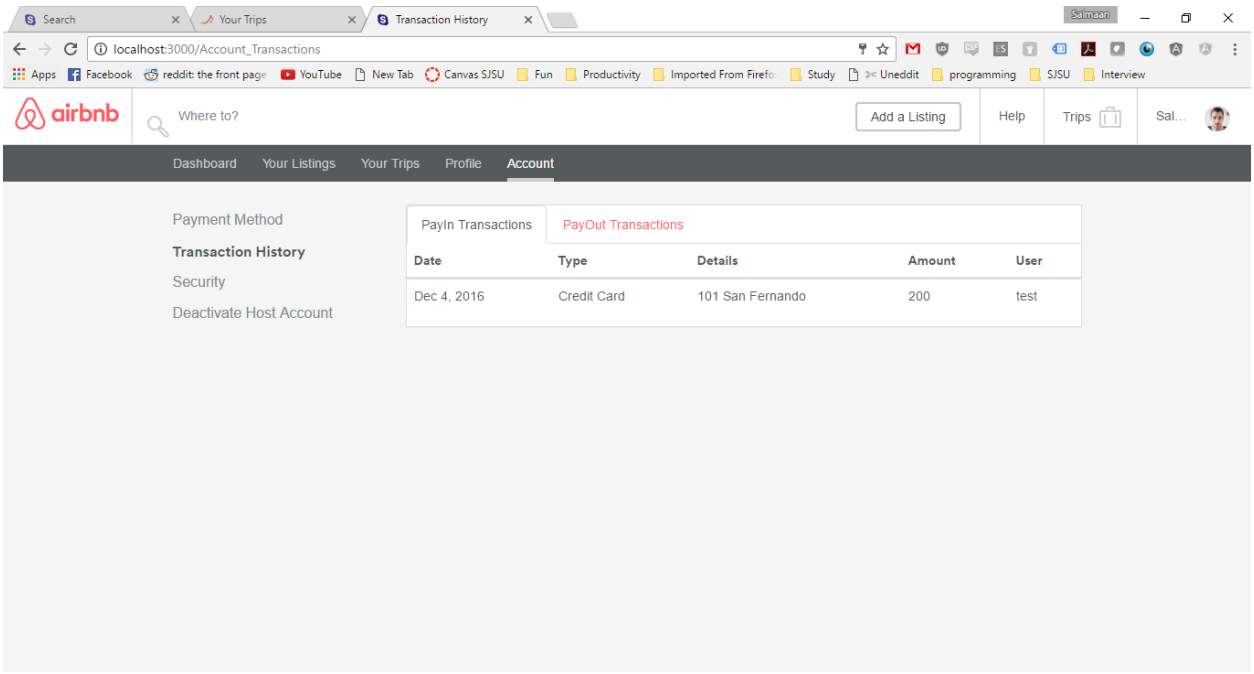


View Itinerary page

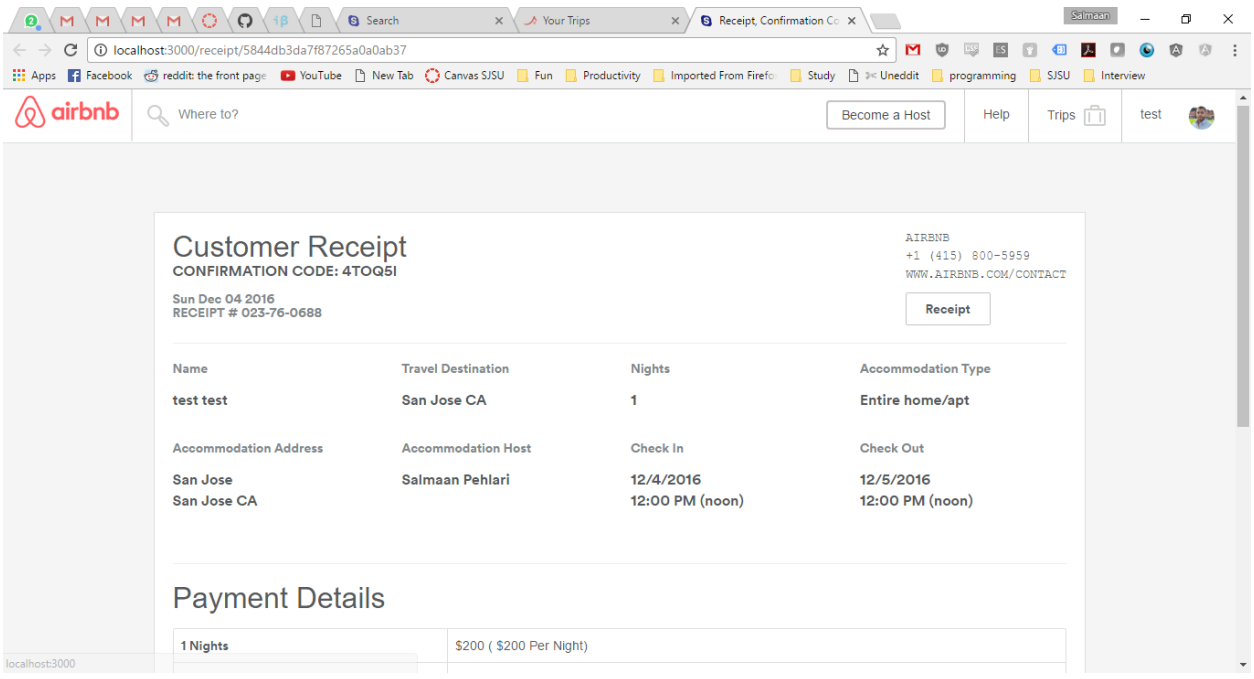


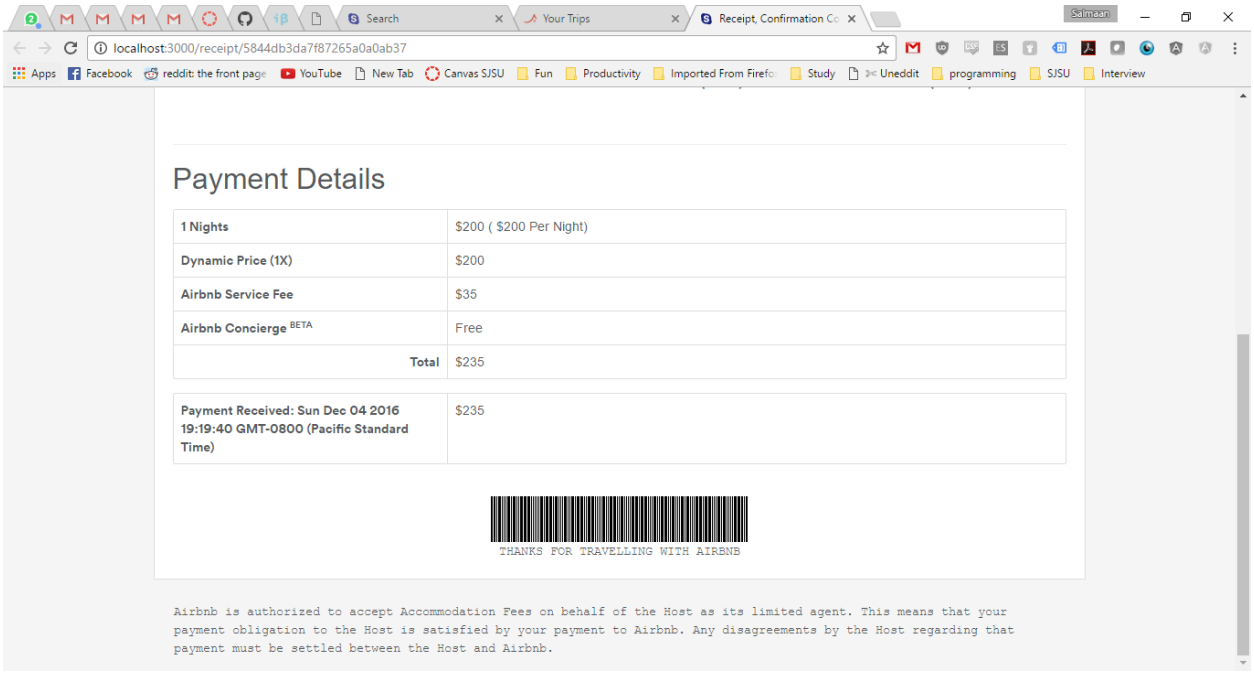


Transaction History Page

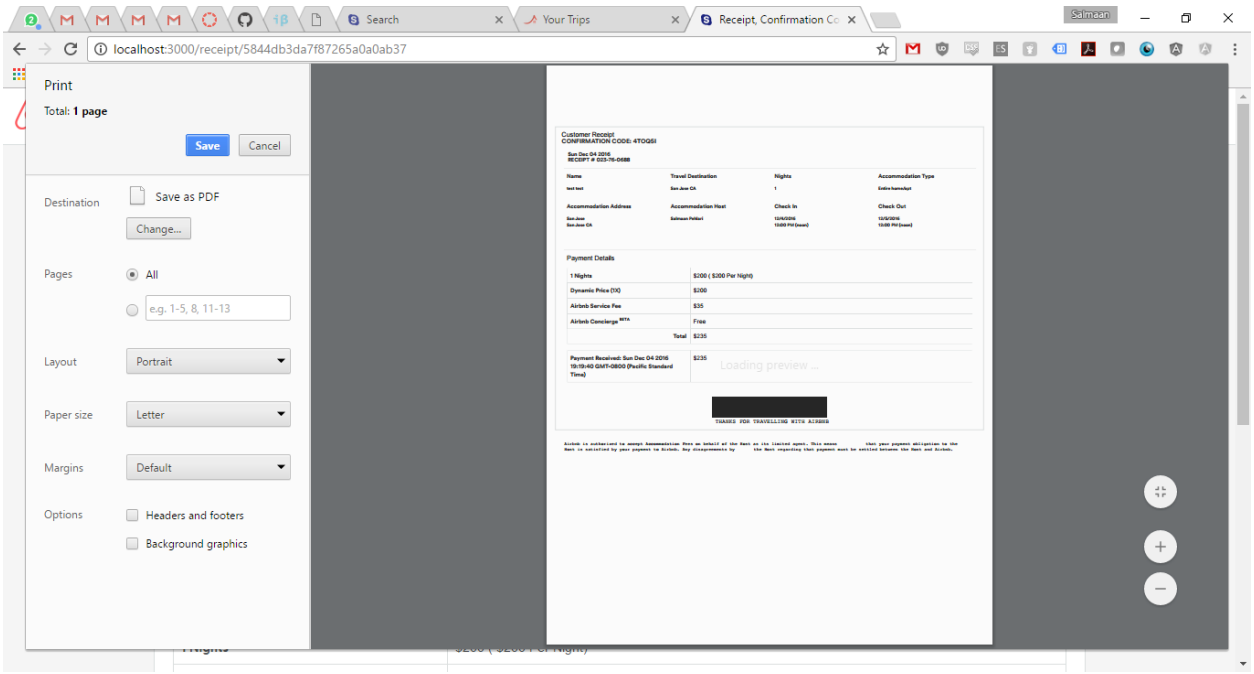


Billing Receipt



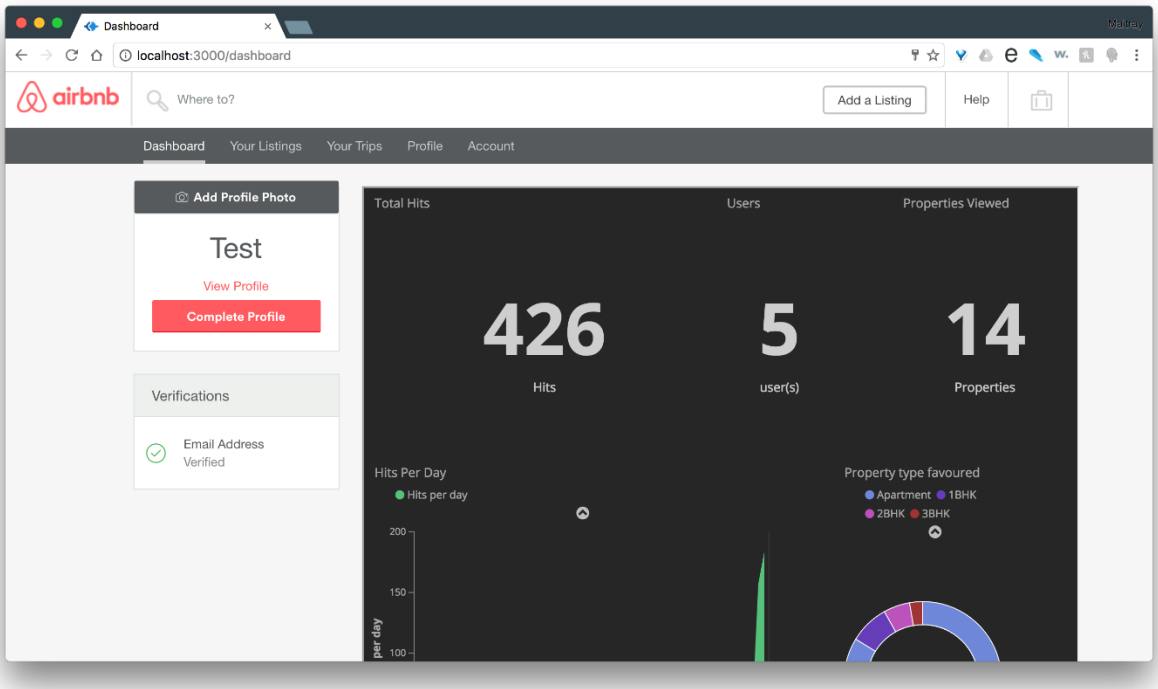


You can save the receipt as pdf format

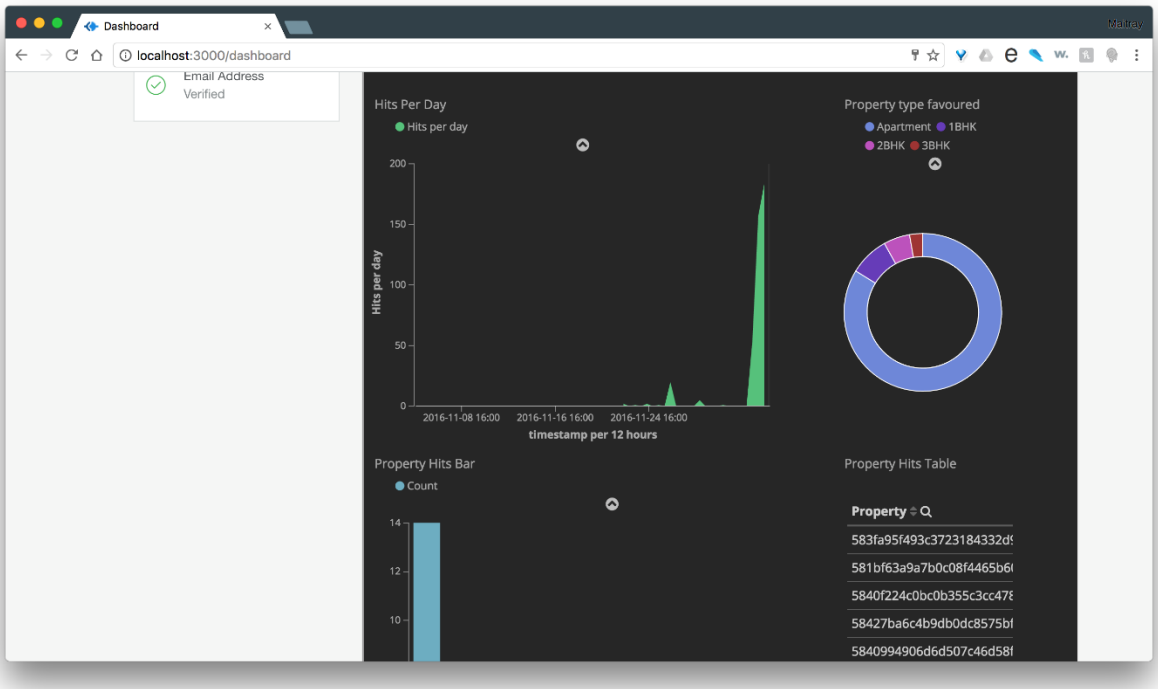


Analytics Module

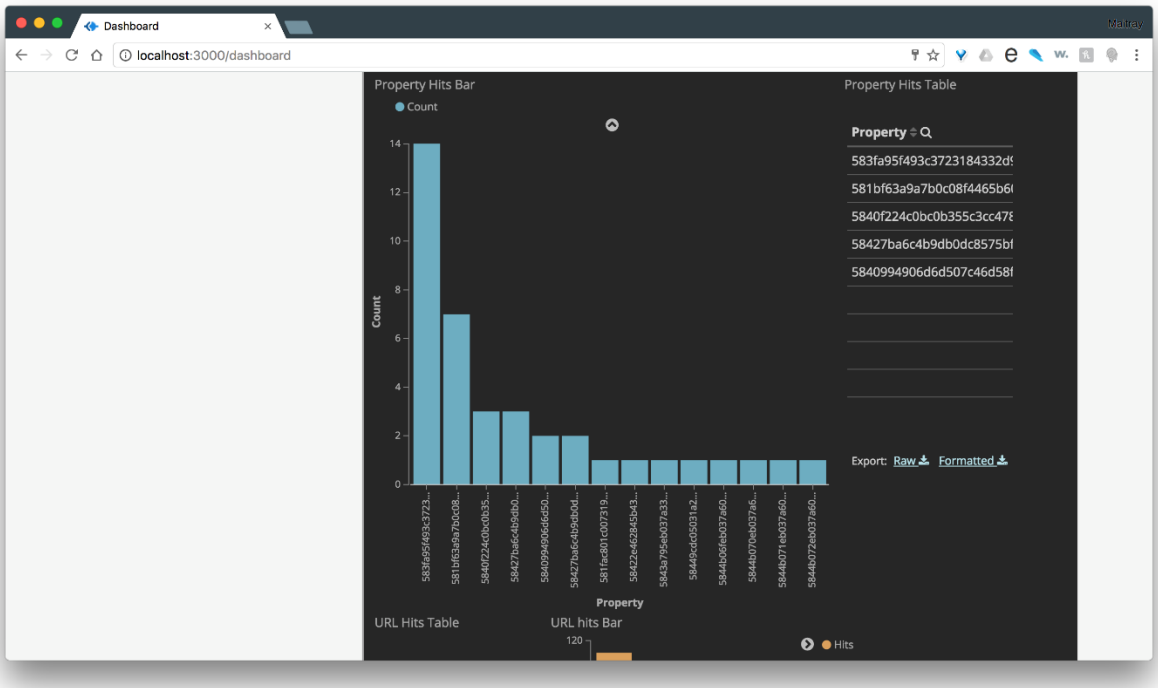
Dashboard tab



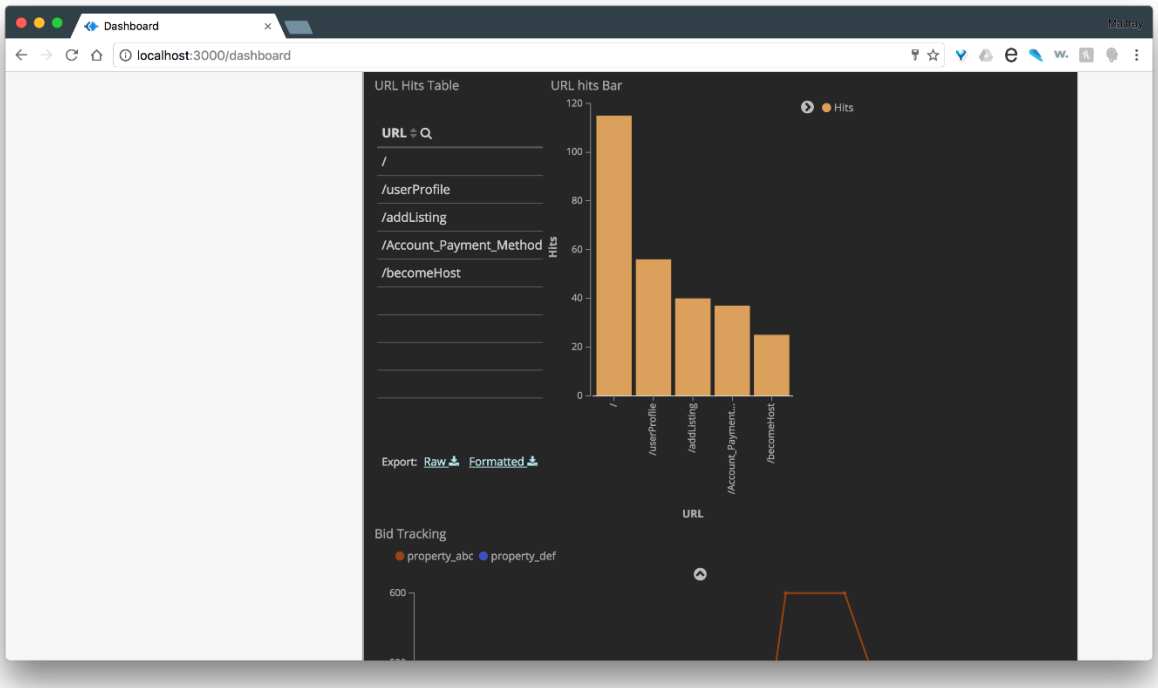
Property Hits and Hits per page will be shown in the following page



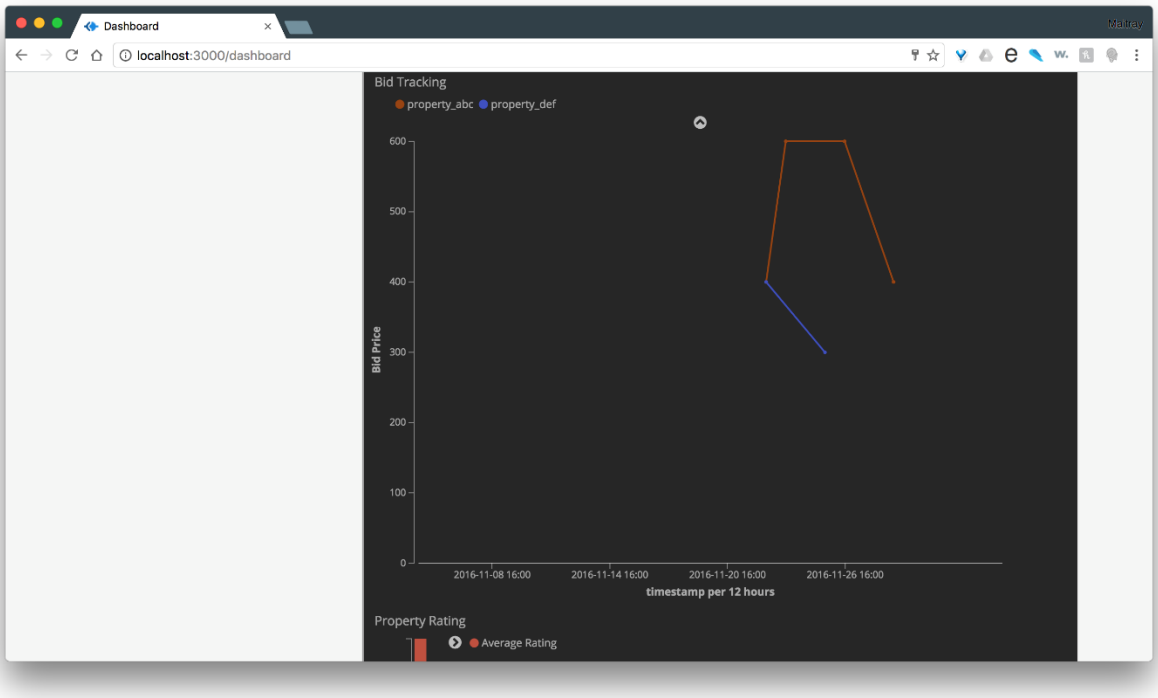
Property Hits bar will be shown in the following screen



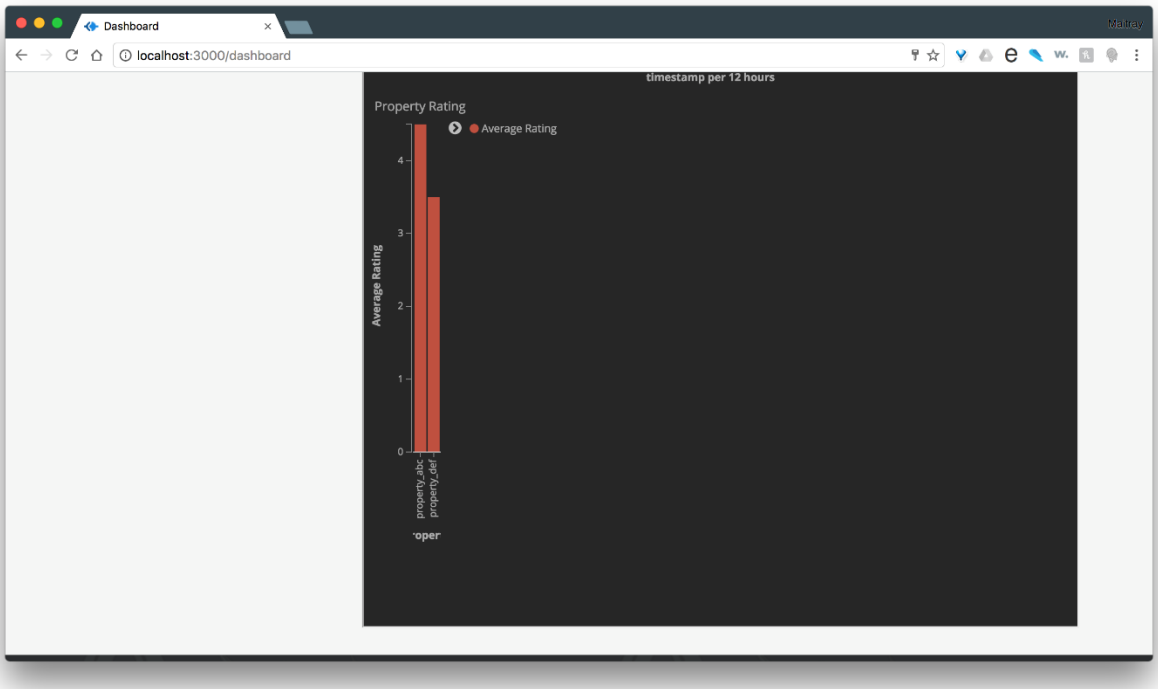
URL hits bar will appear in the following screen



Bid Tracking will appear in the following page



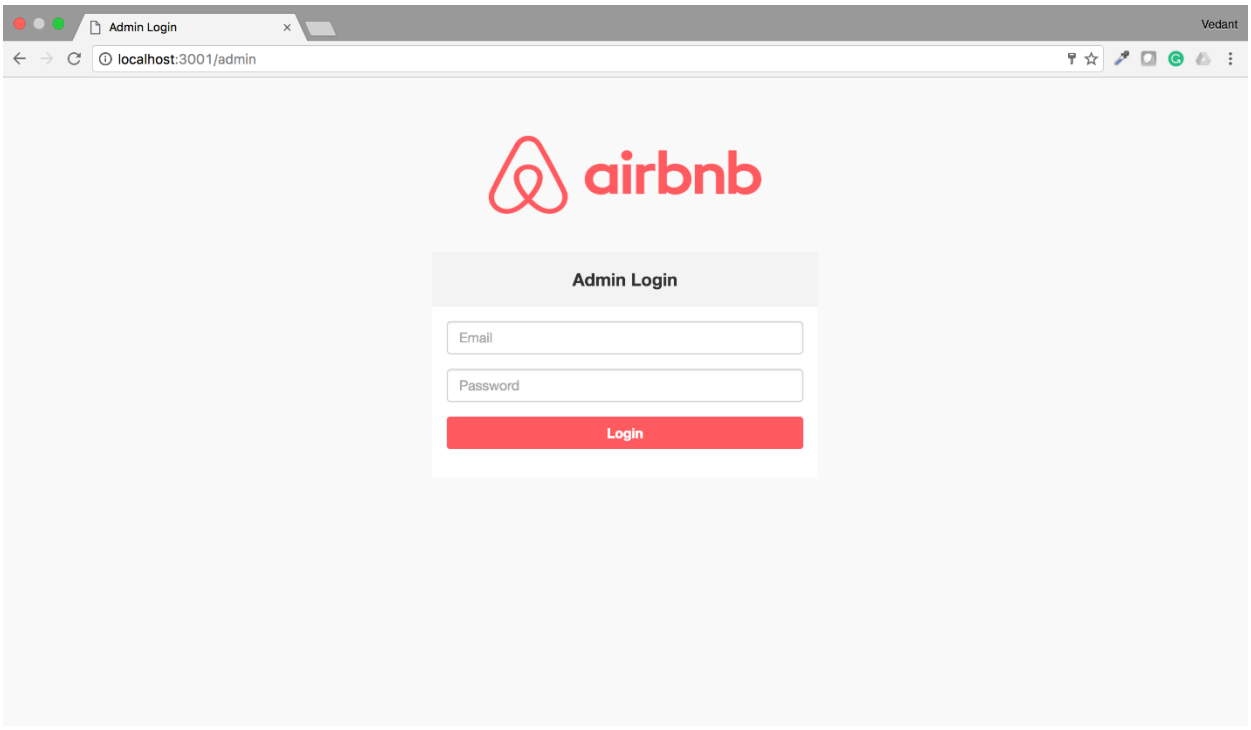
Property Rating will appear in the following page



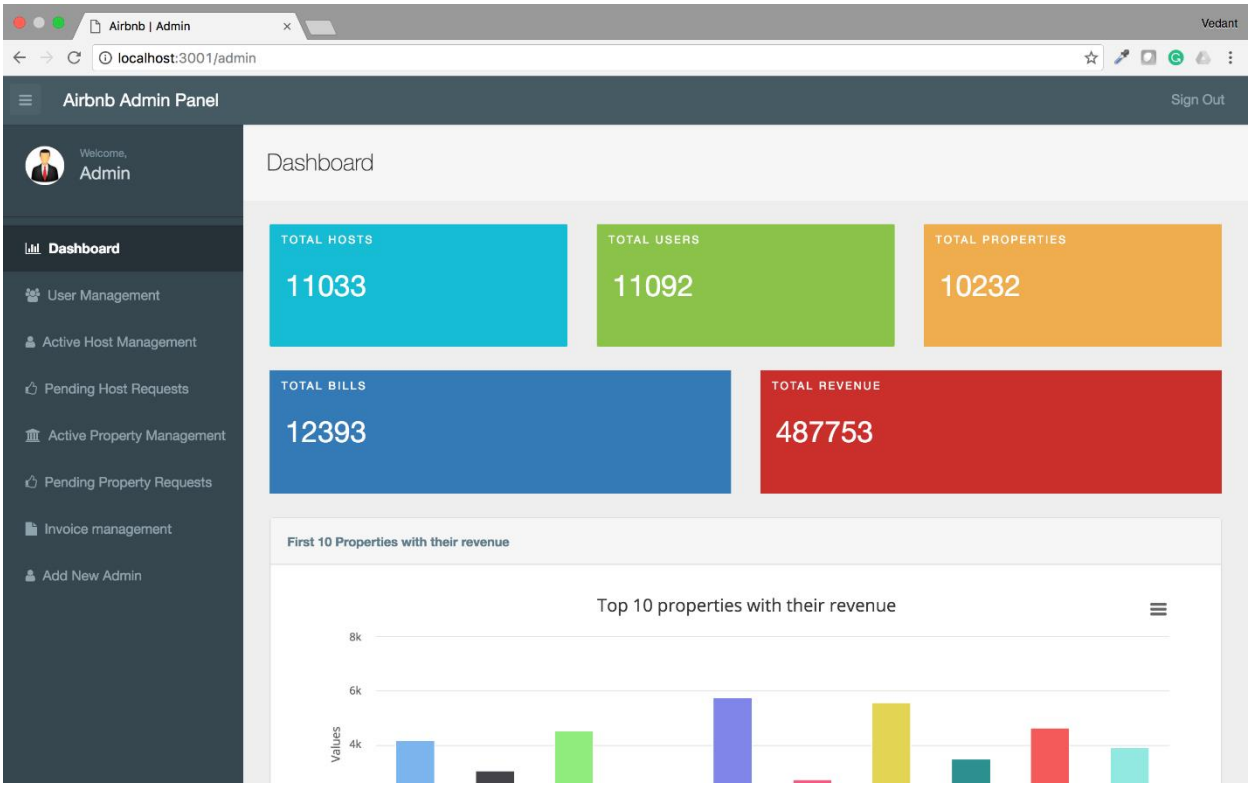
Admin portal features

- View total users, hosts, properties, revenue and number of bills
- View top 10 properties with their revenue
- View 10 cities with their revenue
- View top 10 hosts with their revenue
- View active Users, Hosts and Properties
- View pending Users, Hosts and Properties
- Approved pending Host and Property requests
- View all the bills
- Add new admin to for the system

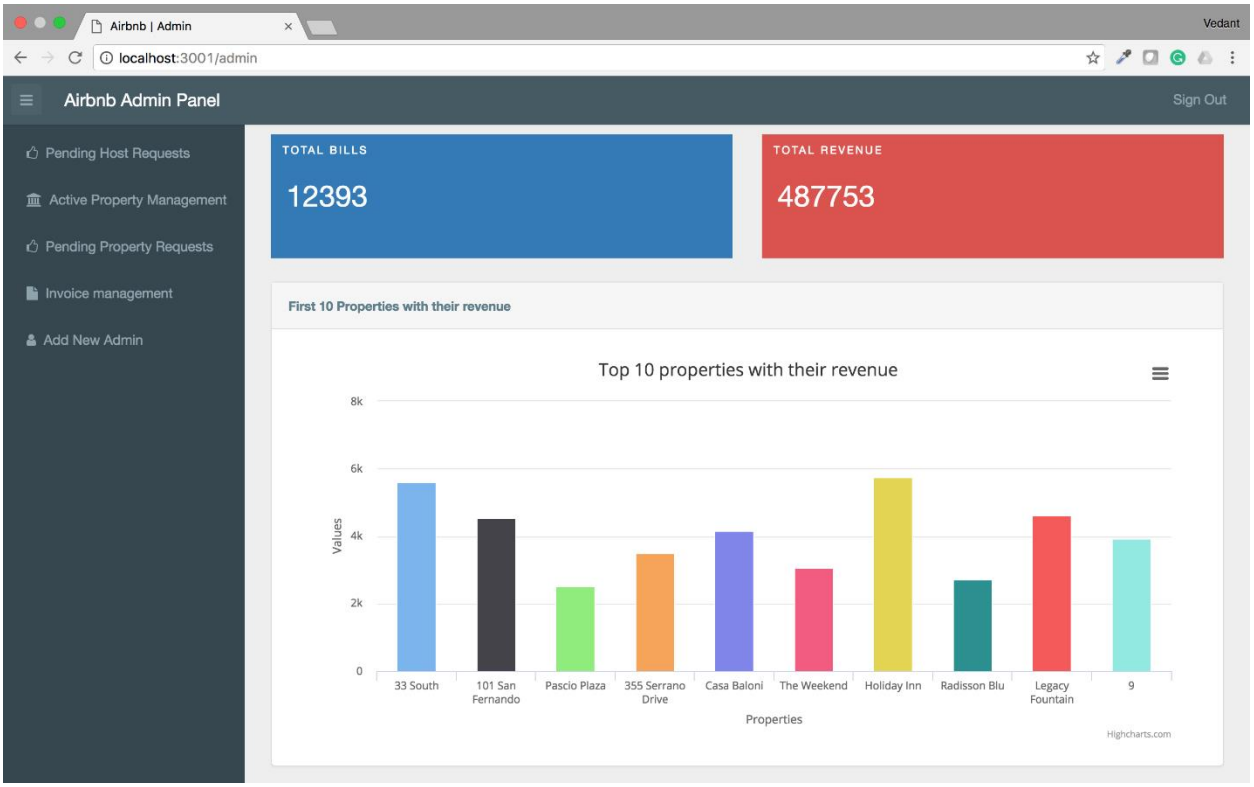
Admin login page



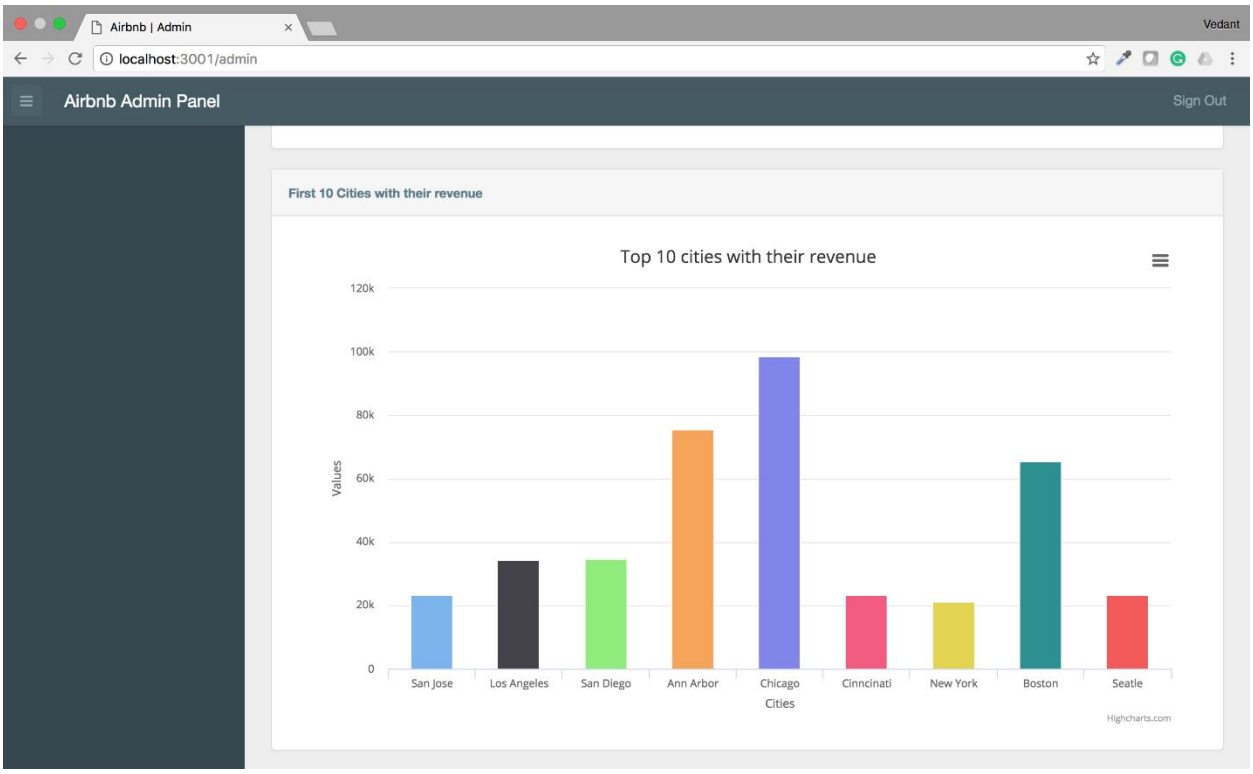
Admin dashboard



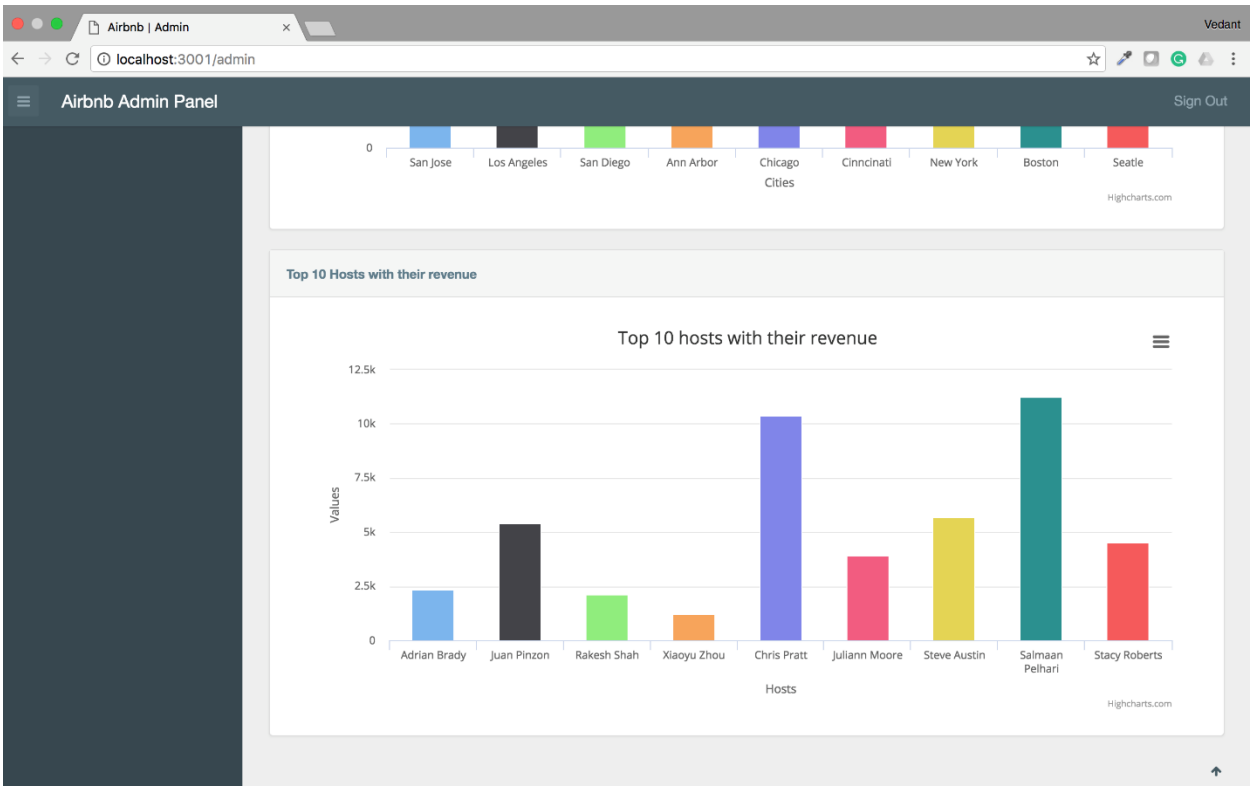
Top 10 properties with their revenue



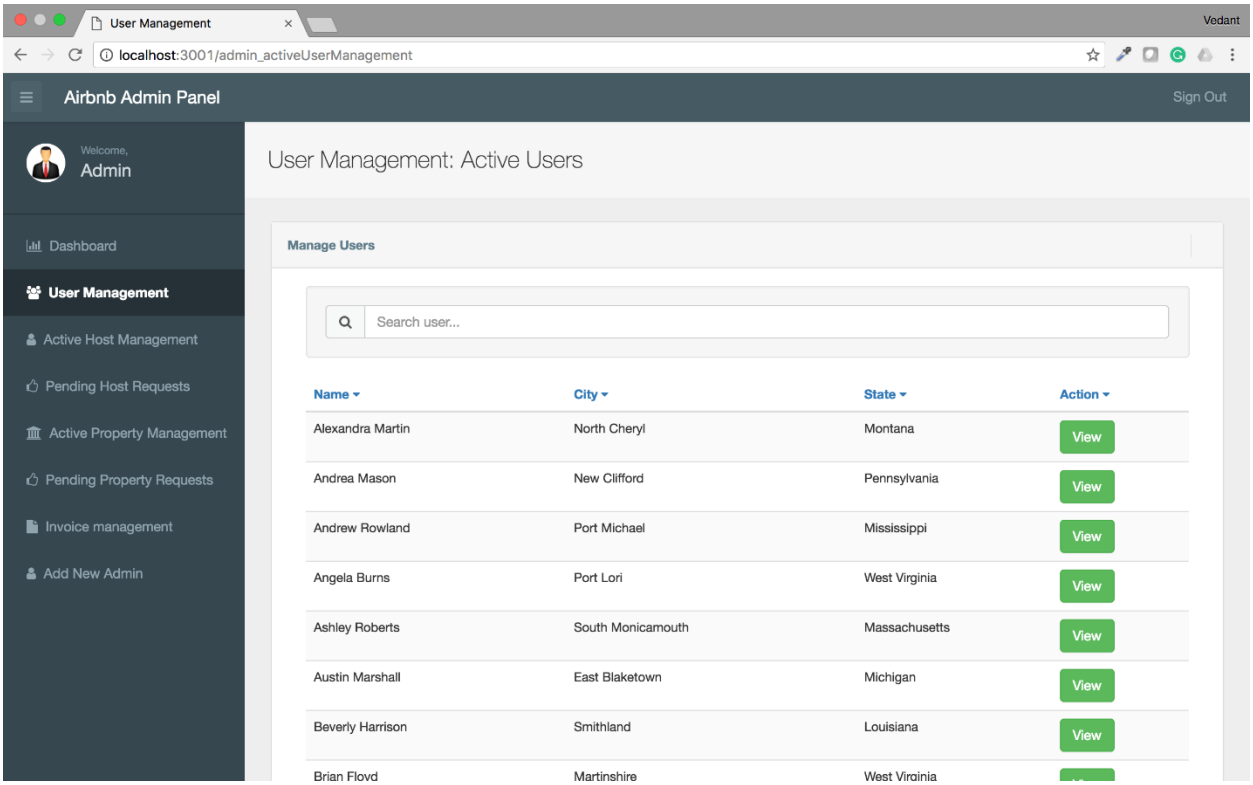
First 10 cities with their revenue



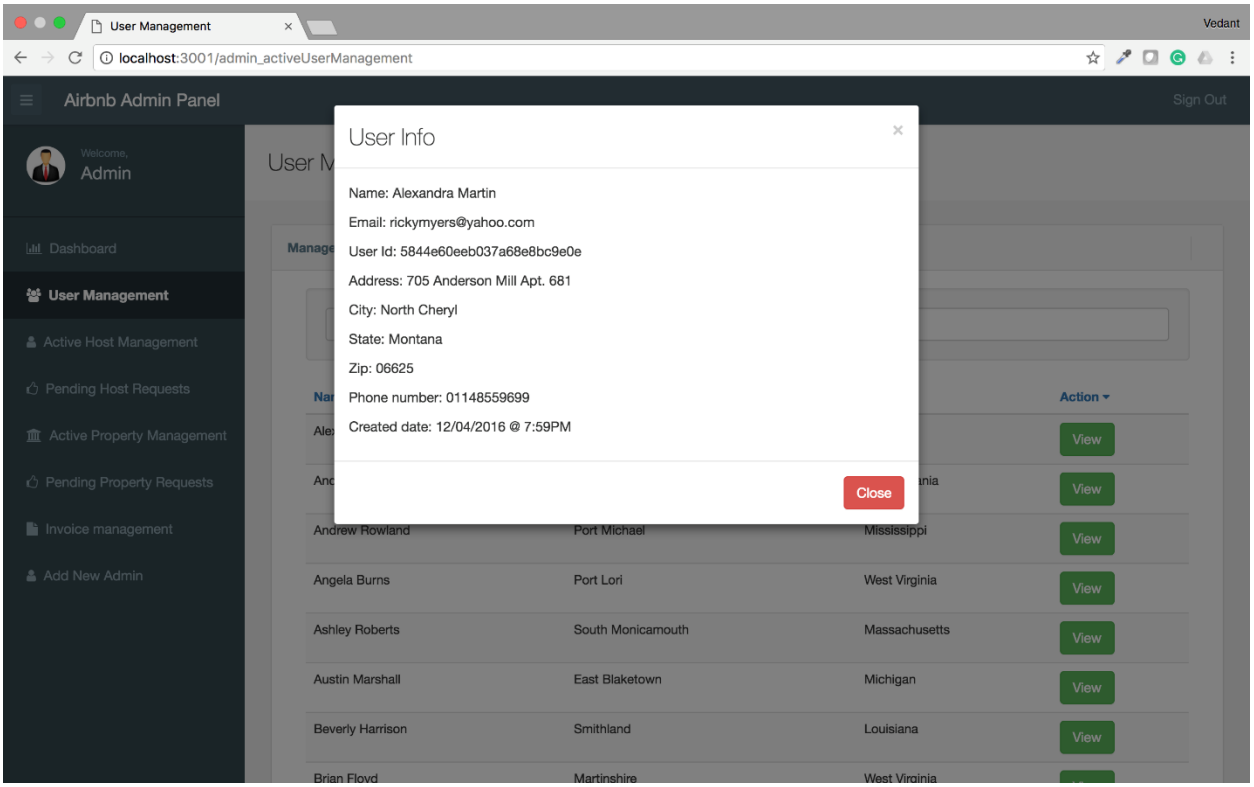
Top 10 hosts with their revenue



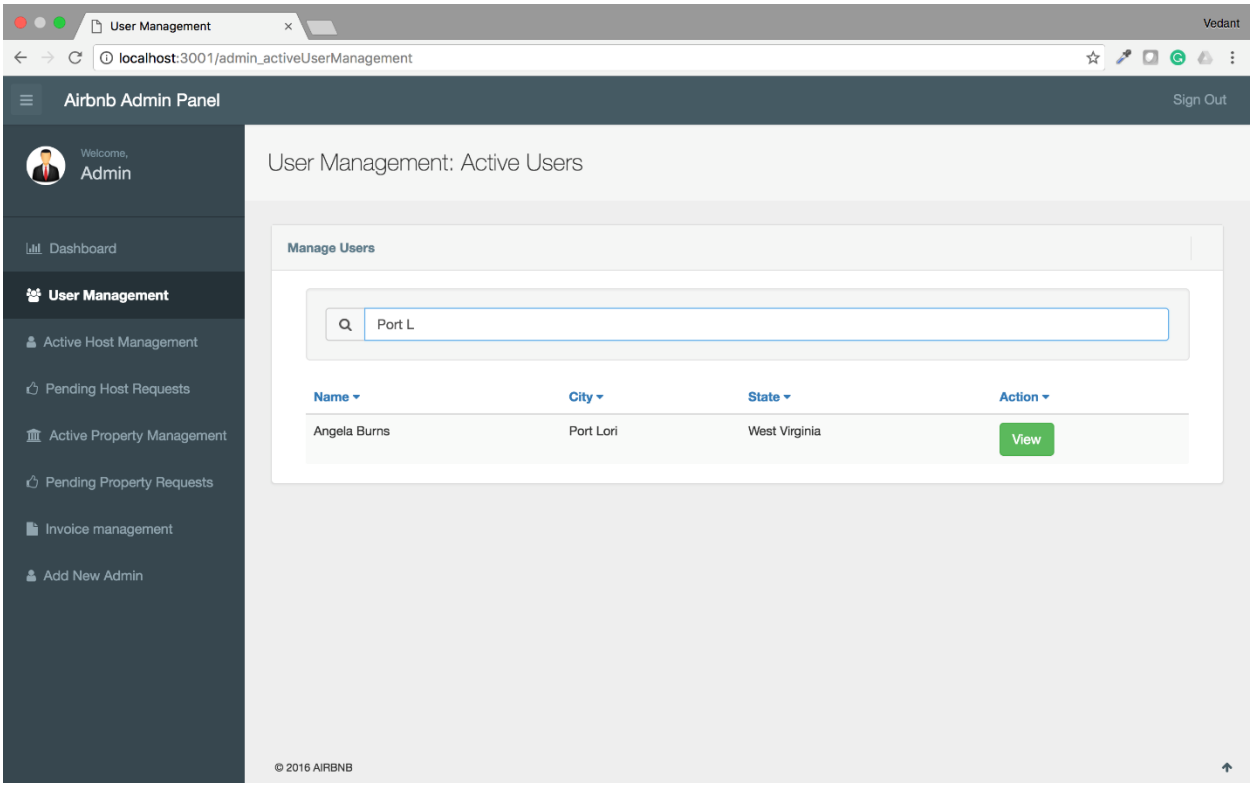
List of all the active users



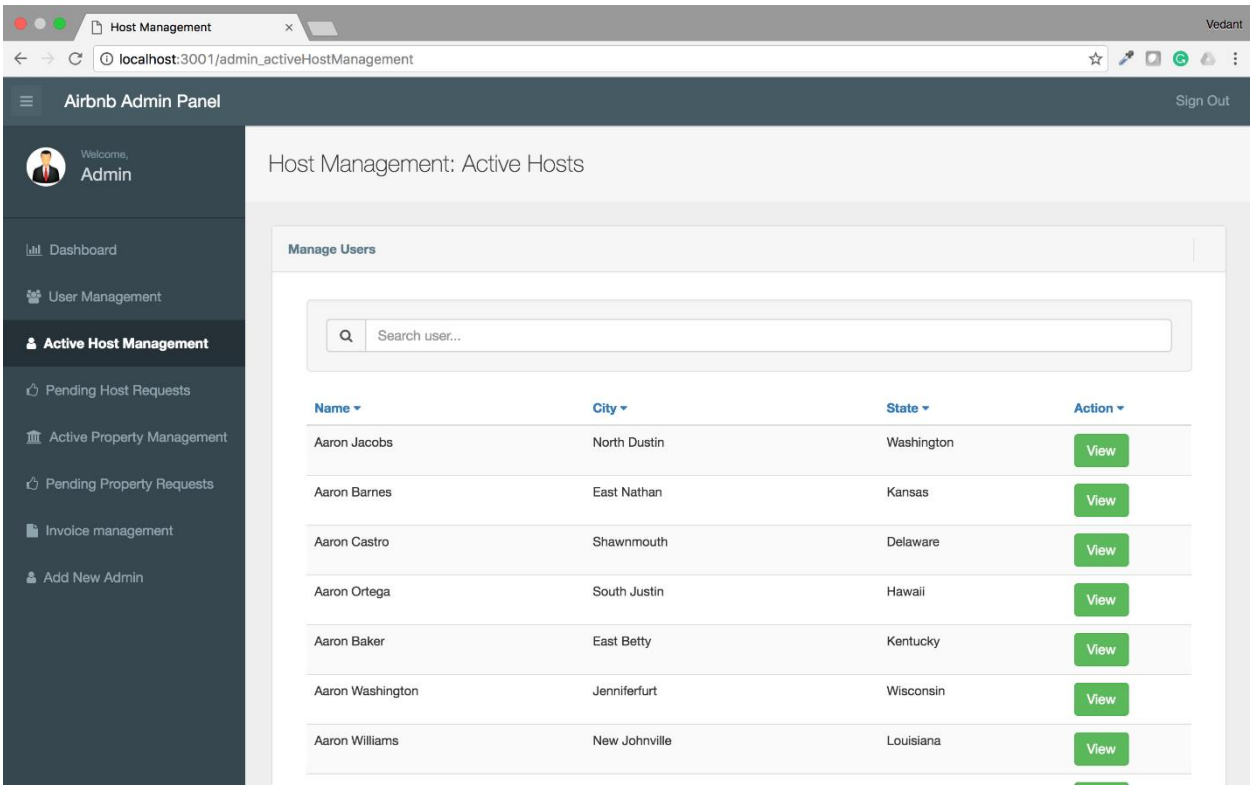
Users profile can be viewed by the admin



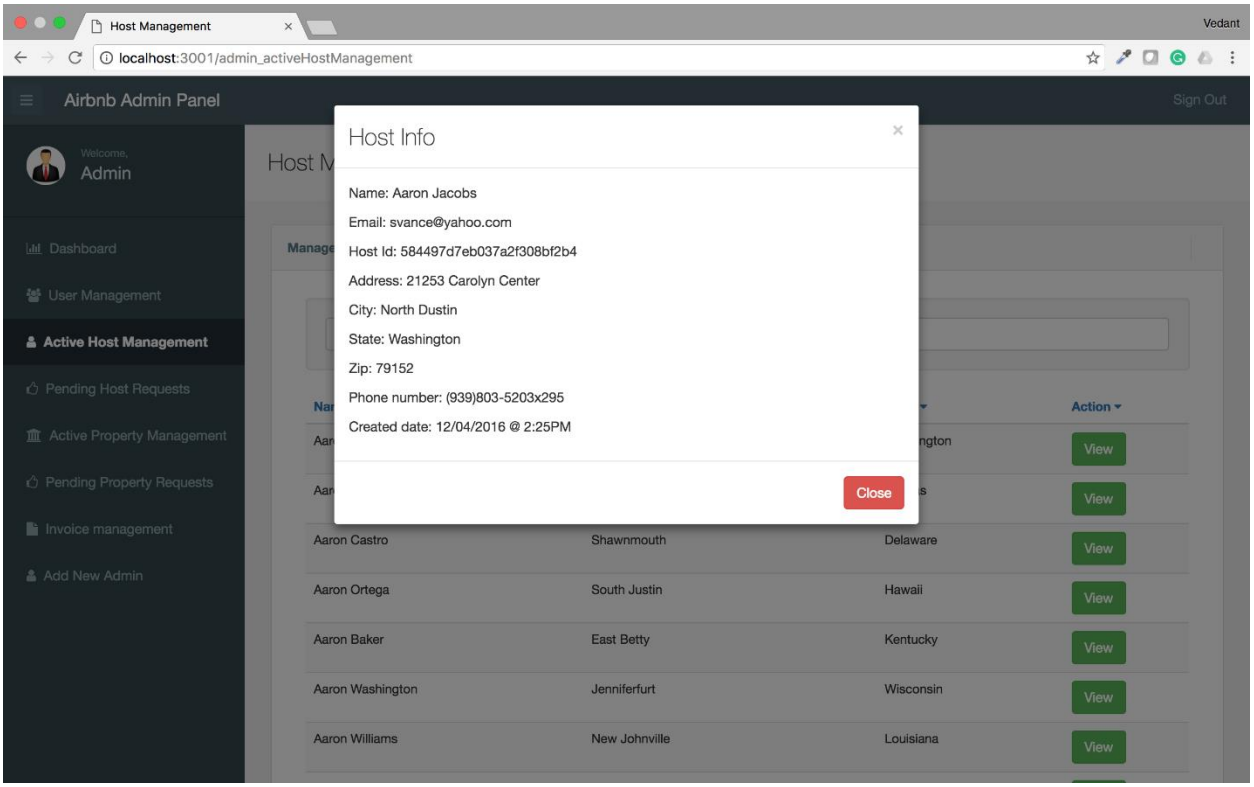
User can be searched based on his Name, city or state(Area)



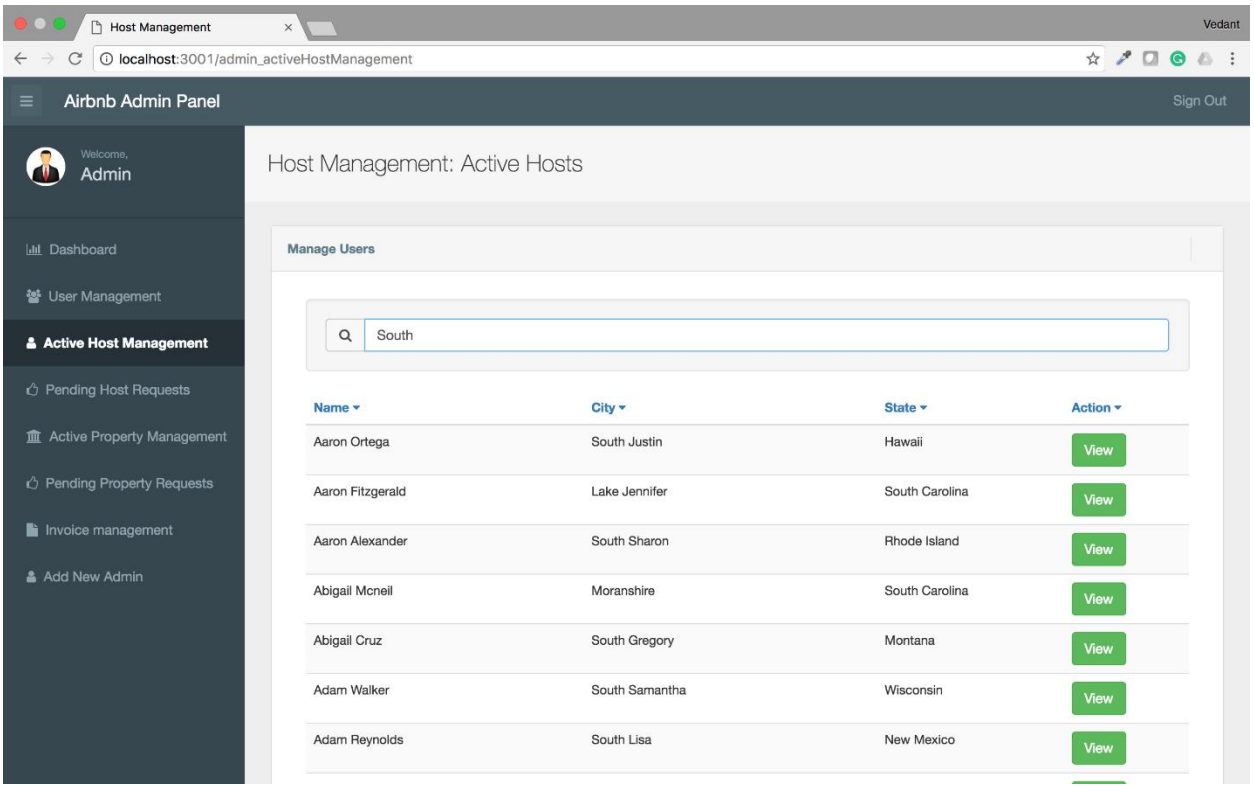
List of all active hosts



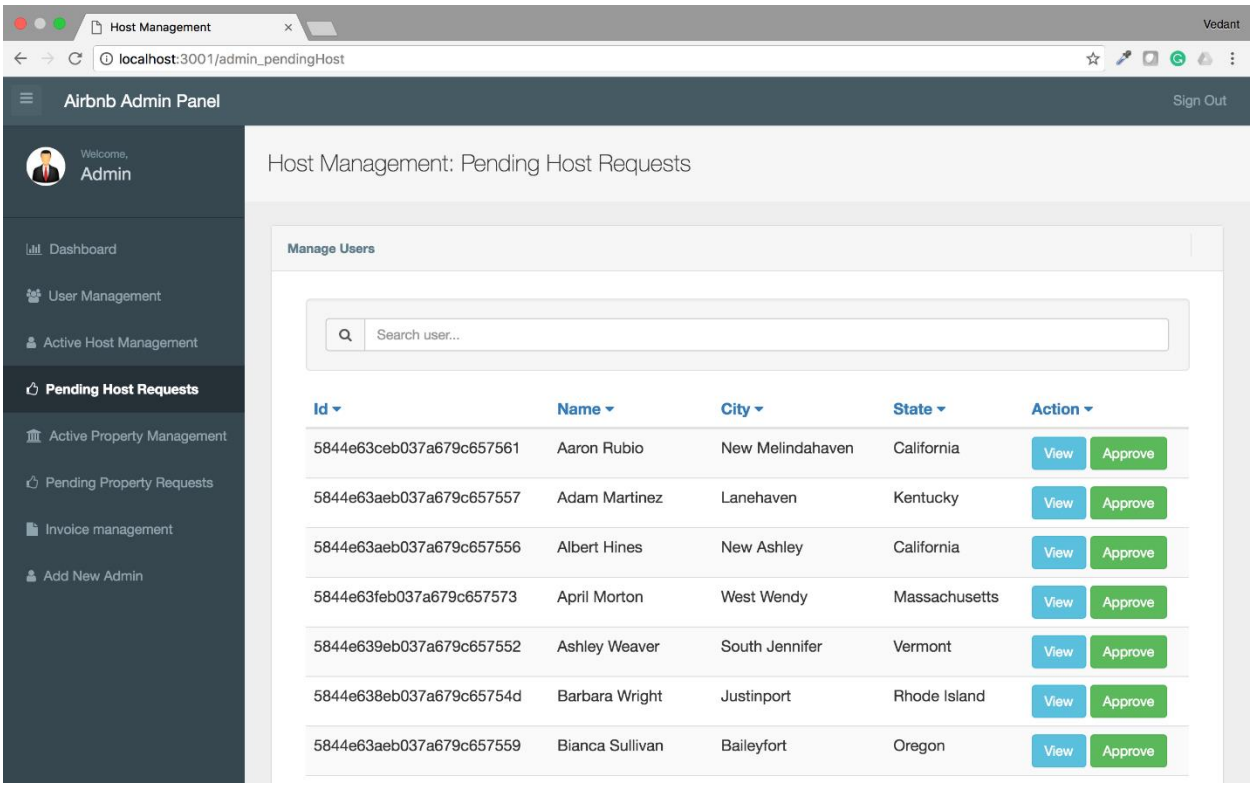
Host profile can be viewed by the admin



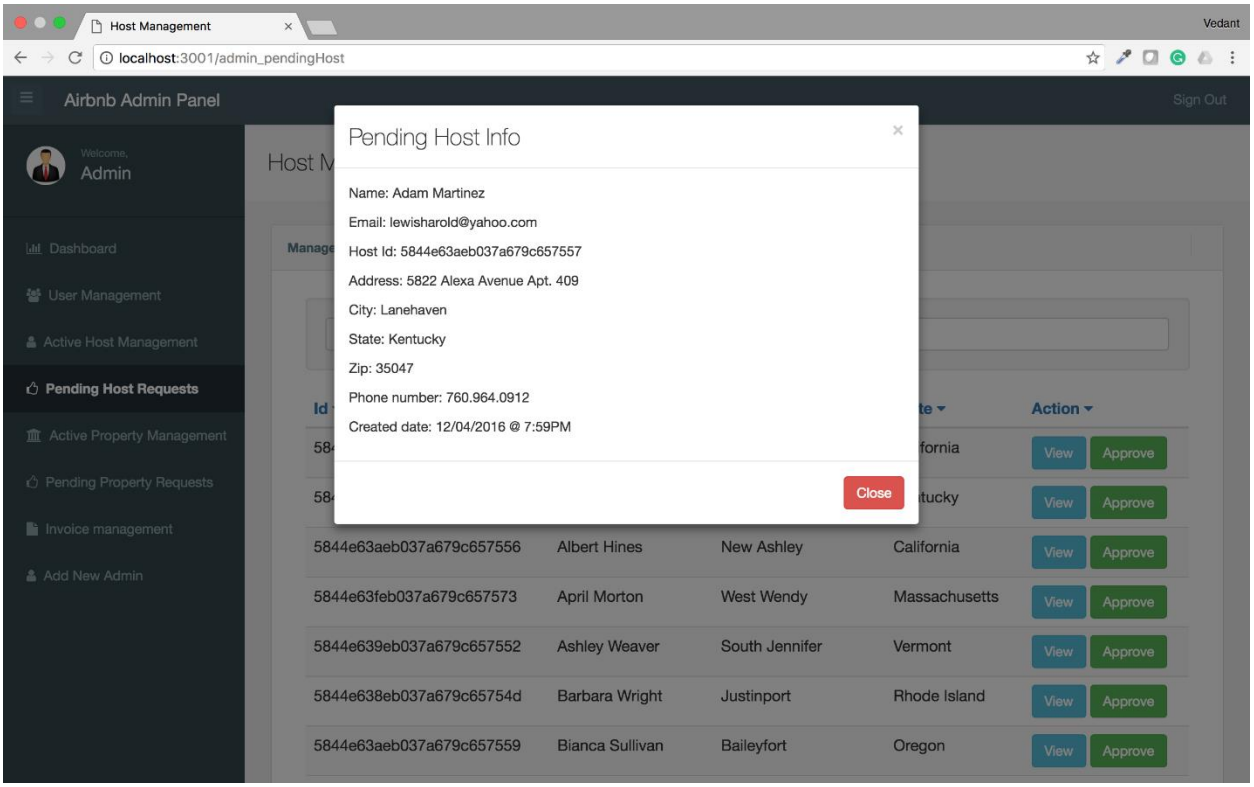
Host can be searched by the admin based on his Name, city, state(area)



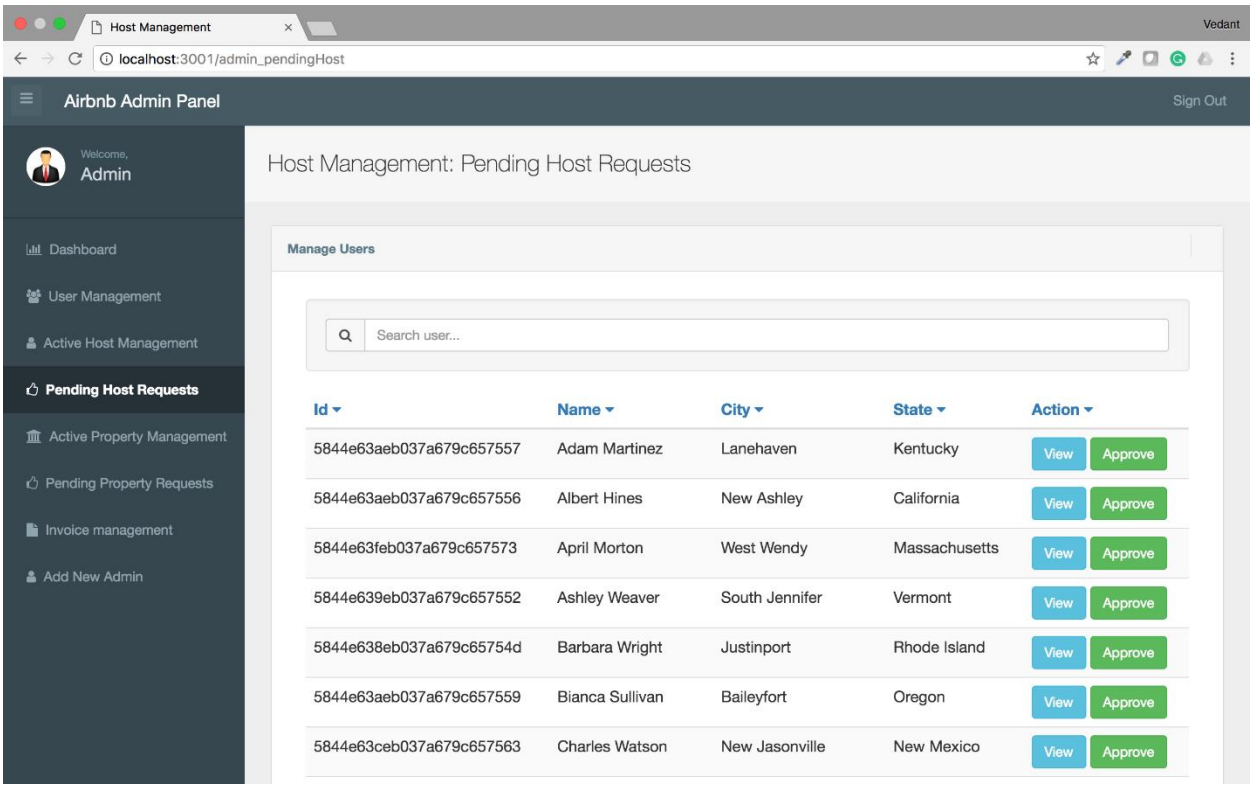
List of all pending hosts can be seen by the users



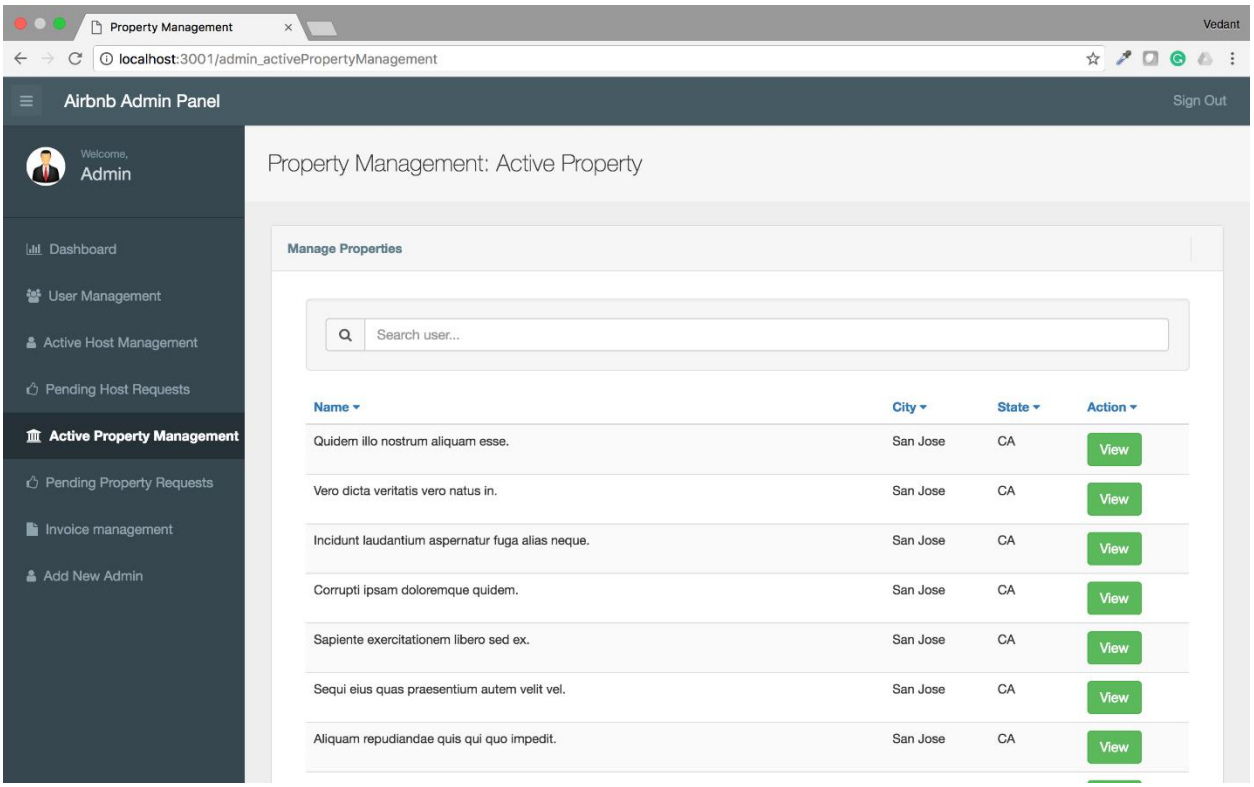
Profile of pending hosts can be viewed by the admin



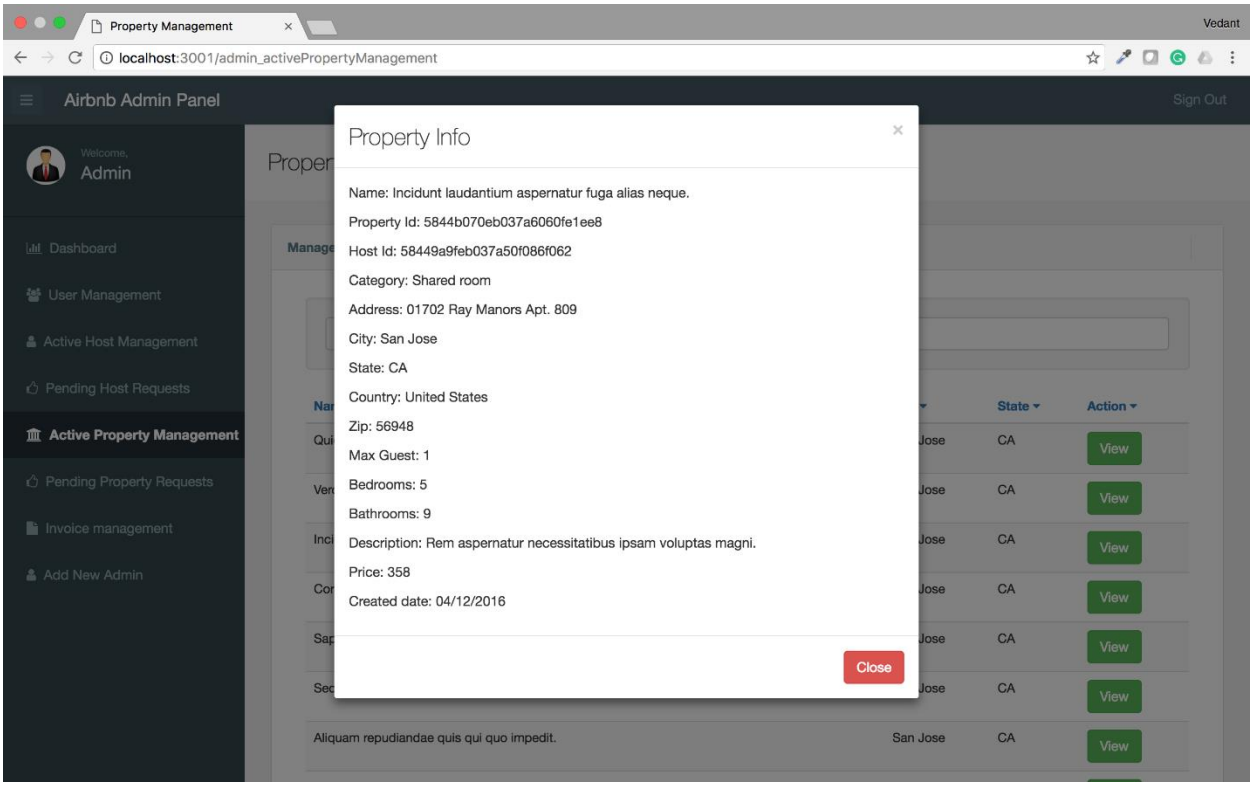
Host can be approved by the admin



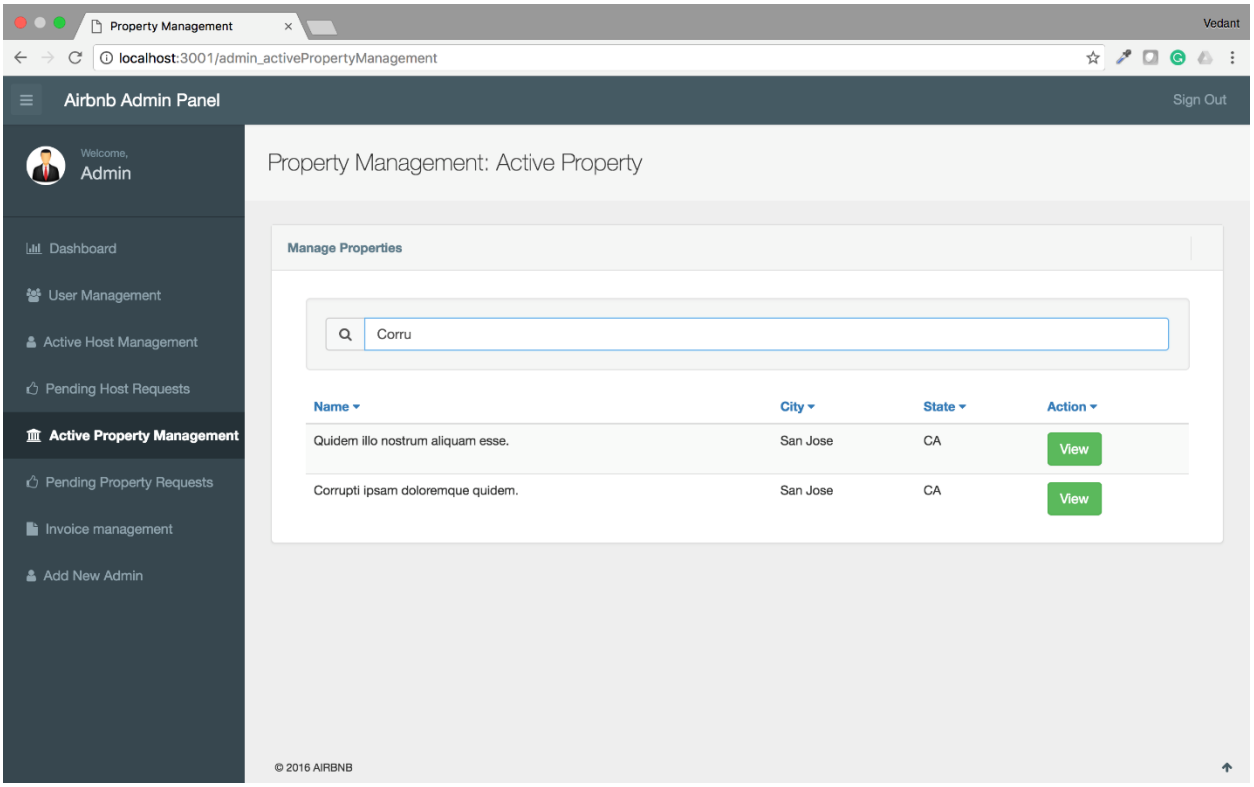
List of all active properties can be seen by the admin



Property details can be viewed by the admin

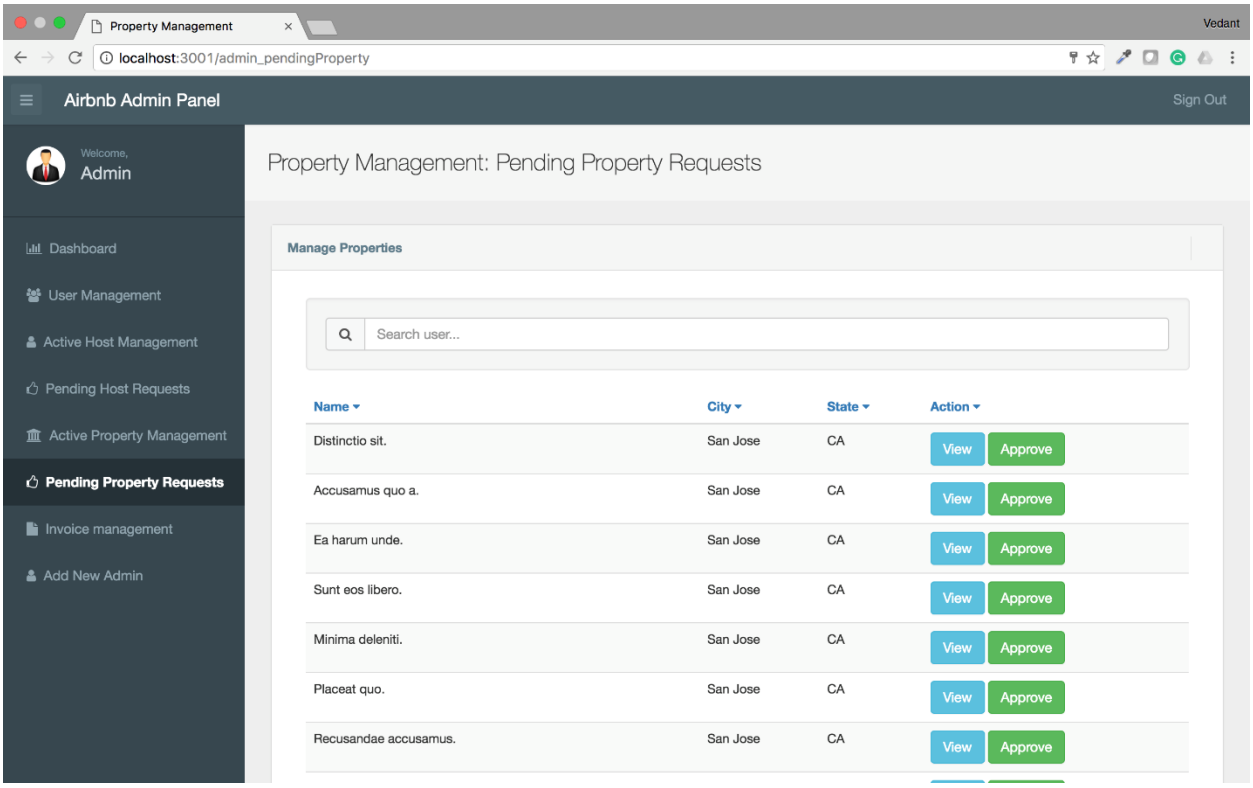


Property can be searched by the admin based on the properties Name, city, state(area)

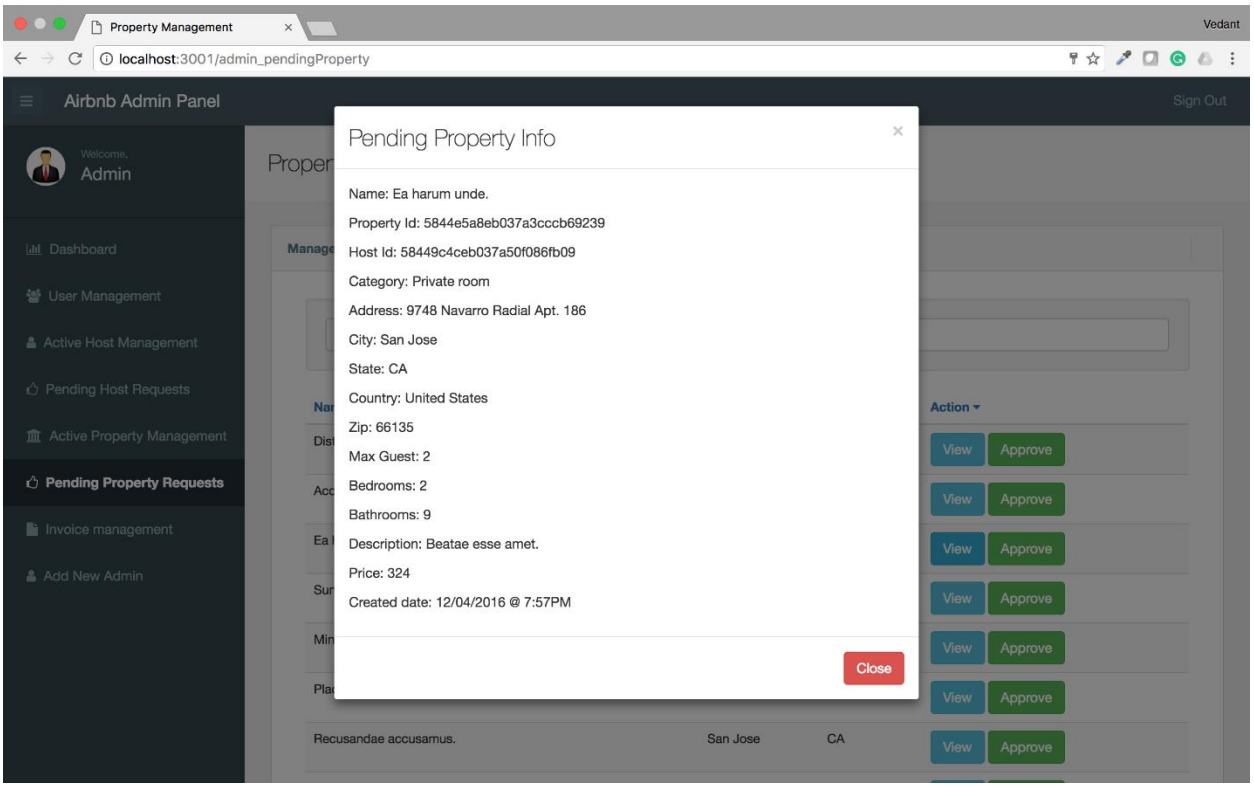


List of all pending properties can be viewed by the admin

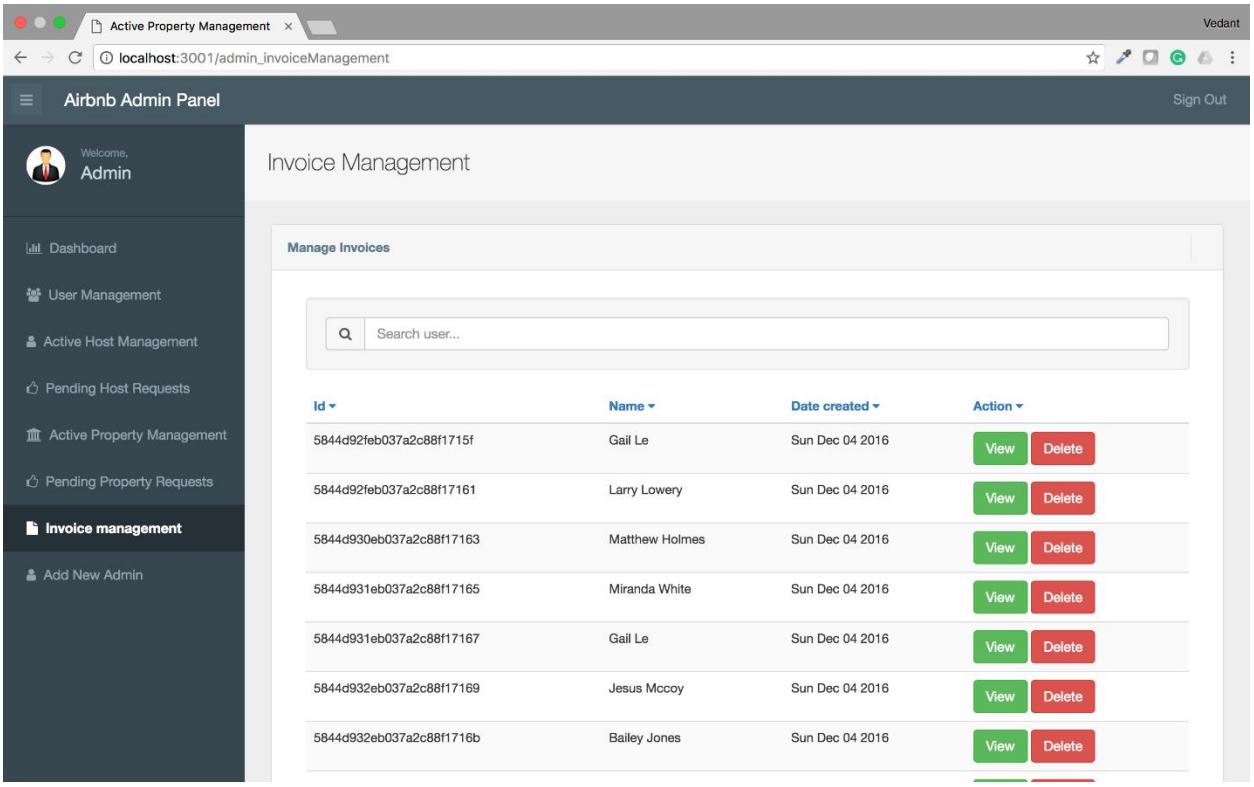
Pending properties can be approved by the admin



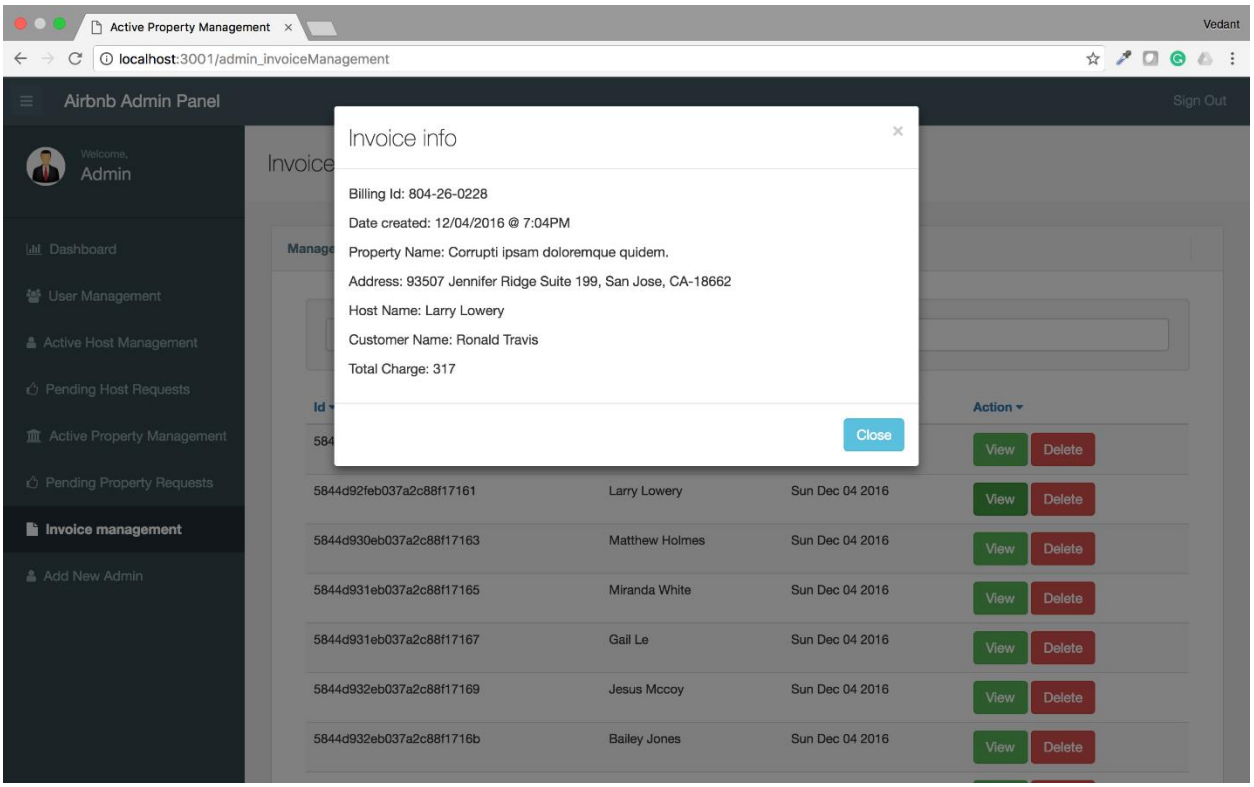
Property details can be viewed by the admin



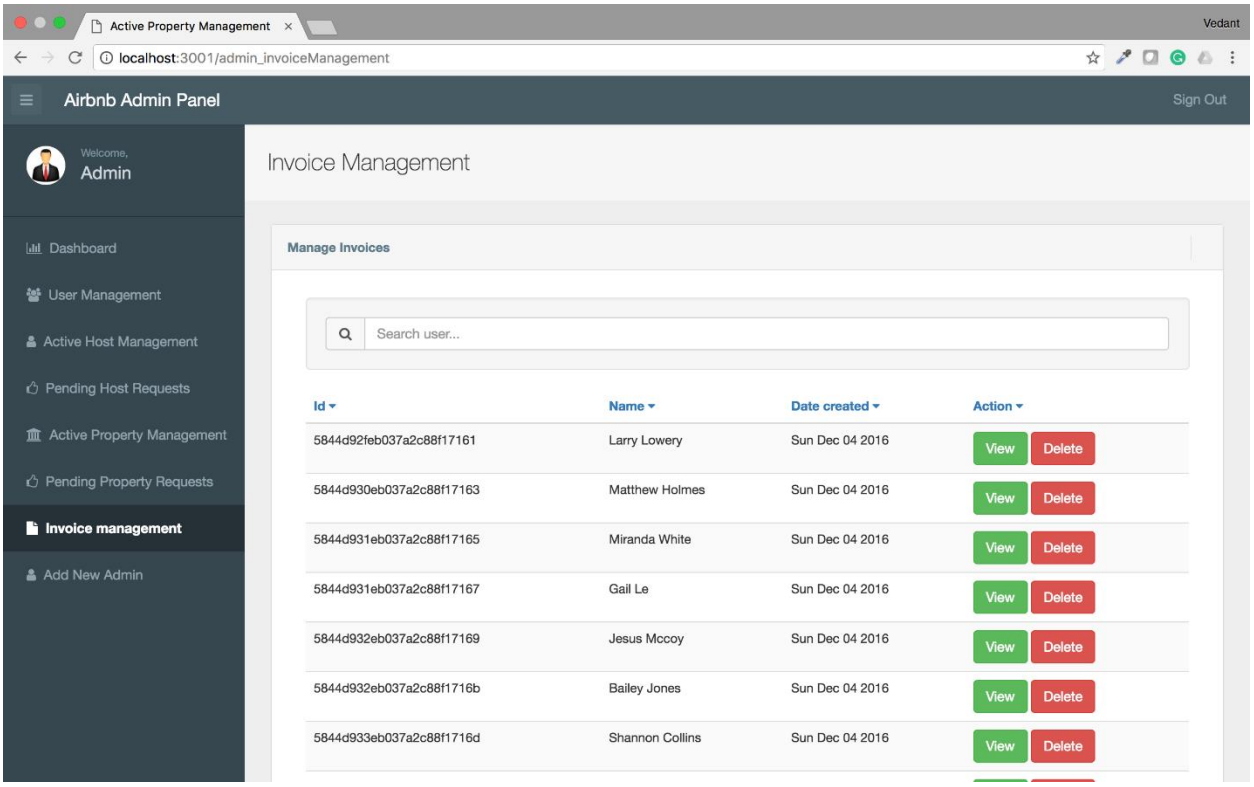
List of all the bills can be viewed by the admin



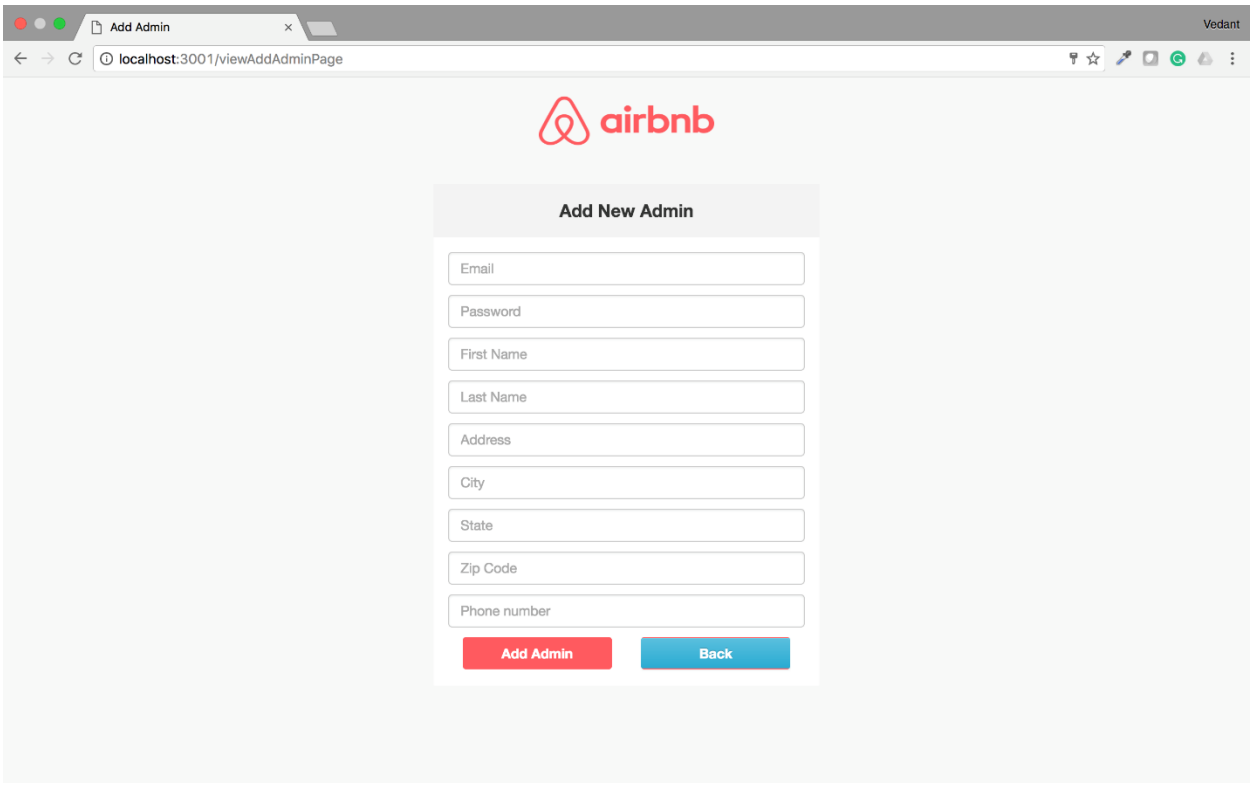
Details of all the bills can be viewed by the admin



Bills can be deleted by the admin



New admin can be added by the admin



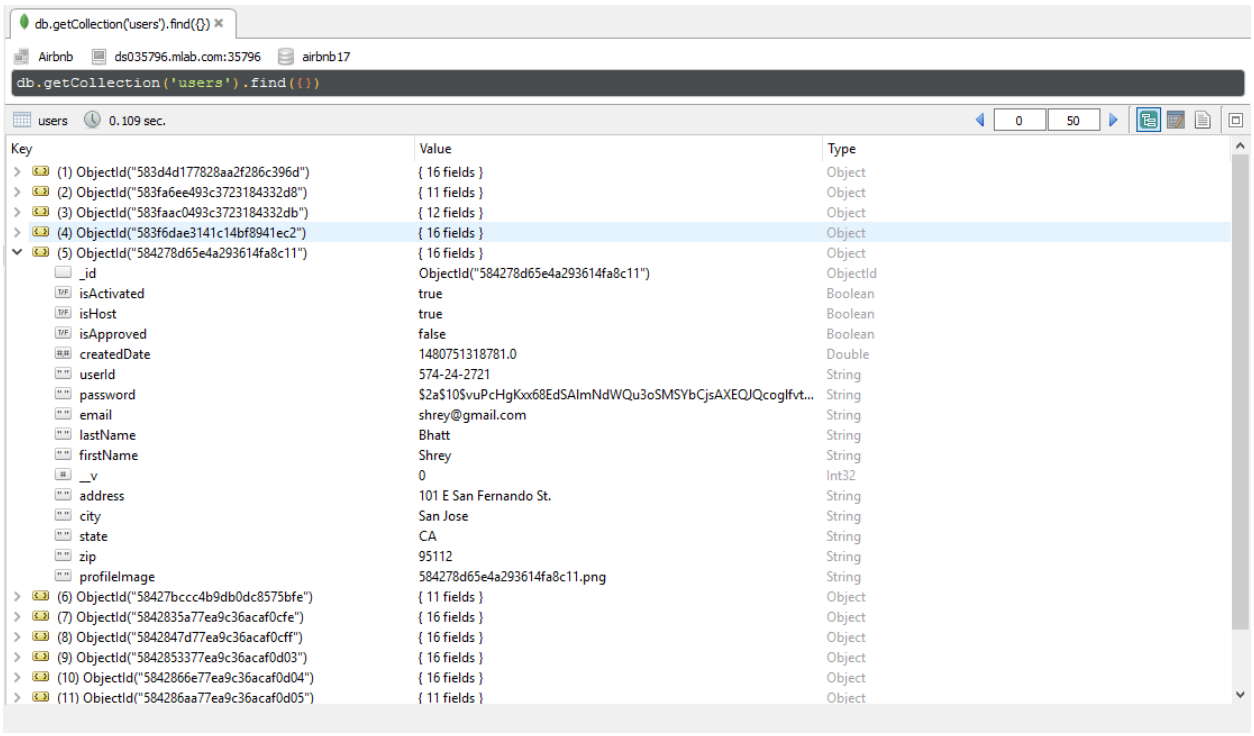
Database Design

MongoDB

1)User Collection

user.js

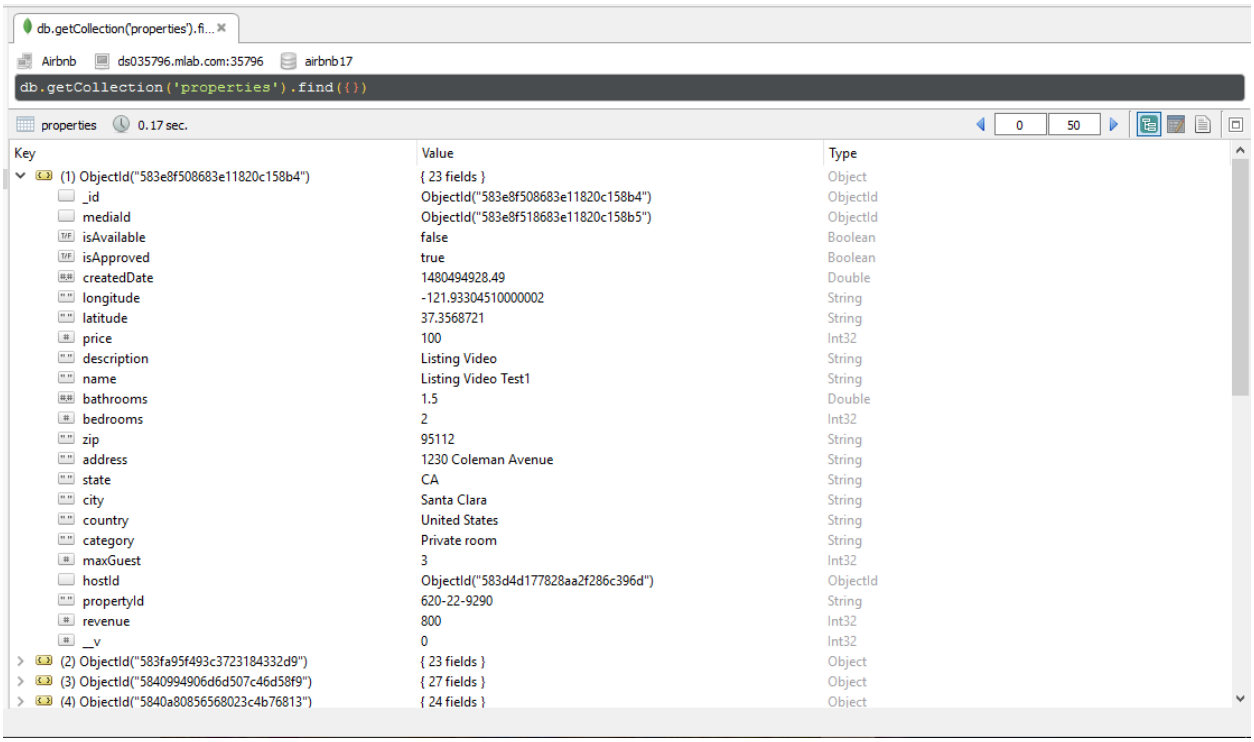
```
var mongoose = require('mongoose');
var Schema = mongoose.Schema;

var userSchema = new Schema({
    userId: {type: String},
    userName: {type: String},
    firstName: {type: String},
    lastName: {type: String},
    password: {type: String},
    email: {type: String, unique:true},
    address: {type: String},
    city: {type: String},
    state: {type: String},
    zip: {type: String},
    phoneNumber: {type: String},
    profileImage: {type: String},
    cardNumber: {type: Number},
    cvv: {type: Number},
    expDate: {type: Number},
    isHost: {type: Boolean},
    isActivated: {type: Boolean},
    isApproved: {type: Boolean},
    createdDate: {type: Number}
});

var User = mongoose.model('User', userSchema);

module.exports = User;
```


db.getCollection('users').find({})
Airbnb
ds035796.mlab.com:35796
airbnb17
db.getCollection('users').find({})
users
0.109 sec.
0
50
Key
Value
Type
(1) ObjectId("583d4d177828aa2f286c396d")
{ 16 fields }
Object
(2) ObjectId("583fa6ee493c3723184332d8")
{ 11 fields }
Object
(3) ObjectId("583faac0493c3723184332db")
{ 12 fields }
Object
(4) ObjectId("583f6dae3141c14bf8941ec2")
{ 16 fields }
Object
(5) ObjectId("584278d65e4a293614fa8c11")
{ 16 fields }
Object
_id
ObjectId("584278d65e4a293614fa8c11")
ObjectId
isActivated
true
Boolean
isHost
true
Boolean
isApproved
false
Boolean
createdDate
1480751318781.0
Double
userId
574-24-2721
String
password
$2a$10$vuPcHgKxx68EdSAlmNdWQu3oSMSYbCjsAXEQJQcoglfvt...
String
email
shrey@gmail.com
String
lastName
Bhatt
String
firstName
Shrey
String
__v
0
Int32
address
101 E San Fernando St.
String
city
San Jose
String
state
CA
String
zip
95112
String
profileImage
584278d65e4a293614fa8c11.png
String
(6) ObjectId("58427bccc4b9db0dc8575bfe")
{ 11 fields }
Object
(7) ObjectId("5842835a77ea9c36acaf0cfe")
{ 16 fields }
Object
(8) ObjectId("5842847d77ea9c36acaf0cff")
{ 16 fields }
Object
(9) ObjectId("5842853377ea9c36acaf0d03")
{ 16 fields }
Object
(10) ObjectId("5842866e77ea9c36acaf0d04")
{ 16 fields }
Object
(11) ObjectId("584286aa77ea9c36acaf0d05")
{ 11 fields }
Object

2)Property Collection

property.js

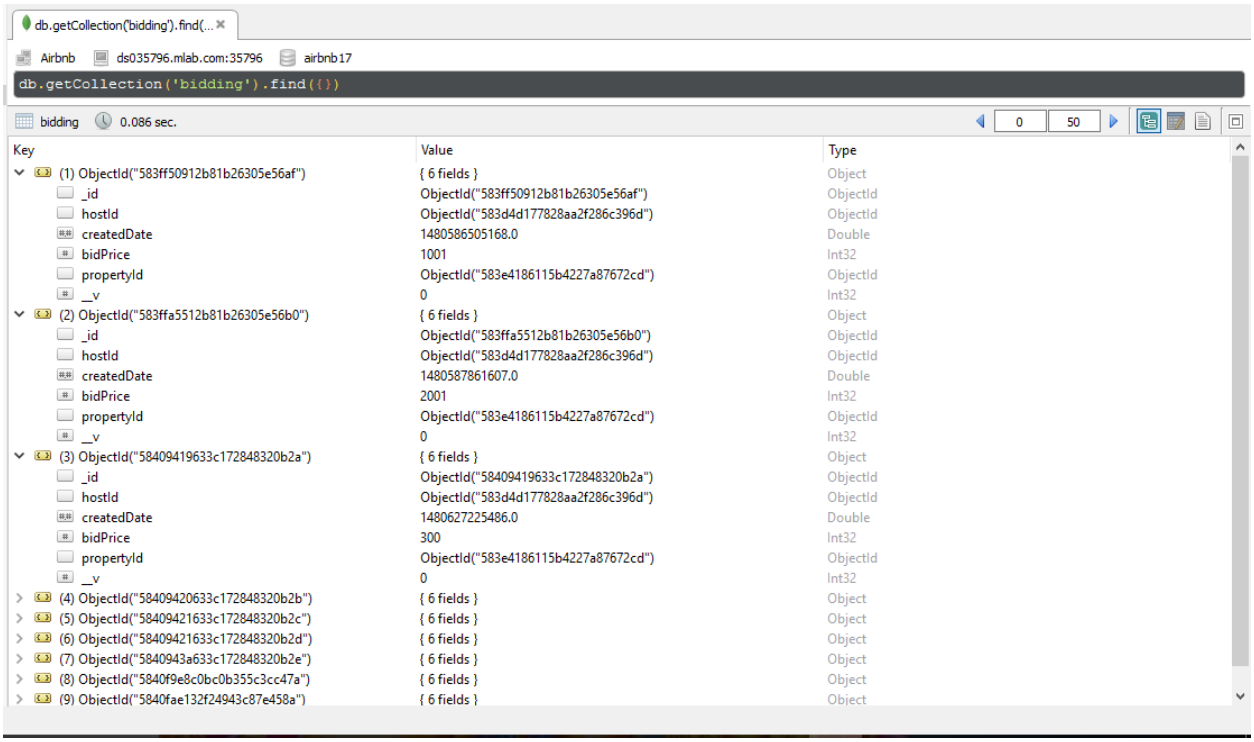
```
var mongoose = require('mongoose');
var Schema = mongoose.Schema;
require('./media');
var propertySchema = new Schema({
    propertyId: {type: String},
    name: {type: String},
    hostId: {type: Schema.Types.ObjectId, ref: 'User'},
    mediaId: {type: Schema.Types.ObjectId, ref: 'Media'},
    category: {type: String},
    address: {type: String},
    city: {type: String},
    state: {type: String},
    country: {type: String},
    zip: {type: String},
    maxGuest: {type: Number},
    bedrooms: {type: Number},
    bathrooms: {type: Number},
    description: {type: String},
    price: {type: Number},
    maxBidPrice: {type: Number},
    biddingDueTime: {type: Number},
    isBidding: {type: Boolean},
    isBidCompleted: {type: Boolean},
    latestBidder: {type: Schema.Types.ObjectId, ref: 'User'},
    isApproved: {type: Boolean},
    isAvailable: {type: Boolean},
    latitude: {type: String},
    longitude: {type: String},
    createdDate: {type: Number},
    revenue: {type: Number},
    startDate:{type:String},
    endDate:{type:String},
    multiplier:{type:String}
});
```


db.getCollection('properties').fi...
Airbnb
ds035796.mlab.com:35796
airbnb17
db.getCollection('properties').find({})
properties
0.17 sec.
0
50
Key
Value
Type
(1) ObjectId("583e8f508683e11820c158b4")
{ 23 fields }
Object
_id
ObjectId("583e8f508683e11820c158b4")
ObjectId
mediaId
ObjectId("583e8f518683e11820c158b5")
ObjectId
isAvailable
false
Boolean
isApproved
true
Boolean
createdDate
1480494928.49
Double
longitude
-121.93304510000002
String
latitude
37.3568721
String
price
100
Int32
description
Listing Video
String
name
Listing Video Test1
String
bathrooms
1.5
Double
bedrooms
2
Int32
zip
95112
String
address
1230 Coleman Avenue
String
state
CA
String
city
Santa Clara
String
country
United States
String
category
Private room
String
maxGuest
3
Int32
hostId
ObjectId("583d4d177828aa2f286c396d")
ObjectId
propertyId
620-22-9290
String
revenue
800
Int32
__v
0
Int32
(2) ObjectId("583fa95f493c3723184332d9")
{ 23 fields }
Object
(3) ObjectId("5840994906d6d507c46d58f9")
{ 27 fields }
Object
(4) ObjectId("5840a80856568023c4b76813")
{ 24 fields }
Object

3)Bidding Collection

bidding.js

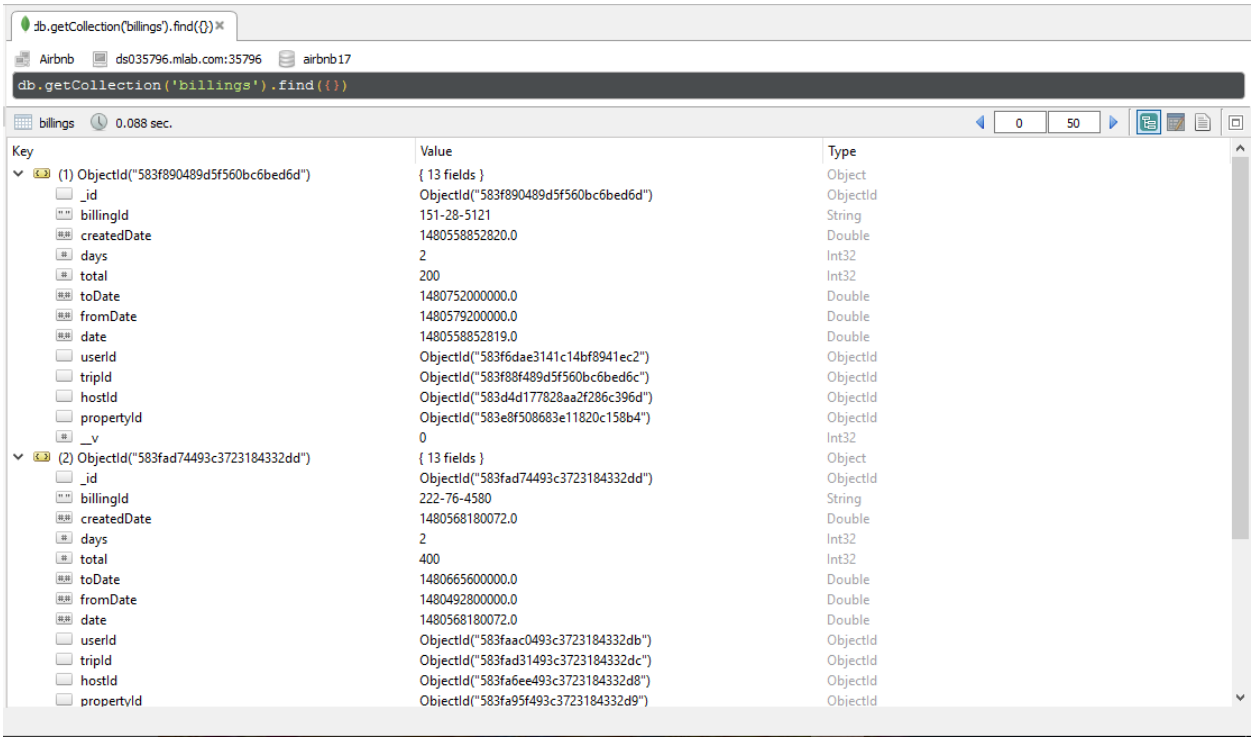
```
var mongoose = require('mongoose');
var Schema = mongoose.Schema;

var biddingSchema = new Schema({
    hostId: {type: Schema.Types.ObjectId, ref: 'user'},
    userId: {type: Schema.Types.ObjectId, ref: 'user'},
    propertyId: {type: Schema.Types.ObjectId, ref: 'property'},
    bidPrice: {type: Number},
    createdDate: {type: Number}
});

var Bidding = mongoose.model('bidding', biddingSchema,'bidding');
module.exports = Bidding;
```


db.getCollection('bidding').find(...
Airbnb
ds035796.mlab.com:35796
airbnb17
db.getCollection('bidding').find({})
bidding
0.086 sec.
0
50
Key
Value
Type
(1) ObjectId("583ff50912b81b26305e56af")
{ 6 fields }
Object
_id
ObjectId("583ff50912b81b26305e56af")
ObjectId
hostId
ObjectId("583d4d177828aa2f286c396d")
ObjectId
createdDate
1480586505168.0
Double
bidPrice
1001
Int32
propertyId
ObjectId("583e4186115b4227a87672cd")
ObjectId
__v
0
Int32
(2) ObjectId("583ffa5512b81b26305e56b0")
{ 6 fields }
Object
_id
ObjectId("583ffa5512b81b26305e56b0")
ObjectId
hostId
ObjectId("583d4d177828aa2f286c396d")
ObjectId
createdDate
1480587861607.0
Double
bidPrice
2001
Int32
propertyId
ObjectId("583e4186115b4227a87672cd")
ObjectId
__v
0
Int32
(3) ObjectId("58409419633c172848320b2a")
{ 6 fields }
Object
_id
ObjectId("58409419633c172848320b2a")
ObjectId
hostId
ObjectId("583d4d177828aa2f286c396d")
ObjectId
createdDate
1480627225486.0
Double
bidPrice
300
Int32
propertyId
ObjectId("583e4186115b4227a87672cd")
ObjectId
__v
0
Int32
(4) ObjectId("58409420633c172848320b2b")
{ 6 fields }
Object
(5) ObjectId("58409421633c172848320b2c")
{ 6 fields }
Object
(6) ObjectId("58409421633c172848320b2d")
{ 6 fields }
Object
(7) ObjectId("5840943a633c172848320b2e")
{ 6 fields }
Object
(8) ObjectId("5840f9e8c0bc0b355c3cc47a")
{ 6 fields }
Object
(9) ObjectId("5840fae132f24943c87e458a")
{ 6 fields }
Object

4)Billing Collection

billing.js

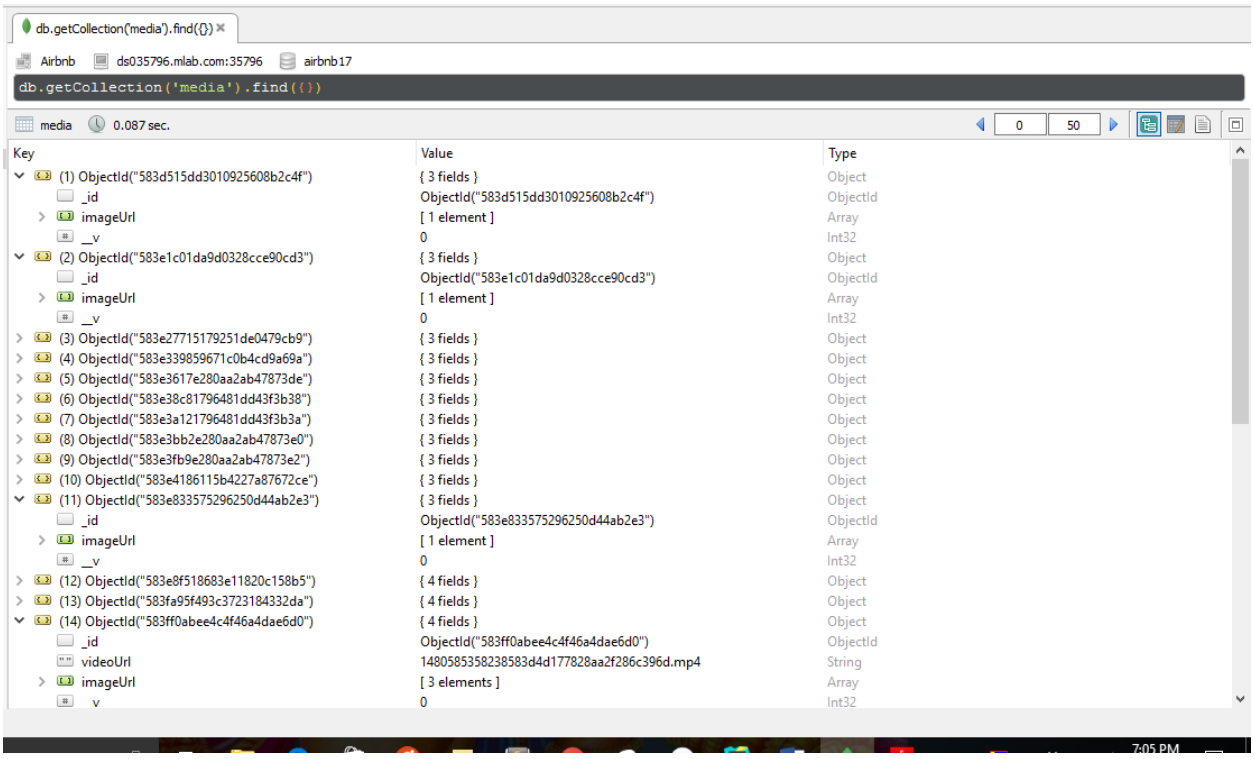
```
var mongoose = require('mongoose');
var Schema = mongoose.Schema;
require('./property');
require('./trip');

var billingSchema = new Schema({
    billingId: {type: String},
    propertyId: {type: Schema.Types.ObjectId, ref: 'Property'},
    hostId: {type: Schema.Types.ObjectId, ref: 'User'},
    tripId: {type: Schema.Types.ObjectId, ref: 'Trip'},
    userId: {type: Schema.Types.ObjectId, ref: 'User'},
    date: {type: Number},
    fromDate: {type: Number},
    toDate: {type: Number},
    total: {type: Number},
    price:{type:Number},
    multiplier:{type:Number},
    days: {type: Number},
    createdDate: {type: Number}

});

var Billing = mongoose.model('Billing', billingSchema);
module.exports = Billing;
```


db.getCollection('billings').find({})
Airbnb
ds035796.mlab.com:35796
airbnb17
db.getCollection('billings').find({})
billings
0.088 sec.
0
50
Key
Value
Type
(1) ObjectId("583f890489d5f560bc6bed6d")
{ 13 fields }
Object
_id
ObjectId("583f890489d5f560bc6bed6d")
ObjectId
billingId
151-28-5121
String
createdDate
1480558852820.0
Double
days
2
Int32
total
200
Int32
toDate
1480752000000.0
Double
fromDate
1480579200000.0
Double
date
1480558852819.0
Double
userId
ObjectId("583f6dae3141c14bf8941ec2")
ObjectId
tripId
ObjectId("583f88f489d5f560bc6bed6c")
ObjectId
hostId
ObjectId("583d4d177828aa2f286c396d")
ObjectId
propertyId
ObjectId("583e8f508683e11820c158b4")
ObjectId
__v
0
Int32
(2) ObjectId("583fad74493c3723184332dd")
{ 13 fields }
Object
_id
ObjectId("583fad74493c3723184332dd")
ObjectId
billingId
222-76-4580
String
createdDate
1480568180072.0
Double
days
2
Int32
total
400
Int32
toDate
1480665600000.0
Double
fromDate
1480492800000.0
Double
date
1480568180072.0
Double
userId
ObjectId("583faac0493c3723184332db")
ObjectId
tripId
ObjectId("583fad31493c3723184332dc")
ObjectId
hostId
ObjectId("583fa6ee493c3723184332d8")
ObjectId
propertyId
ObjectId("583fa95f493c3723184332d9")
ObjectId

5)Media Collection

media.js

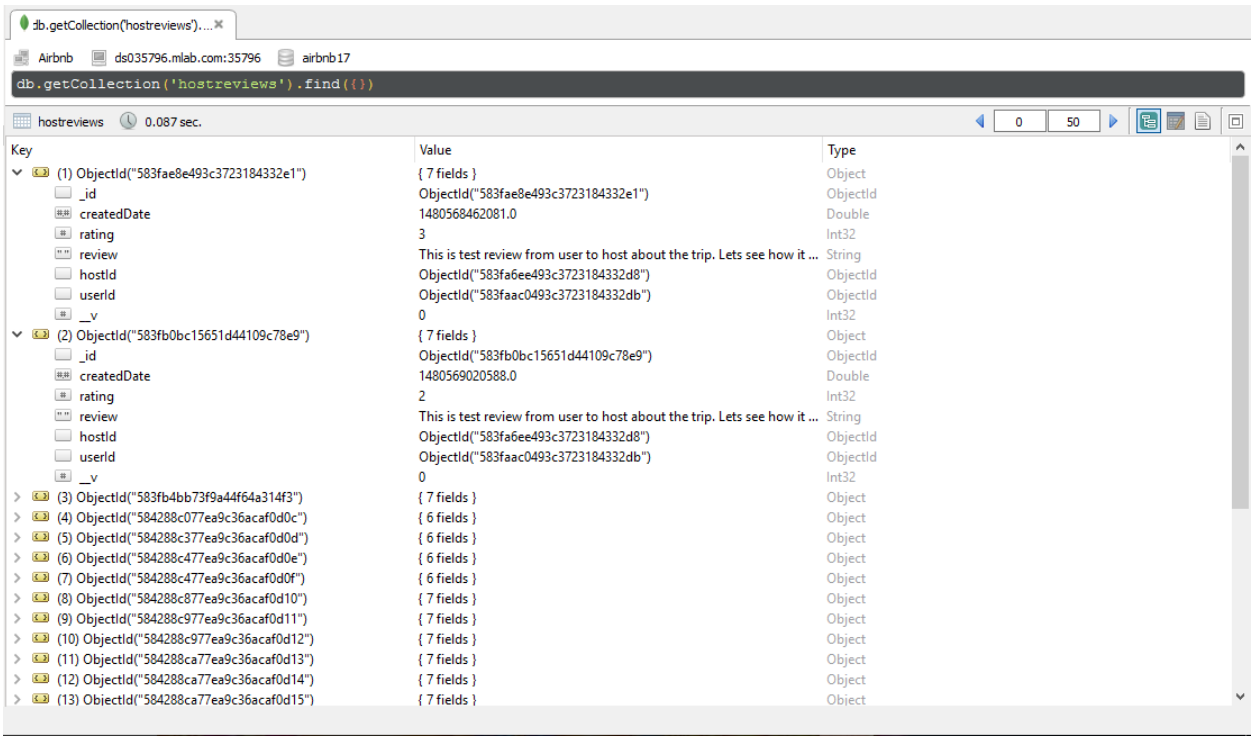
```
var mongoose = require('mongoose');
var Schema = mongoose.Schema;

var mediaSchema = new Schema({
    imageUrl: [String],
    videoUrl: {type: String}
});

var Media = mongoose.model('Media', mediaSchema);

module.exports = Media;
```


db.getCollection('media').find({})
Airbnb
ds035796.mlab.com:35796
airbnb17
db.getCollection('media').find({})
media
0.087 sec.
0
50
Key
Value
Type
(1) ObjectId("583d515dd3010925608b2c4f")
{ 3 fields }
Object
_id
ObjectId("583d515dd3010925608b2c4f")
ObjectId
imageUrl
[1 element]
Array
__v
0
Int32
(2) ObjectId("583e1c01da9d0328cce90cd3")
{ 3 fields }
Object
_id
ObjectId("583e1c01da9d0328cce90cd3")
ObjectId
imageUrl
[1 element]
Array
__v
0
Int32
(3) ObjectId("583e27715179251de0479cb9")
{ 3 fields }
Object
(4) ObjectId("583e339859671c0b4cd9a69a")
{ 3 fields }
Object
(5) ObjectId("583e3617e280aa2ab47873de")
{ 3 fields }
Object
(6) ObjectId("583e38c81796481dd43f3b38")
{ 3 fields }
Object
(7) ObjectId("583e3a121796481dd43f3b3a")
{ 3 fields }
Object
(8) ObjectId("583e3bb2e280aa2ab47873e0")
{ 3 fields }
Object
(9) ObjectId("583e3fb9e280aa2ab47873e2")
{ 3 fields }
Object
(10) ObjectId("583e4186115b4227a87672ce")
{ 3 fields }
Object
(11) ObjectId("583e833575296250d44ab2e3")
{ 3 fields }
Object
_id
ObjectId("583e833575296250d44ab2e3")
ObjectId
imageUrl
[1 element]
Array
__v
0
Int32
(12) ObjectId("583e8f518683e11820c158b5")
{ 4 fields }
Object
(13) ObjectId("583fa95f493c3723184332da")
{ 4 fields }
Object
(14) ObjectId("583ff0abee4c4f46a4dae6d0")
{ 4 fields }
Object
_id
ObjectId("583ff0abee4c4f46a4dae6d0")
ObjectId
videoUrl
1480585358238583d4d177828aa2f286c396d.mp4
String
imageUrl
[3 elements]
Array
v
0
Int32
7:05 PM

6)Host Review Collection

hostReview.js

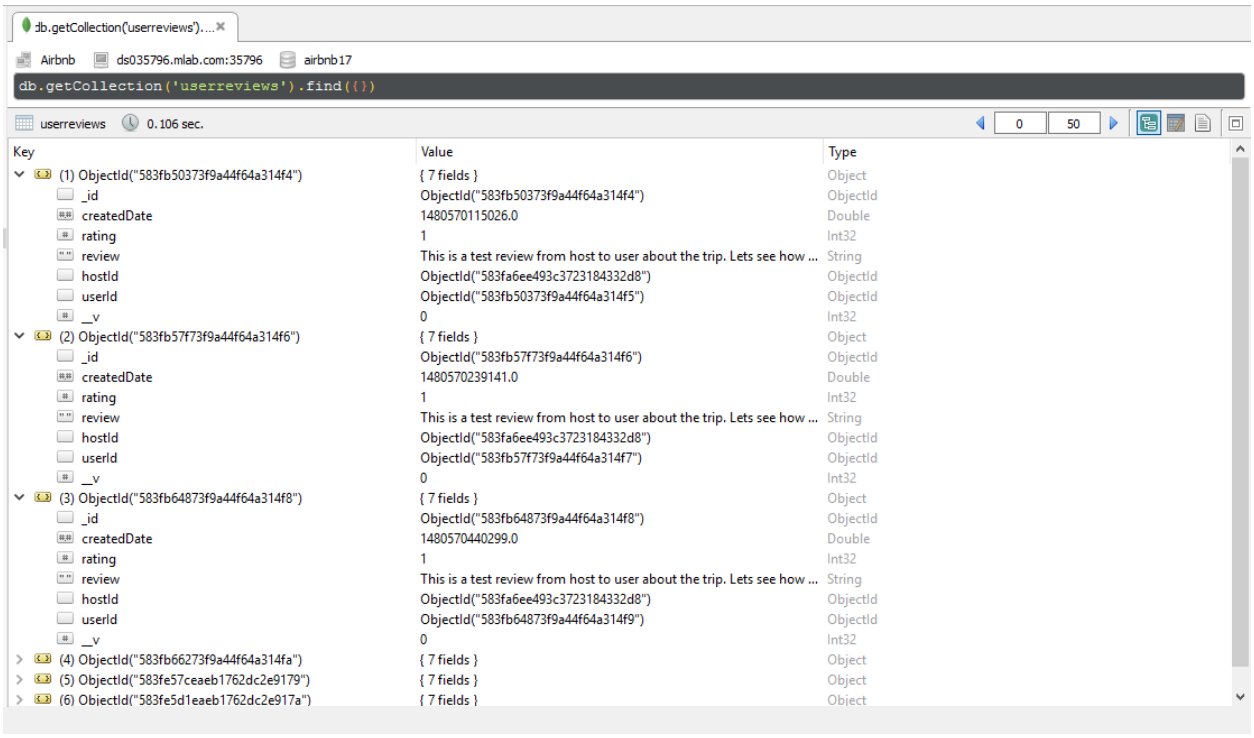
```
var mongoose = require('mongoose');
var Schema = mongoose.Schema;

var hostReviewSchema = new Schema({
    hostId: {type: Schema.Types.ObjectId, ref: 'User'},
    userId: {type: Schema.Types.ObjectId, ref: 'User'},
    review: {type: String},
    rating: {type: Number},
    imageUrl: {type: String},
    createdDate: {type: Number}

});

var HostReview = mongoose.model('HostReview', hostReviewSchema);
module.exports = HostReview;
```


db.getCollection('hostreviews'). ...
Airbnb
ds035796.mlab.com:35796
airbnb17
db.getCollection('hostreviews').find({})
hostreviews
0.087 sec.
0
50
Key
Value
Type
(1) ObjectId("583fae8e493c3723184332e1")
{ 7 fields }
Object
_id
ObjectId("583fae8e493c3723184332e1")
ObjectId
createdDate
1480568462081.0
Double
rating
3
Int32
review
This is test review from user to host about the trip. Lets see how it ...
String
hostId
ObjectId("583fa6ee493c3723184332d8")
ObjectId
userId
ObjectId("583faac0493c3723184332db")
ObjectId
__v
0
Int32
(2) ObjectId("583fb0bc15651d44109c78e9")
{ 7 fields }
Object
_id
ObjectId("583fb0bc15651d44109c78e9")
ObjectId
createdDate
1480569020588.0
Double
rating
2
Int32
review
This is test review from user to host about the trip. Lets see how it ...
String
hostId
ObjectId("583fa6ee493c3723184332d8")
ObjectId
userId
ObjectId("583faac0493c3723184332db")
ObjectId
__v
0
Int32
(3) ObjectId("583fb4bb73f9a44f64a314f3")
{ 7 fields }
Object
(4) ObjectId("584288c077ea9c36acaf0d0c")
{ 6 fields }
Object
(5) ObjectId("584288c377ea9c36acaf0d0d")
{ 6 fields }
Object
(6) ObjectId("584288c477ea9c36acaf0d0e")
{ 6 fields }
Object
(7) ObjectId("584288c477ea9c36acaf0d0f")
{ 6 fields }
Object
(8) ObjectId("584288c877ea9c36acaf0d10")
{ 7 fields }
Object
(9) ObjectId("584288c977ea9c36acaf0d11")
{ 7 fields }
Object
(10) ObjectId("584288c977ea9c36acaf0d12")
{ 7 fields }
Object
(11) ObjectId("584288ca77ea9c36acaf0d13")
{ 7 fields }
Object
(12) ObjectId("584288ca77ea9c36acaf0d14")
{ 7 fields }
Object
(13) ObjectId("584288ca77ea9c36acaf0d15")
{ 7 fields }
Object

7)User Review Collection

user.js

```
var mongoose = require('mongoose');
var Schema = mongoose.Schema;

var userSchema = new Schema({
    userId: {type: String},
    userName: {type: String},
    firstName: {type: String},
    lastName: {type: String},
    password: {type: String},
    email: {type: String, unique:true},
    address: {type: String},
    city: {type: String},
    state: {type: String},
    zip: {type: String},
    phoneNumber: {type: String},
    profileImage: {type: String},
    cardNumber: {type: Number},
    cvv: {type: Number},
    expDate: {type: Number},
    isHost: {type: Boolean},
    isActivated: {type: Boolean},
    isApproved: {type: Boolean},
    createdDate: {type: Number}
});

var User = mongoose.model('User', userSchema);

module.exports = User;
```


db.getCollection('userreviews')....
Airbnb
ds035796.mlab.com:35796
airbnb17
db.getCollection('userreviews').find({})
userreviews
0.106 sec.
0
50
Key
Value
Type
(1) ObjectId("583fb50373f9a44f64a314f4")
{ 7 fields }
Object
_id
ObjectId("583fb50373f9a44f64a314f4")
ObjectId
createdDate
1480570115026.0
Double
rating
1
Int32
review
This is a test review from host to user about the trip. Lets see how ...
String
hostId
ObjectId("583fa6ee493c3723184332d8")
ObjectId
userId
ObjectId("583fb50373f9a44f64a314f5")
ObjectId
__v
0
Int32
(2) ObjectId("583fb57f73f9a44f64a314f6")
{ 7 fields }
Object
_id
ObjectId("583fb57f73f9a44f64a314f6")
ObjectId
createdDate
1480570239141.0
Double
rating
1
Int32
review
This is a test review from host to user about the trip. Lets see how ...
String
hostId
ObjectId("583fa6ee493c3723184332d8")
ObjectId
userId
ObjectId("583fb57f73f9a44f64a314f7")
ObjectId
__v
0
Int32
(3) ObjectId("583fb64873f9a44f64a314f8")
{ 7 fields }
Object
_id
ObjectId("583fb64873f9a44f64a314f8")
ObjectId
createdDate
1480570440299.0
Double
rating
1
Int32
review
This is a test review from host to user about the trip. Lets see how ...
String
hostId
ObjectId("583fa6ee493c3723184332d8")
ObjectId
userId
ObjectId("583fb64873f9a44f64a314f9")
ObjectId
__v
0
Int32
(4) ObjectId("583fb66273f9a44f64a314fa")
{ 7 fields }
Object
(5) ObjectId("583fe57ceaeb1762dc2e9179")
{ 7 fields }
Object
(6) ObjectId("583fe5d1eaeb1762dc2e917a")
{ 7 fields }
Object

8)Property Review Collection

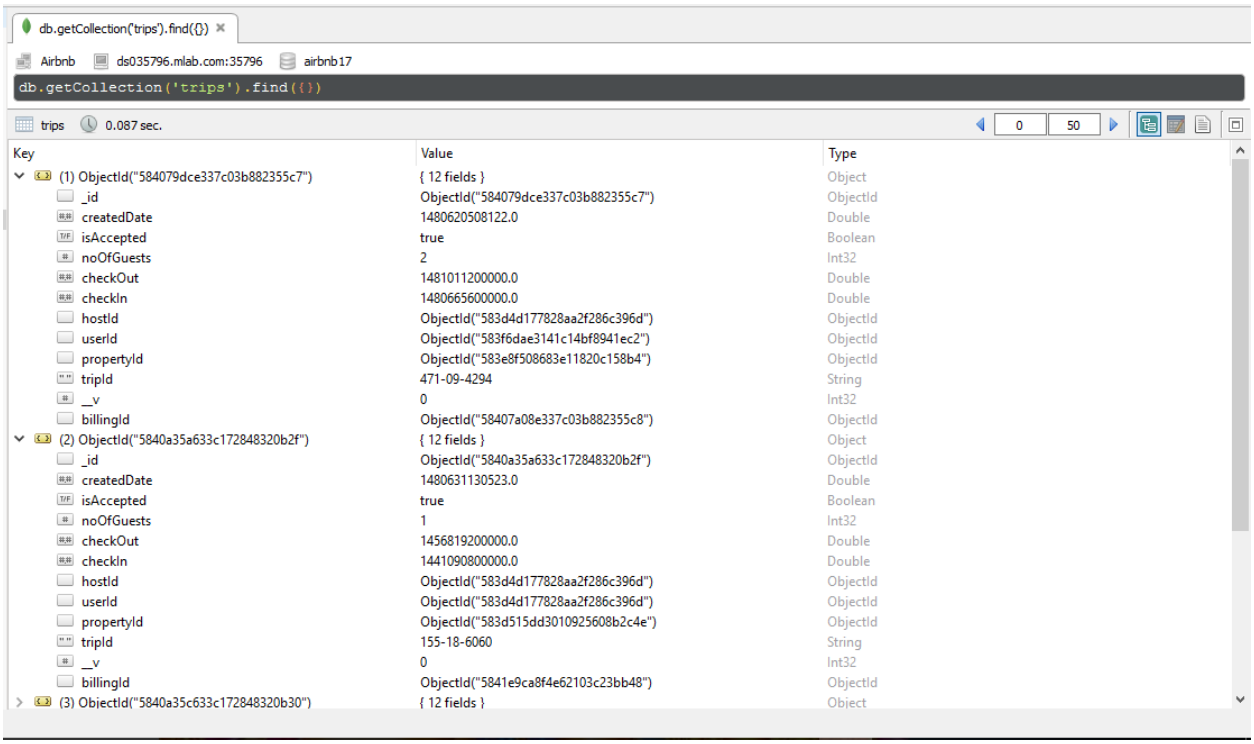
propertyReview.js

```
var mongoose = require('mongoose');
var Schema = mongoose.Schema;

var propertyReviewSchema = new Schema({
    userId: {type: Schema.Types.ObjectId, ref: 'User'},
    propertyId: {type: Schema.Types.ObjectId, ref: 'Property'},
    review: {type: String},
    rating: {type: Number},
    imageUrl: {type: String},
    createdDate: {type: Number}

});

var PropertyReview = mongoose.model('PropertyReview', propertyReviewSchema);
module.exports = PropertyReview;
```

9)Trip Collection

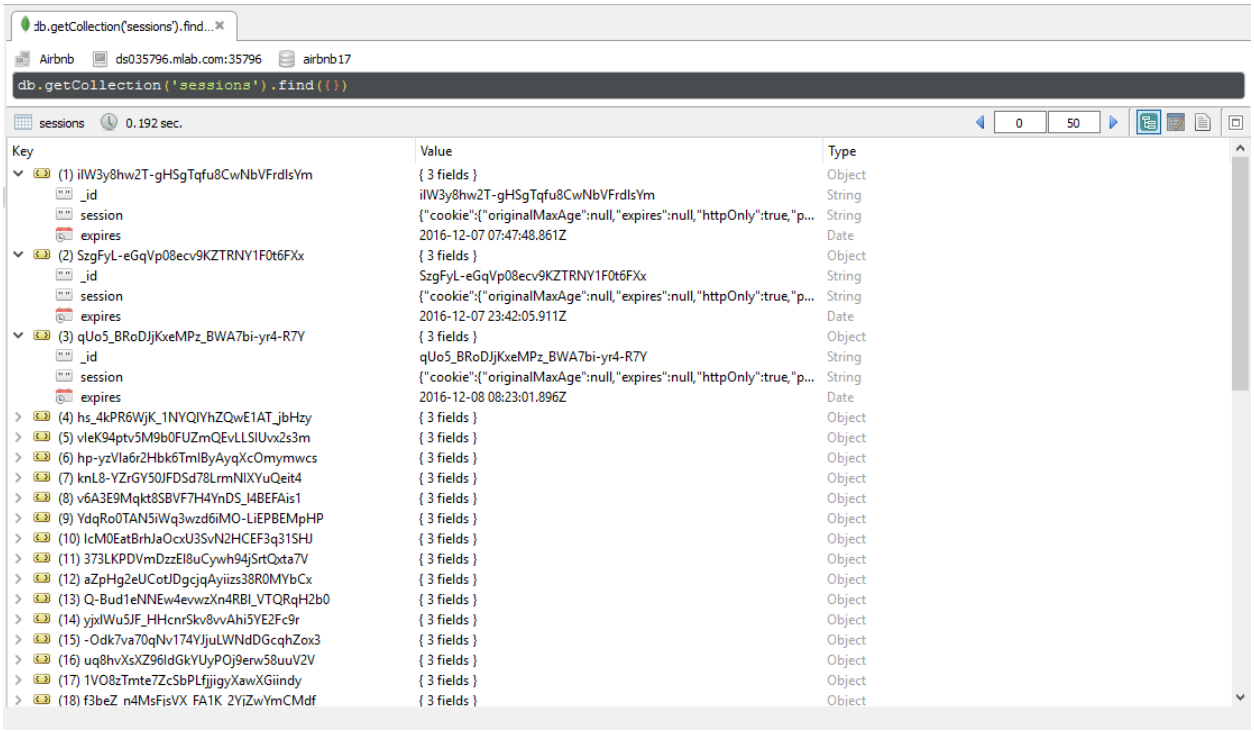
trip.js

```
var mongoose = require('mongoose');
var Schema = mongoose.Schema;
require('./property');
require('./billing');
var tripSchema = new Schema({
    tripId: {type: String},
    propertyId: {type: Schema.Types.ObjectId, ref: 'Property'},
    userId: {type: Schema.Types.ObjectId, ref: 'User'},
    hostId: {type: Schema.Types.ObjectId, ref: 'User'},
    billingId: {type: Schema.Types.ObjectId, ref: 'Billing'},
    checkIn: {type: Number},
    checkOut: {type: Number},
    noOfGuests: {type: Number},
    isAccepted: {type: Boolean},
    createdDate: {type: Number}

});

var Trip = mongoose.model('Trip', tripSchema);
module.exports = Trip;
```


db.getCollection('trips').find({})
Airbnb
ds035796.mlab.com:35796
airbnb17
db.getCollection('trips').find({})
trips
0.087 sec.
0
50
Key
Value
Type
(1) ObjectId("584079dce337c03b882355c7")
{ 12 fields }
Object
_id
ObjectId("584079dce337c03b882355c7")
ObjectId
createdDate
1480620508122.0
Double
isAccepted
true
Boolean
noOfGuests
2
Int32
checkOut
1481011200000.0
Double
checkIn
1480665600000.0
Double
hostId
ObjectId("583d4d177828aa2f286c396d")
ObjectId
userId
ObjectId("583f6dae3141c14bf8941ec2")
ObjectId
propertyId
ObjectId("583e8f508683e11820c158b4")
ObjectId
tripId
471-09-4294
String
__v
0
Int32
billingId
ObjectId("58407a08e337c03b882355c8")
ObjectId
(2) ObjectId("5840a35a633c172848320b2f")
{ 12 fields }
Object
_id
ObjectId("5840a35a633c172848320b2f")
ObjectId
createdDate
1480631130523.0
Double
isAccepted
true
Boolean
noOfGuests
1
Int32
checkOut
1456819200000.0
Double
checkIn
1441090800000.0
Double
hostId
ObjectId("583d4d177828aa2f286c396d")
ObjectId
userId
ObjectId("583d4d177828aa2f286c396d")
ObjectId
propertyId
ObjectId("583d515dd3010925608b2c4e")
ObjectId
tripId
155-18-6060
String
__v
0
Int32
billingId
ObjectId("5841e9ca8f4e62103c23bb48")
ObjectId
(3) ObjectId("5840a35c633c172848320b30")
{ 12 fields }
Object

- **Mongo Session Management**

```
app.use(session({
    secret: 'keyboard cat',
    resave: false,
    saveUninitialized: true,
    duration: 30 * 60 * 1000,
    activeDuration: 5 * 60 * 1000,
    store: new mongoStore({
        url: "mongodb://airbnb:airbnb@ds035796.mlab.com:35796/airbnb17"
    })
}));
```



- **Connection Pooling**
- Mongo Connection Pooling

mongoconnect.js

```
var mongoose = require('mongoose');
var gracefulShutdown;
var dbURI = 'mongodb://airbnb:airbnb@ds035796.mlab.com:35796/airbnb17';
/*var dbURI = 'mongodb://localhost:27017/airbnb';*/
if (process.env.NODE_ENV === 'production') {
    dbURI = process.env.MONGOLAB_URI;
}
mongoose.connect(dbURI, {server: {poolSize: 10}});
// CONNECTION EVENTS
mongoose.connection.on('connected', function () {
    console.log('Mongoose connected to- ' + dbURI);
});
mongoose.connection.on('error', function (err) {
    console.log('Mongoose connection error: ' + err);
});


mongoose.connection.on('disconnected', function () {
    console.log('Mongoose disconnected');
});
// CAPTURE APP TERMINATION / RESTART EVENTS
// To be called when process is restarted or terminated
gracefulShutdown = function (msg, callback) {
    mongoose.connection.close(function () {
        console.log('Mongoose disconnected through ' + msg);
        callback();
    });
};
// For nodemon restarts
process.once('SIGUSR2', function () {
    gracefulShutdown('nodemon restart', function () {
        process.kill(process.pid, 'SIGUSR2');
    });
```

Cache management using Redis

We can use redis to drastically improve performance of the application.

For this application, we implement a caching system to cache the most viewed properties. Everytime a property is requested from the user, it is stored in a hashmap data structure using redis. Next time if the same property is requested, the application doesn't need to query the database and can be returned instantly.

We also set up a expiry time to make sure the cache is not old enough to return old data.

We can see improved performance using this method.

```
var redis = require('redis');
var client = redis.createClient();
var logger=require('../routes/usertracking');

client.on('ready', function () {

    console.log("Redis Ready");

});

client.on('error', function (err) {
    console.log("Error " + err);
});

```

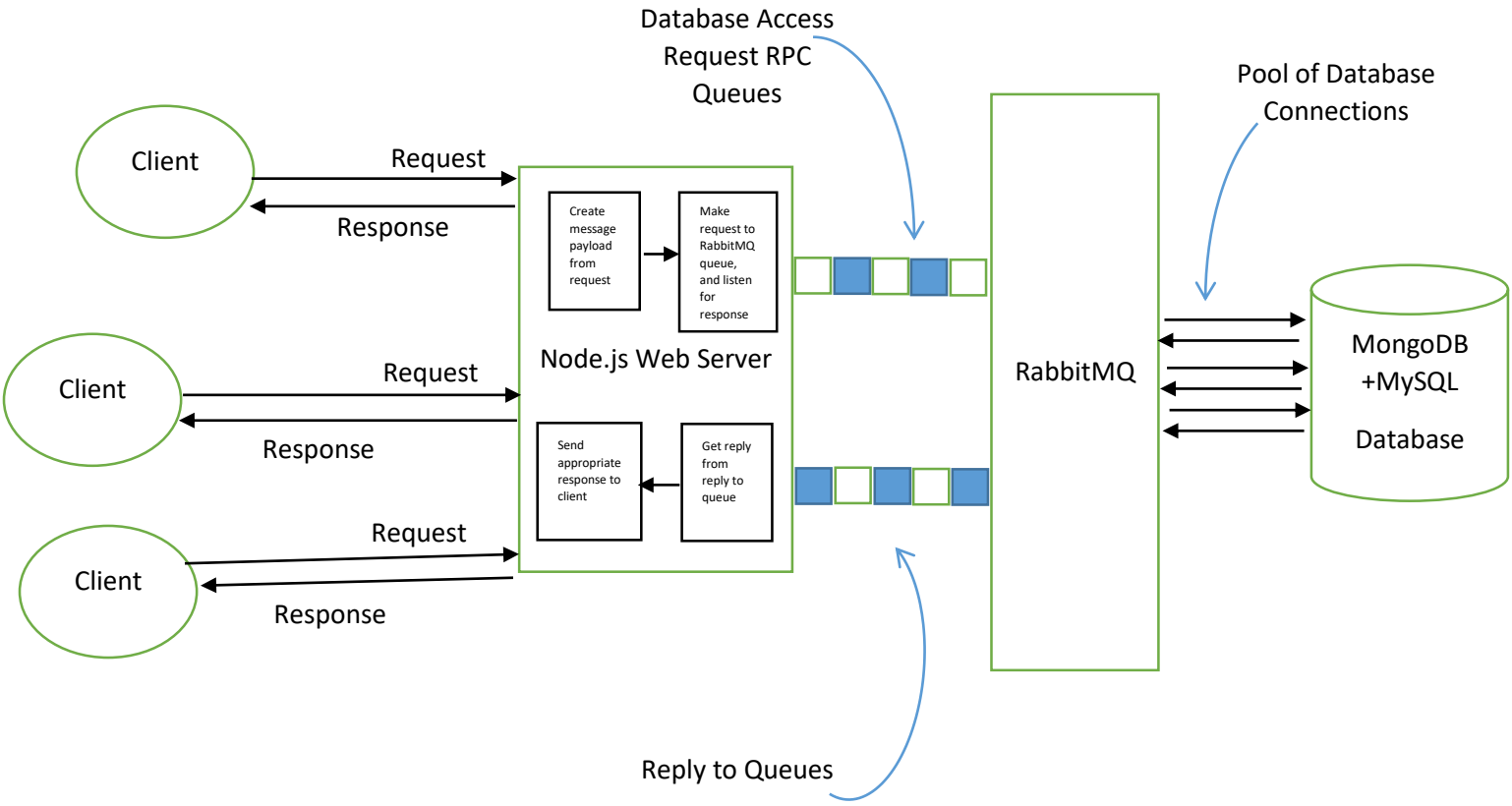
```
    client.hget("properties", id, function (err, result) {

        if (result) {
            console.log("From redis");
            obj=JSON.parse(result);
            logger.info(req.session.firstName+" clicked on "+obj.id,{'user':req.session.firstName,'property_clicked':obj.id,'
            res.end(result);
        }
        else {
            getProperty(req, res);
        }
    });
}
else {
    getProperty(req, res);
}
};
```

```
function getProperty (req, res) {
    var id = req.param("propertyId");

    var msg_payload = {
        id: id
    };

    mq_client.make_request('property_detail_queue', msg_payload, function (err, result) {
        if (err) {
            console.log("From mongoose");
            console.log(err);
            var json_responses = {"statusCode": 401};
            res.send(json_responses);
        } else {
            console.log(result.id);
            console.log("From mongoose");
            client.hmset("properties", result.id, JSON.stringify(result), redis.print);
            // var json_responses = {"statusCode": 200, "data": result};
            logger.info(req.session.firstName+" clicked on "+result.id,{'user':req.session.firstNa

            res.send(result);
            res.end();
```

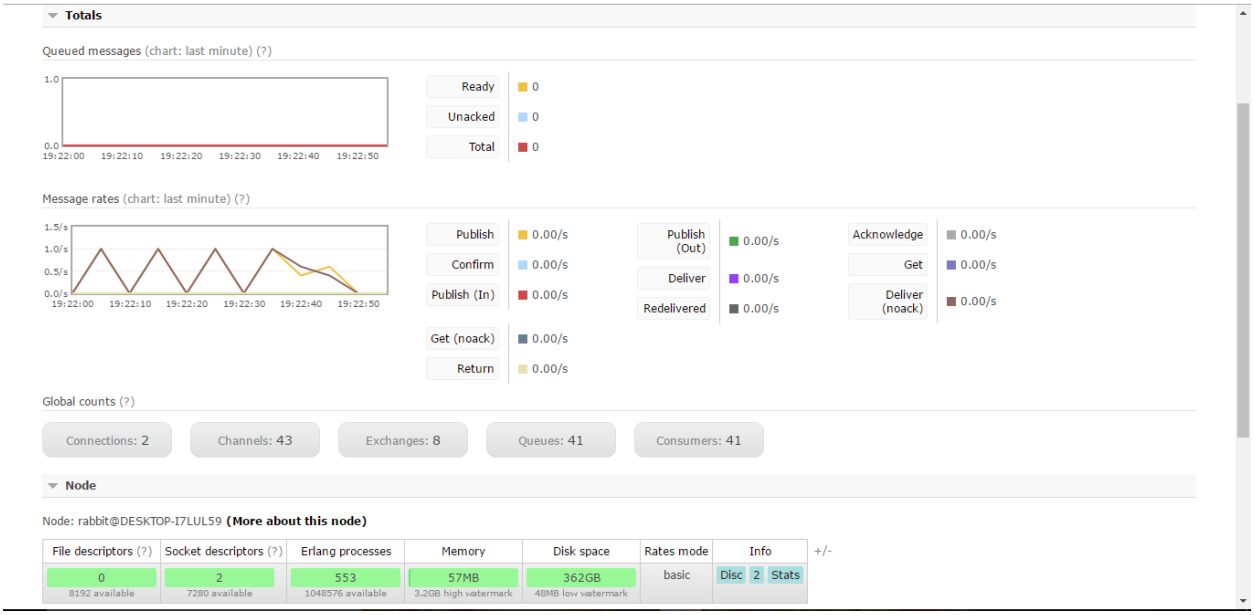
- **Scalability and Robustness**

  The application has been made scalable with the help of RabbitMQ to handle multiple concurrent users. It is possible because we have used different queues for different requests.

- **RabbitMQ Architecture**



- **RabbitMQ Management Console**



- **RabbitMQ Queues**

Overview			Messages			Message rates		
Name	Features	**State**	**Ready**	**Unacked**	**Total**	**incoming**	**deliver / get**	**ack**
acceptTrip_queue	AD	idle	0	0	0			
addHostReview_queue	AD	idle	0	0	0			
addNewListing_queue	AD	idle	0	0	0			
addUserReview_queue	AD	idle	0	0	0			
amq.gen-PNw_LUjU8pcMer0_gCW76Q	Excl AD	idle	0	0	0	0.00/s	0.00/s	0.00/s
amq.gen-YCbTYDoO3JpmOtKYf5M1Jw	Excl AD	idle	0	0	0	0.00/s	0.60/s	0.00/s
bidCron_queue	AD	running	0	0	0	0.00/s	0.00/s	0.00/s
cardDetail_queue	AD	idle	0	0	0			
checkHost_queue	AD	idle	0	0	0			
confirmBooking_queue	AD	idle	0	0	0			
dynamicPricingCron_queue	AD	running	0	0	0	0.00/s	0.00/s	0.00/s
editProperty_queue	AD	idle	0	0	0			
editUser_queue	AD	idle	0	0	0			
getActiveListings_queue	AD	idle	0	0	0			
getDatainEditProperty_queue	AD	idle	0	0	0			
getHostReview_queue	AD	idle	0	0	0			
getItinerary_queue	AD	idle	0	0	0			
getPropertyDetails_queue	AD	idle	0	0	0			
getReservations_queue	AD	idle	0	0	0			
getUserProfile_queue	AD	idle	0	0	0			
getUserReview_queue	AD	idle	0	0	0			
getUserTrips_queue	AD	idle	0	0	0			
hostReviewsCount_queue	AD	idle	0	0	0			

```
cnn.queue('login_queue', function (q) {...});

cnn.queue('register_queue', function (q) {...});
cnn.queue('search_queue', function (q) {...});
cnn.queue('property_detail_queue', function (q) {...});
cnn.queue('editUser_queue', function (q) {...});

cnn.queue('loadEditUser_queue', function (q) {...});

cnn.queue('loadReviewAboutPage_queue', function (q) {...});

cnn.queue('loadReviewByPage_queue', function (q) {...});
cnn.queue('hostReviewsCount_queue', function (q) {...});

cnn.queue('uploadProfileImage_queue', function (q) {...});

cnn.queue('loadProfilePhotoPage_queue', function (q) {...});
cnn.queue('updatePassword_queue', function (q) {...});
cnn.queue('updatePaymentMethod_queue', function (q) {...});
cnn.queue('payinTransaction_queue', function (q) {...});
```

```
    cnn.queue('updatePaymentMethod_queue', function (q) {...});
    cnn.queue('payinTransaction_queue', function (q) {...});
    cnn.queue('payoutTransactions_queue', function (q) {...});
    cnn.queue('receiptPage_queue', function (q) {...});
    cnn.queue('loadPaymentPage_queue', function (q) {...});
    cnn.queue('getPropertyDetails_queue', function (q) {...});
    cnn.queue('confirmBooking_queue', function (q) {...});
    cnn.queue('getUserTrips_queue', function (q) {...});
    cnn.queue('getActiveListings_queue', function (q) {...});
    cnn.queue('addNewListing_queue', function (q) {...});
    cnn.queue('getReservations_queue', function (q) {...});
    cnn.queue('acceptTrip_queue', function (q) {...});
    cnn.queue('getItinerary_queue', function (q) {...});
    cnn.queue('getUserProfile_queue', function (q) {...});
    cnn.queue('getUserReview_queue', function (q) {...});
    cnn.queue('getHostReview_queue', function (q) {...});
    cnn.queue('addUserReview_queue', function (q) {...});
    cnn.queue('addHostReview_queue', function (q) {...});
    cnn.queue('updateUserToHost_queue', function (q) {...});
    cnn.queue('editProperty_queue', function (q) {...});
    cnn.queue('getDatainEditProperty_queue', function (q) {...});
    cnn.queue('checkHost_queue', function (q) {...});
    cnn.queue('cardDetail_queue', function (q) {...});
    cnn.queue('updateBasePrice_queue', function (q) {...});
    cnn.queue('bidCron_queue', function (q) {...});
    cnn.queue('dynamicPricingCron_queue', function (q) {...});
    cnn.queue('removeListing_queue', function (q) {...});
});
```

- **Bcrypt Encryption**

```
93  exports.registerUser = function (req, res) {
94      var f_name = req.body.first_name;
95      var l_name = req.body.last_name;
96      var email_id = req.body.email_id;
97      var pwd = req.body.password;
98      var sess = req.session;
99      sess.isLoggedIn = false;
100     var salt = bcrypt.genSaltSync(10);
101     var passwordToSave = bcrypt.hashSync(pwd, salt);
102     var msg_payload = {
103         firstName: f_name,
104         lastName: l_name,
105         email_id: email_id,
106         password: passwordToSave
107     };
108
109     mq_client.make_request('register_queue', msg_payload, function (err, results) {
110         if (err) {
111             res.json({
112                 success: false,
113                 message: 'Error in registration.'
114             });
115             res.end();
116         }
117         if (results) {
118
119             sess.userSSN = results.userId;
120             sess.firstName = results.firstName;
121             sess.lastName = results.lastName;
122             sess.userId = results._id;
123             sess.email = results.email;
124             sess.isLoggedIn = true;
125             res.json({
126                 success: true,
```

- **Passport.js Authentication**

```
exports.authenticateUser = function (req, res, next) {

//  var email_id = req.body.email_id;
//  var pwd = req.body.password;
    var sess = req.session;
    sess.isLoggedIn = false;
    console.log("in signin");
    passport.authenticate('login', function (err, user) {
        console.log("Result" + user);
        if (err) {
            return next(err);
        }

        if (!user) {

            res.json({
                success: false,
                message: 'No user found'
            });
            res.end();
            /*return res.redirect('/signup');*/
        }

        if (user) {

            sess.email =user.email;
            sess.isLoggedIn = true;
            sess.userSSN=user.userId;
            sess.firstName=user.firstName;
            sess.lastName=user.lastName;
            sess.userId=user._id;
            sess.isHost = user.isHost;
            if(user.profileImage)
```

```
function isAuthenticated(req, res, next) {
    if (req.session.user_id) {
        console.log(req.session.user_id);
        return next();
    }
    res.redirect('/signinPg');
}
```

- **Dynamic Pricing Algorithm**

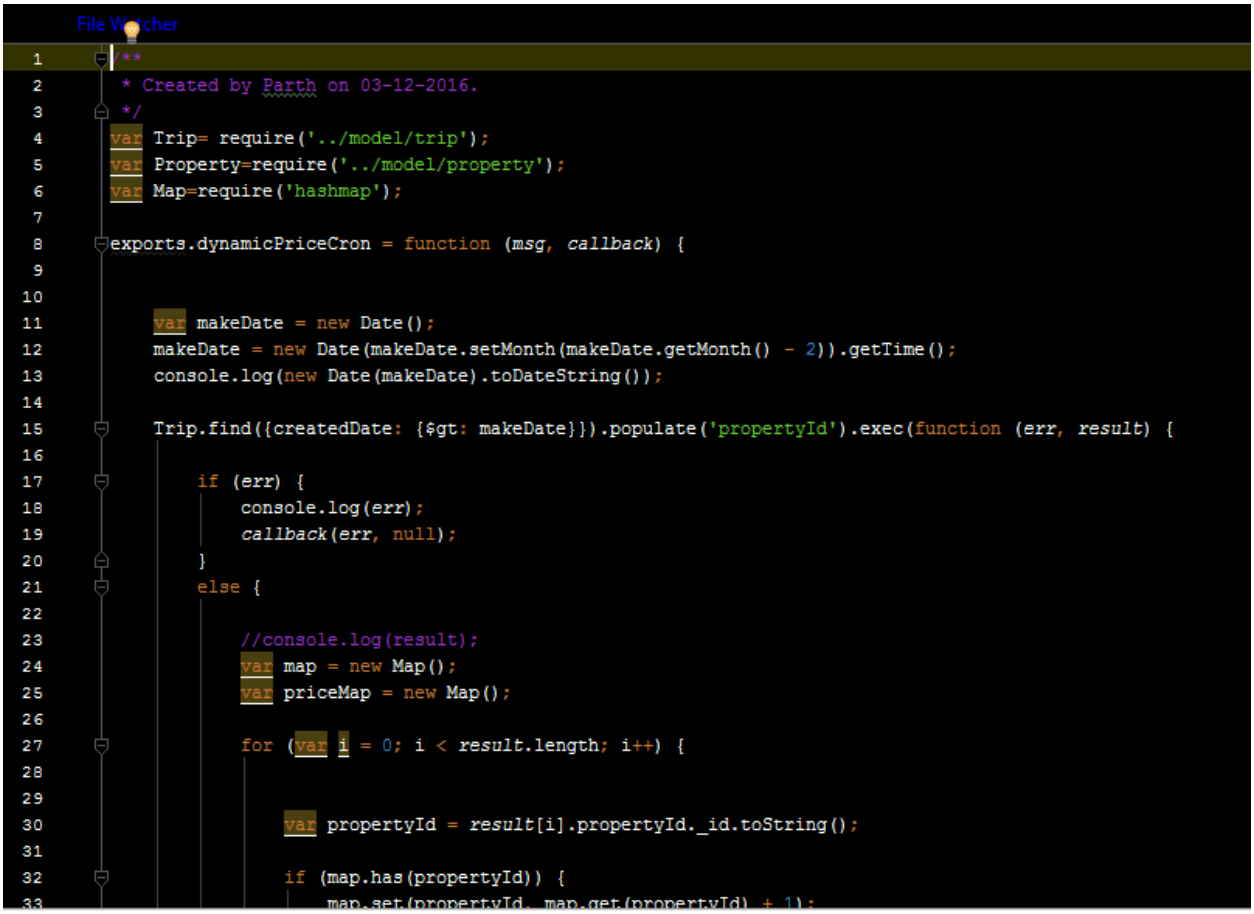
Explanation

We implemented a dynamic pricing algorithm to generate fair pricing for properties relative to the popularity of the property among the user.

We implement it by using the data of popularity of the property. We can measure popularity by finding number of trips booked in the past two months. We can use this data to generate future demand of the property.

Using the popularity, we generate a multiplier which we can assign to the property. The multiplier ranges from 1 to 2 in steps of 0.2-0.3 depending on the popularity.

The multiplier is clearly displayed while booking the property as well as in the receipts.



```
File Watcher
1   /**
2    * Created by Parth on 03-12-2016.
3    */
4   var Trip= require('../model/trip');
5   var Property=require('../model/property');
6   var Map=require('hashmap');
7
8   exports.dynamicPriceCron = function (msg, callback) {
9
10
11      var makeDate = new Date();
12      makeDate = new Date(makeDate.setMonth(makeDate.getMonth() - 2)).getTime();
13      console.log(new Date(makeDate).toDateString());
14
15      Trip.find({createdDate: {$gt: makeDate}}).populate('propertyId').exec(function (err, result) {
16
17          if (err) {
18              console.log(err);
19              callback(err, null);
20          }
21          else {
22
23              //console.log(result);
24              var map = new Map();
25              var priceMap = new Map();
26
27              for (var i = 0; i < result.length; i++) {
28
29
30                  var propertyId = result[i].propertyId._id.toString();
31
32                  if (map.has(propertyId)) {
33                      map.set(propertyId, map.get(propertyId) + 1);
```

File Watcher

```
28
29
30                  var propertyId = result[i].propertyId._id.toString();
31
32                  if (map.has(propertyId)) {
33                      map.set(propertyId, map.get(propertyId) + 1);
34                  }
35                  else {
36                      map.set(propertyId, 1);
37                      priceMap.set(propertyId, result[i].propertyId.price);
38                  }
39              }
40              //onsole.log(map);
41              var propertyIDArray = [];
42              var dynamicMultiplierArray = [];
43              //console.log("count " + map.count());
44              if (map.count() > 0) {
45
46                  map.forEach(function (value, key) {
47                      if (value >= 5 && value < 20) {
48
49                          propertyIDArray.push(key);
50                          var newMultiplier = 1.2;
51                          dynamicMultiplierArray.push(newMultiplier);
52                      }
53                      else if (value >= 20 && value < 40) {
54
55                          propertyIDArray.push(key);
56                          var newMultiplier = Number(priceMap.get(key)) * 1.5;
57                          dynamicMultiplierArray.push(newMultiplier);
58                      }
59                      else if (value >= 40 && value < 50) {
60                          propertyIDArray.push(key);
```

File Watcher

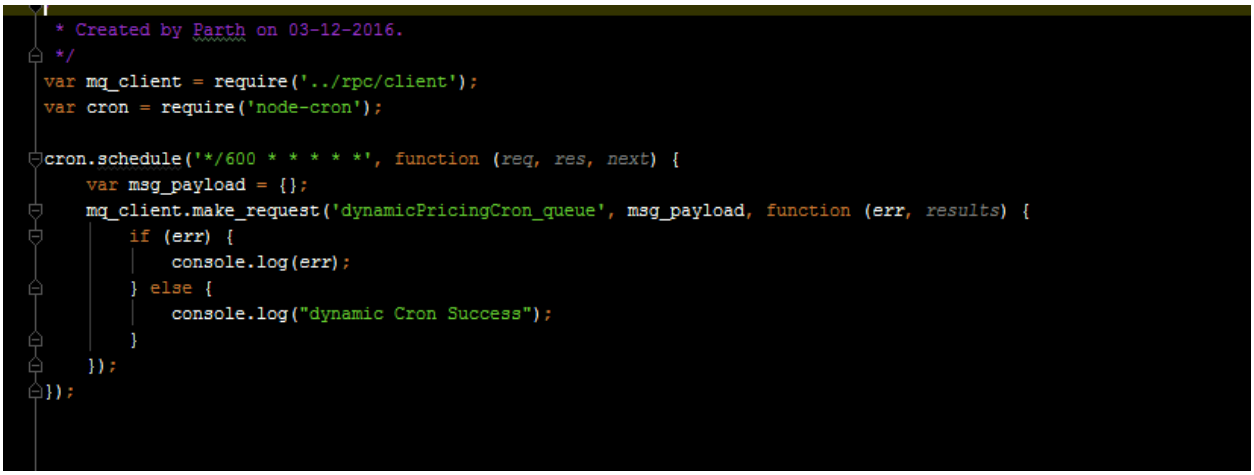
```
52                  }
53                  else if (value >= 20 && value < 40) {
54
55                      propertyIDArray.push(key);
56                      var newMultiplier = Number(priceMap.get(key)) * 1.5;
57                      dynamicMultiplierArray.push(newMultiplier);
58                  }
59                  else if (value >= 40 && value < 50) {
60                      propertyIDArray.push(key);
61                      var newMultiplier = Number(priceMap.get(key)) * 1.8;
62                      dynamicMultiplierArray.push(newMultiplier);
63                  }
64                  else if (value >= 50) {
65                      propertyIDArray.push(key);
66                      var newMultiplier = Number(priceMap.get(key)) * 2.0;
67                      dynamicMultiplierArray.push(newMultiplier);
68                  }
69              });
70          }
71          var bulk = Property.collection.initializeOrderedBulkOp();
72          // console.log(bulk);
73          for (var i = 0; i < propertyIDArray.length; i++) {
74
75              console.log("Property id " + propertyIDArray[i] + ": " + dynamicMultiplierArray[i]);
76              Property.update({_id: propertyIDArray[i]}, {
77                  $set: {
78                      multiplier: dynamicMultiplierArray[i]
79                  }
80              }, function (err, result) {
81                  if (err) {
82                      console.log(err);
83                      callback(err, null);
84                  }
```

File Watcher

```
75              console.log("Property id " + propertyIDArray[i] + ": " + dynamicMultiplierArray[i]);
76              Property.update({_id: propertyIDArray[i]}, {
77                  $set: {
78                      multiplier: dynamicMultiplierArray[i]
79                  }
80              }, function (err, result) {
81                  if (err) {
82                      console.log(err);
83                      callback(err, null);
84                  }
85                  if (!result) {
86                      console.log("not updated");
87                      console.log(result);
88                      callback(null, result);
89                  }
90                  if (result) {
91                      console.log("Updated");
92                      console.log(result);
93                      callback(null, result);
94                  }
95              });
96
97          }
98
99
100     }
101 });
102 };
```

Dynamic pricing cron

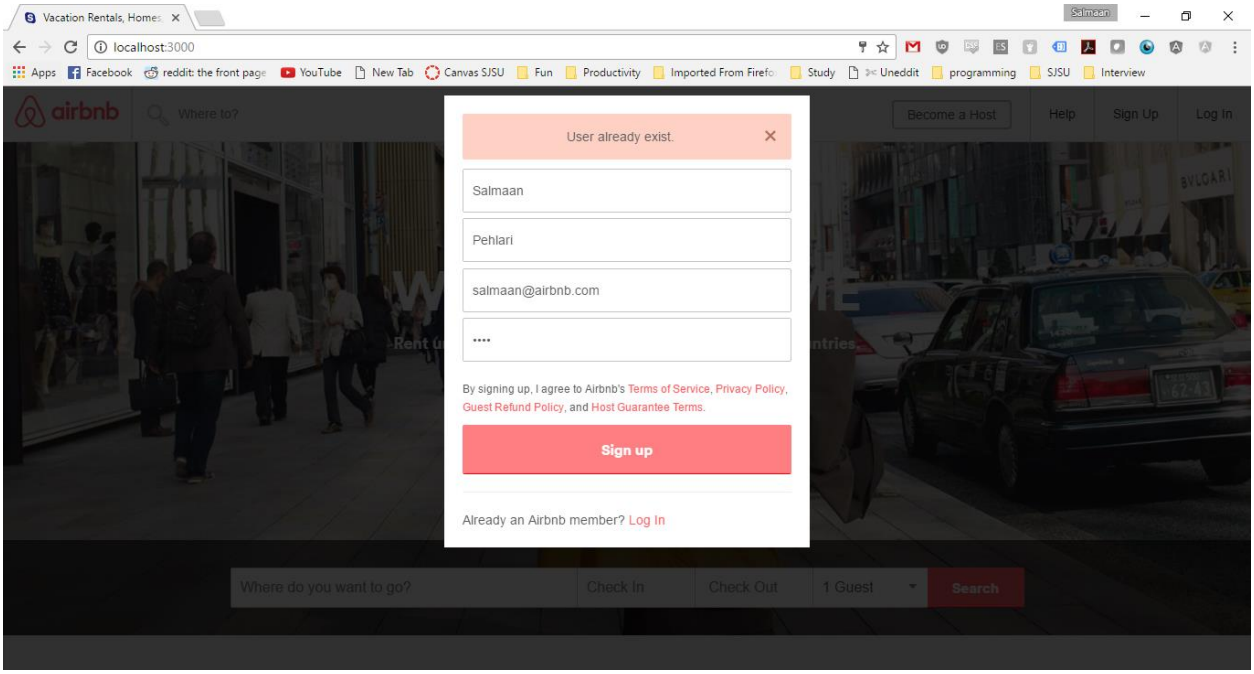
We implemented a cron job to generate and update the dynamic pricing of the properties. The cron checks the trip details for the properties and passes it to the dynamic pricing algorithm.

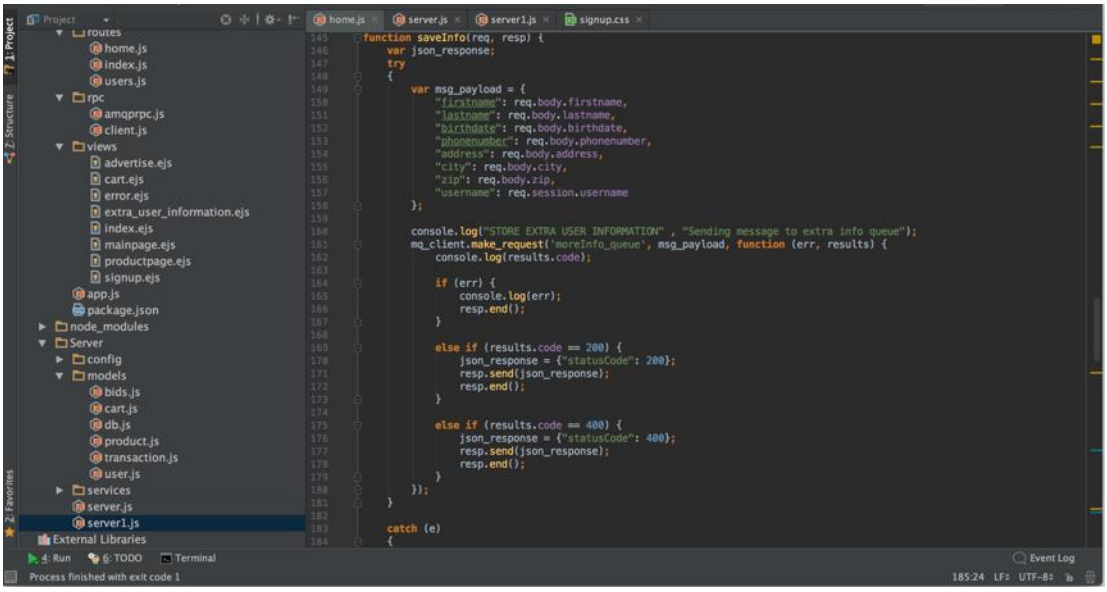


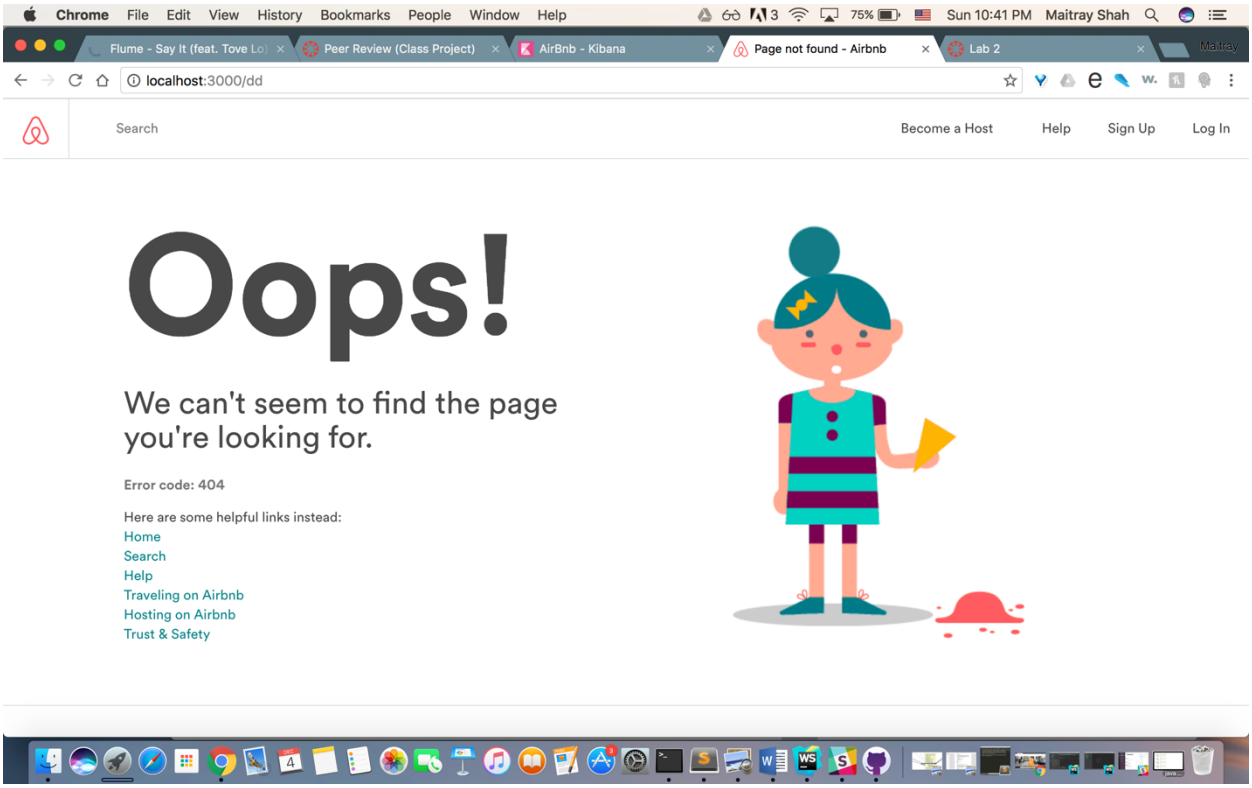
```
 * Created by Parth on 03-12-2016.
 */
var mq_client = require('../rpc/client');
var cron = require('node-cron');

cron.schedule('*/600 * * * * *', function (req, res, next) {
    var msg_payload = {};
    mq_client.make_request('dynamicPricingCron_queue', msg_payload, function (err, results) {
        if (err) {
            console.log(err);
        } else {
            console.log("dynamic Cron Success");
        }
    });
});
```

- **Exception Handling and Validation**

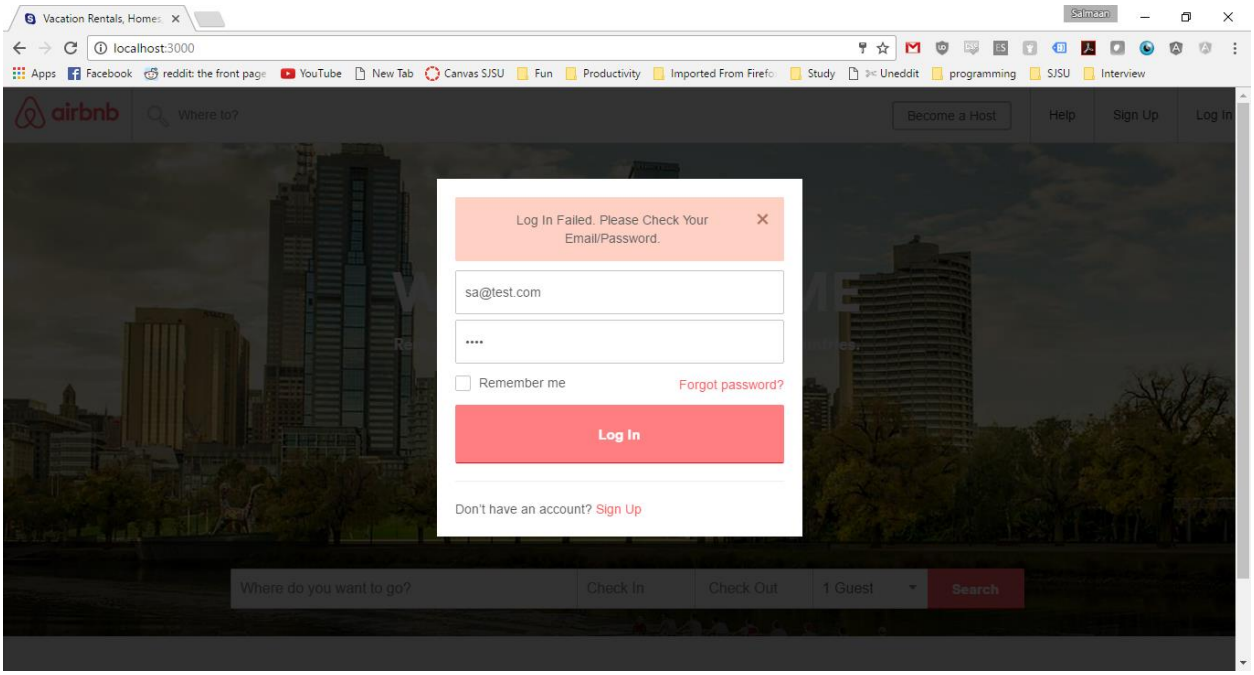
Validation in Registration - If user already exists then the following screen will appear



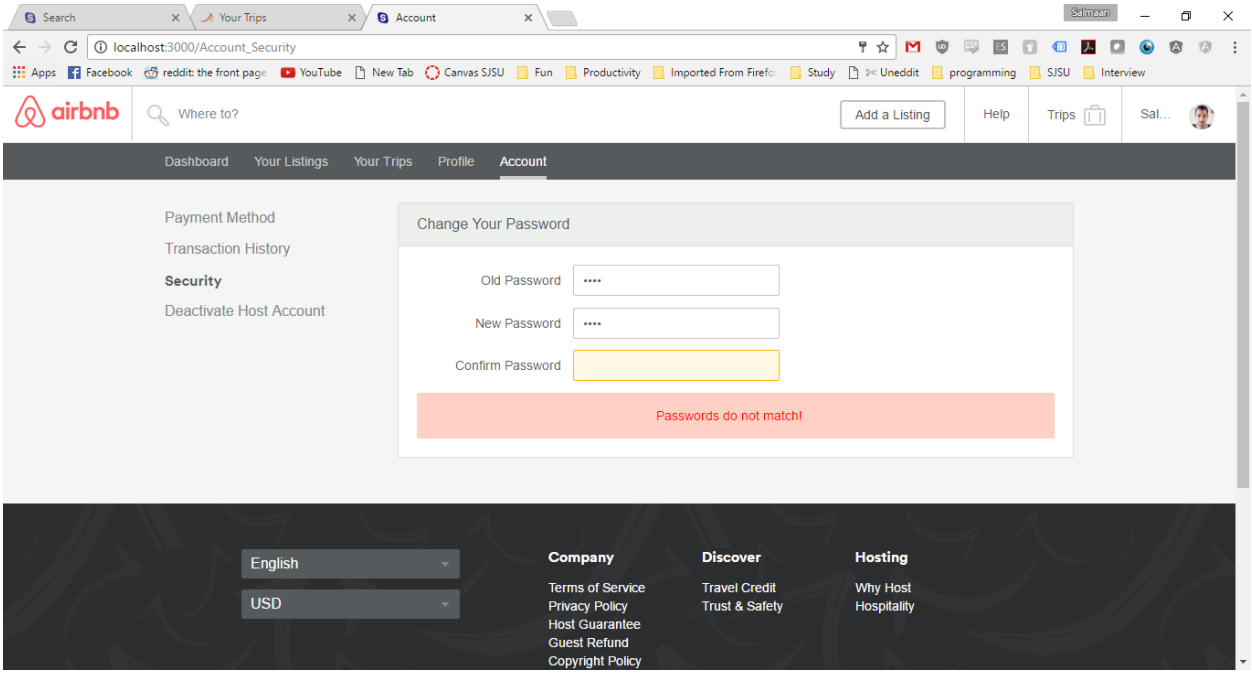




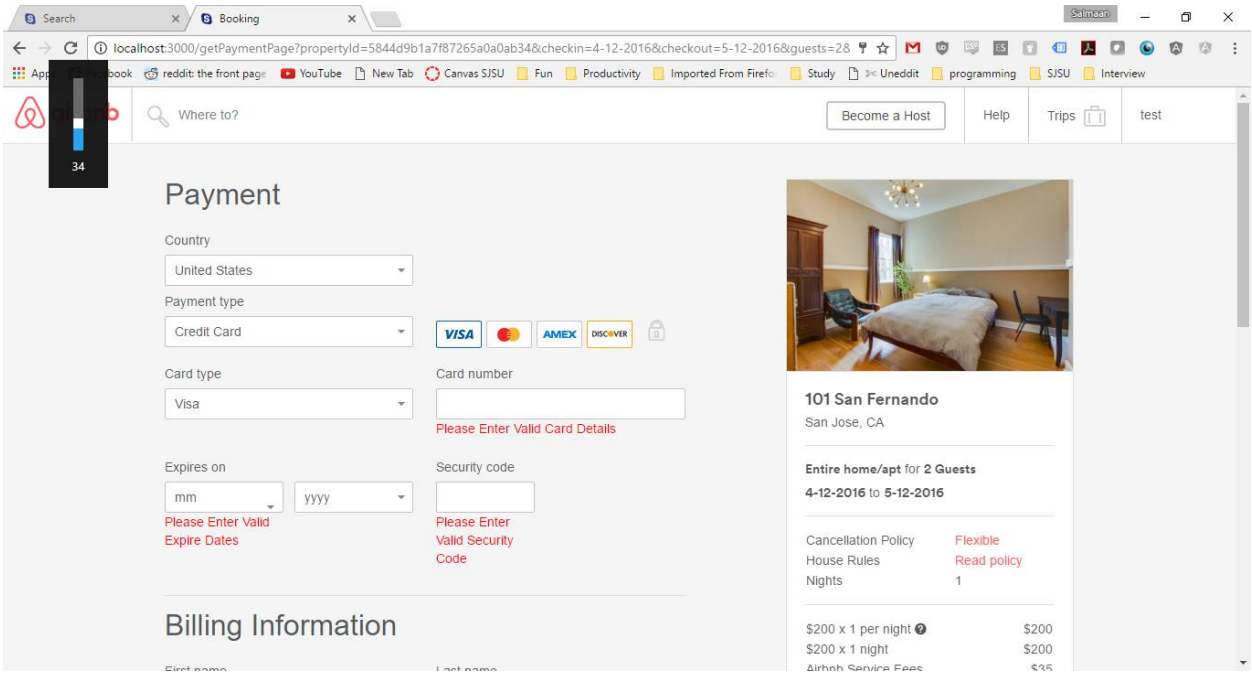
Validation in Login - If email and/or password are incorrect then the following screen will appear.

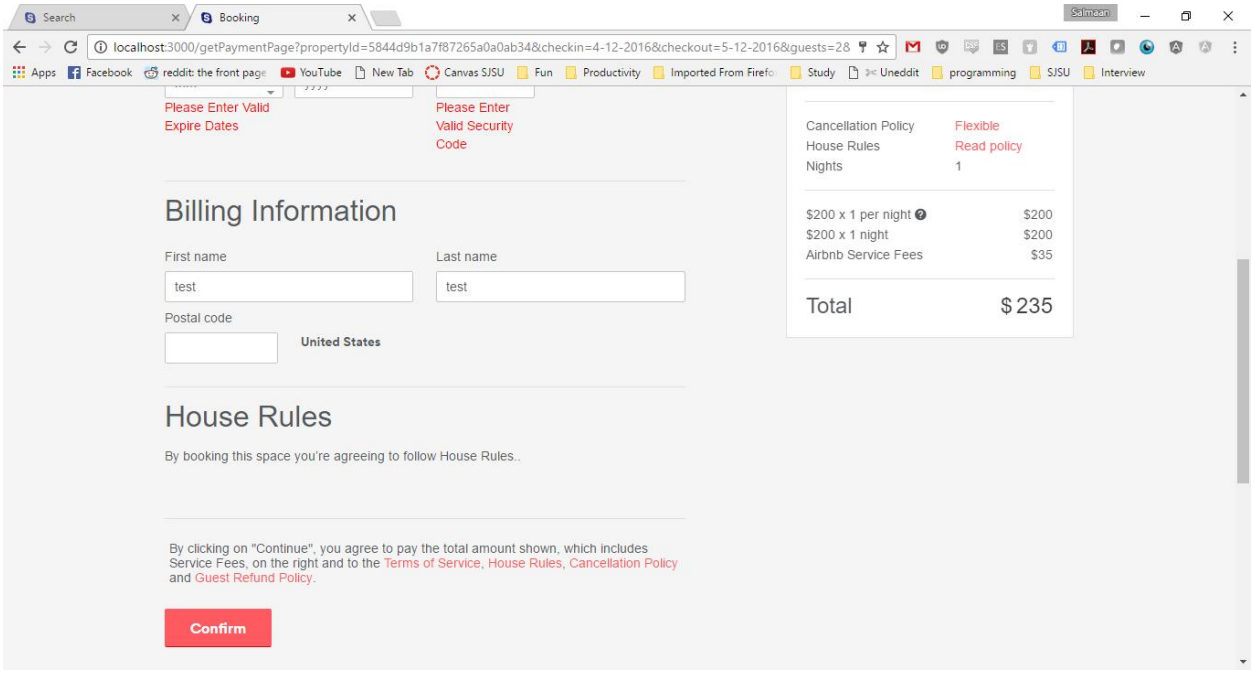


Validation in change password - If the new password and the confirm password do not match then the following screen will appear.

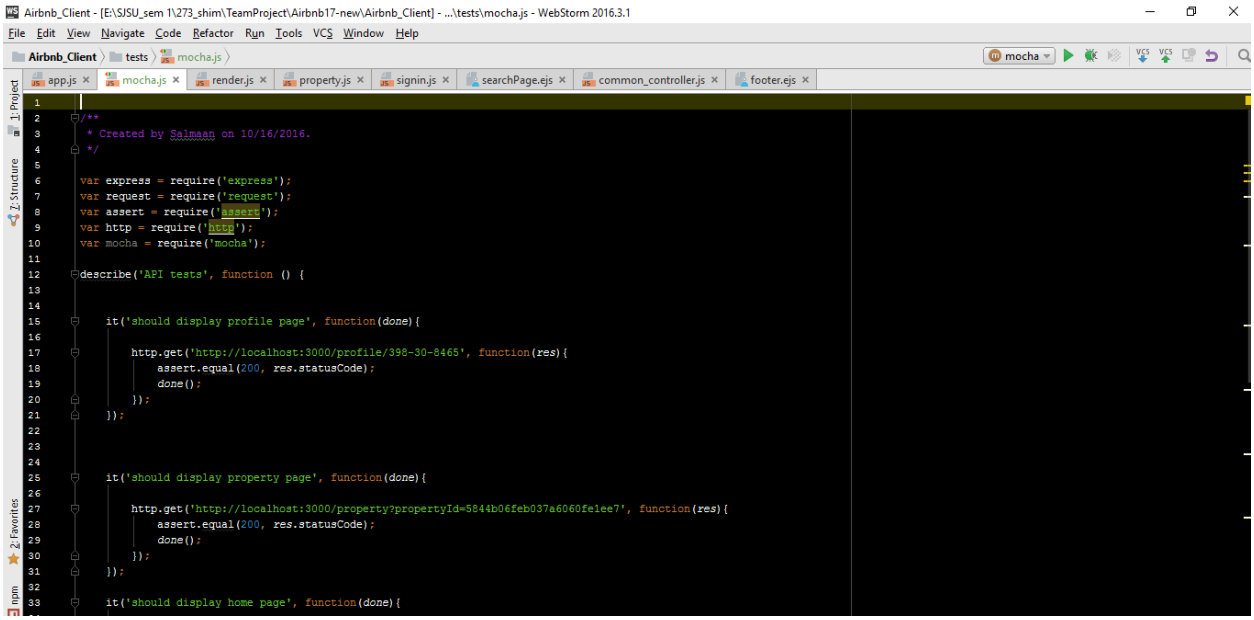


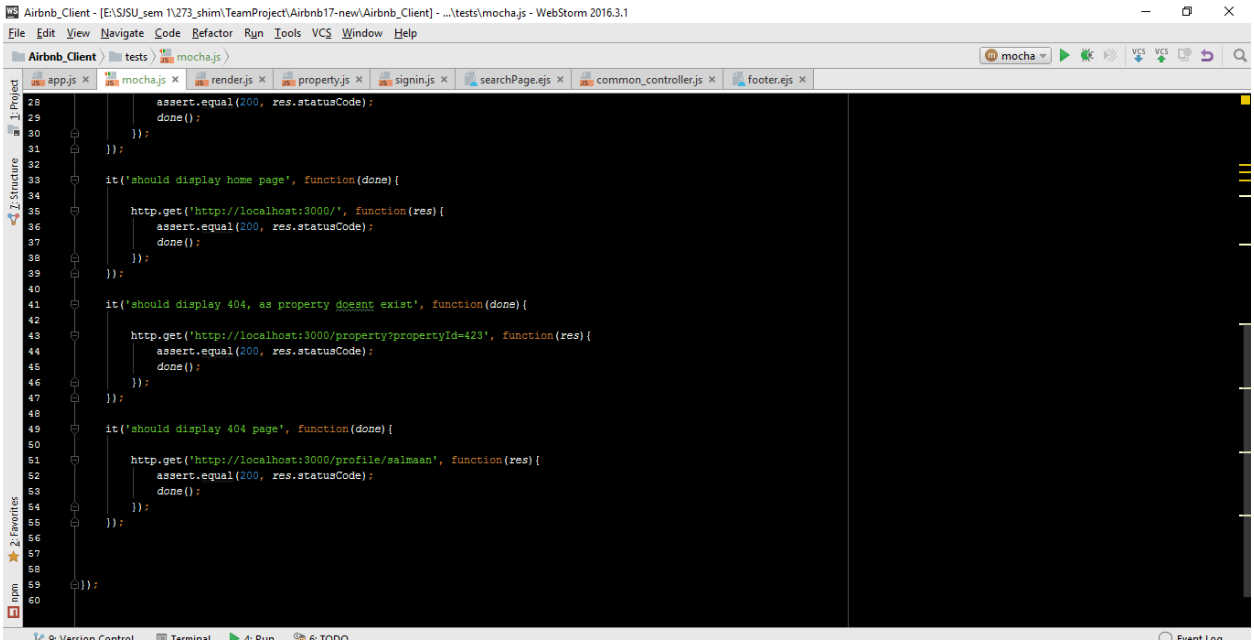
If any of the credit card details are not in valid format then the following screen will appear




Search
Booking
Balmeet
localhost:3000/getPaymentPage?propertyId=5844d9b1a7f87265a0a0ab34&checkin=4-12-2016&checkout=5-12-2016&guests=28
Apps
Facebook
reddit: the front page
YouTube
New Tab
Canvas SJSU
Fun
Productivity
Imported From Firefo
Study
Uneddit
programming
SJSU
Interview
Please Enter Valid Expire Dates
Please Enter Valid Security Code
Billing Information
First name
test
Last name
test
Postal code
United States
House Rules
By booking this space you're agreeing to follow House Rules..
By clicking on "Continue", you agree to pay the total amount shown, which includes Service Fees, on the right and to the Terms of Service, House Rules, Cancellation Policy and Guest Refund Policy.
Confirm
Cancellation Policy
Flexible
House Rules
Read policy
Nights
1
$200 x 1 per night
$200
$200 x 1 night
$200
Airbnb Service Fees
$35
Total
$ 235

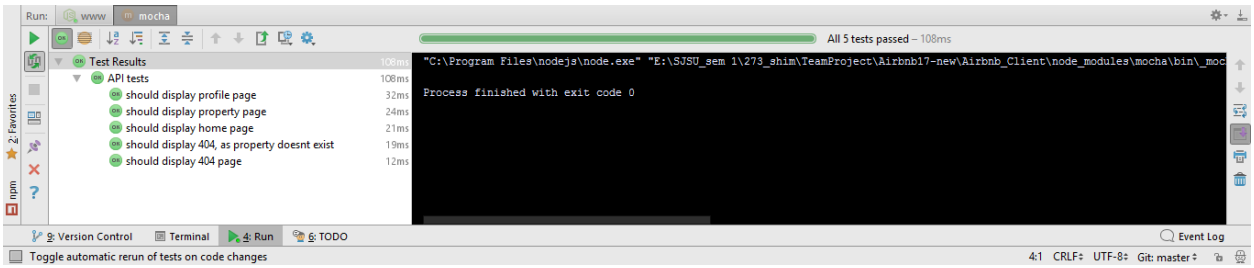
- **Mocha Testing:**





<u>Output</u>:

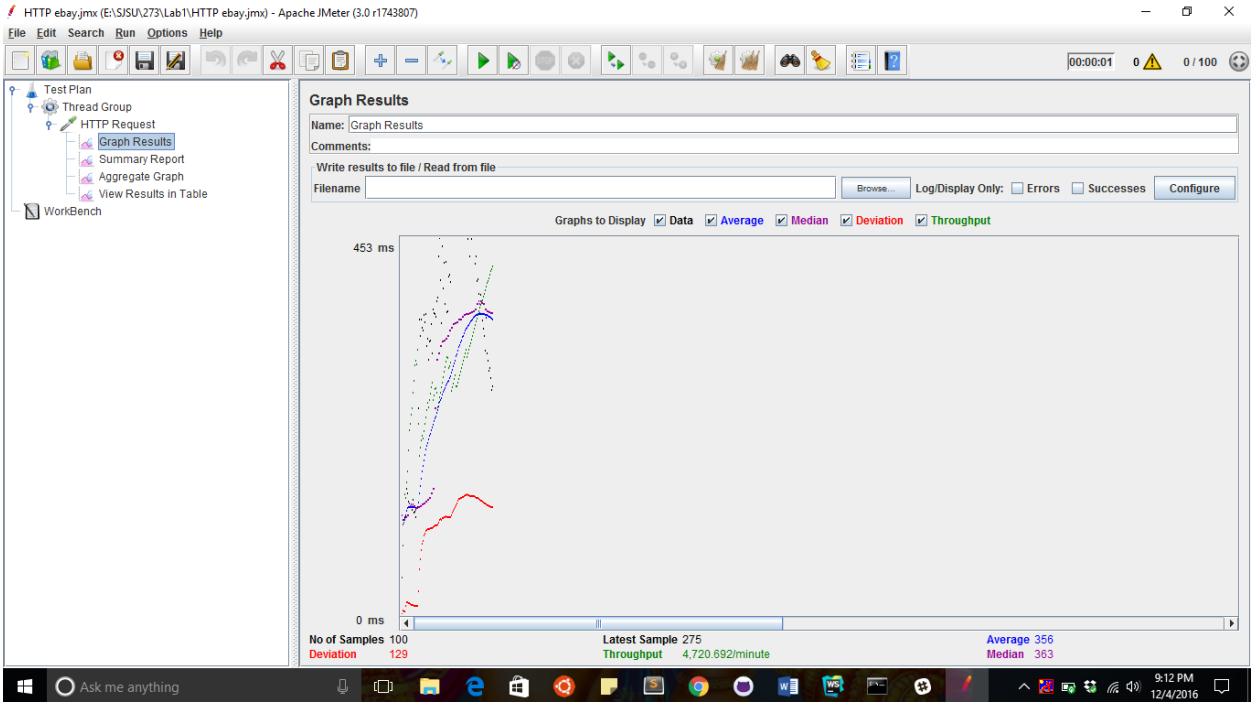
Mocha Testing screenshots

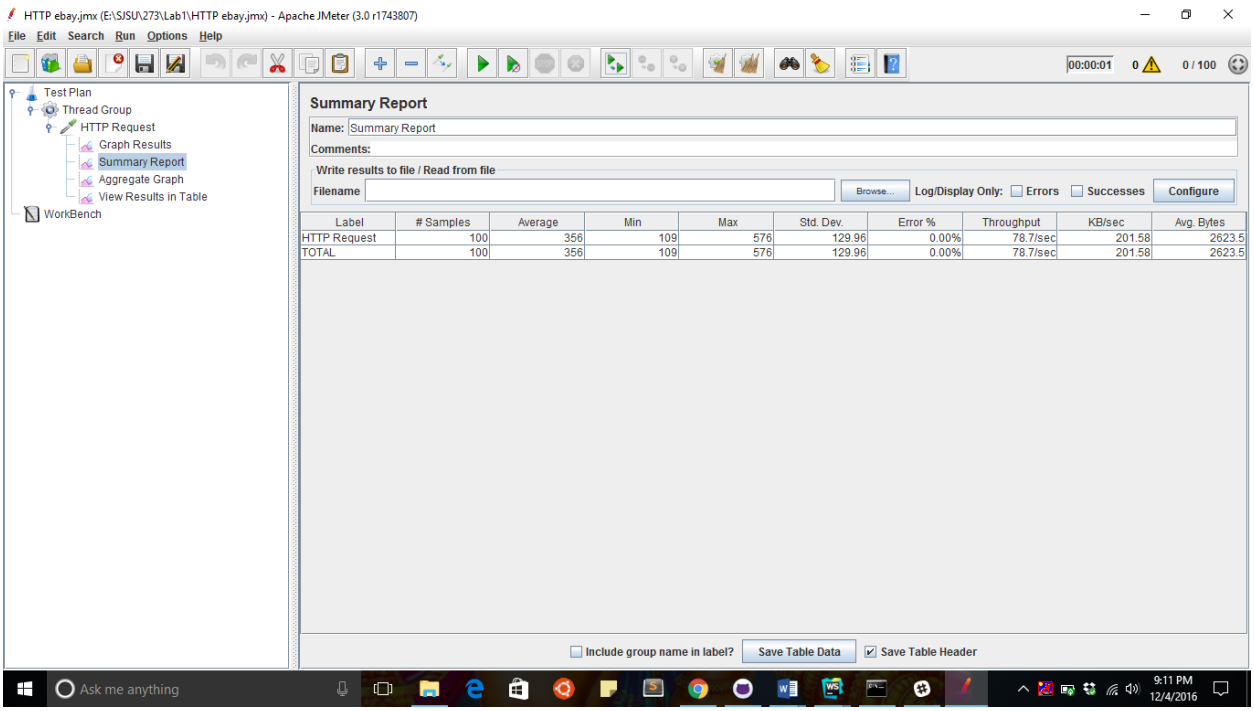


JMeter Testing:

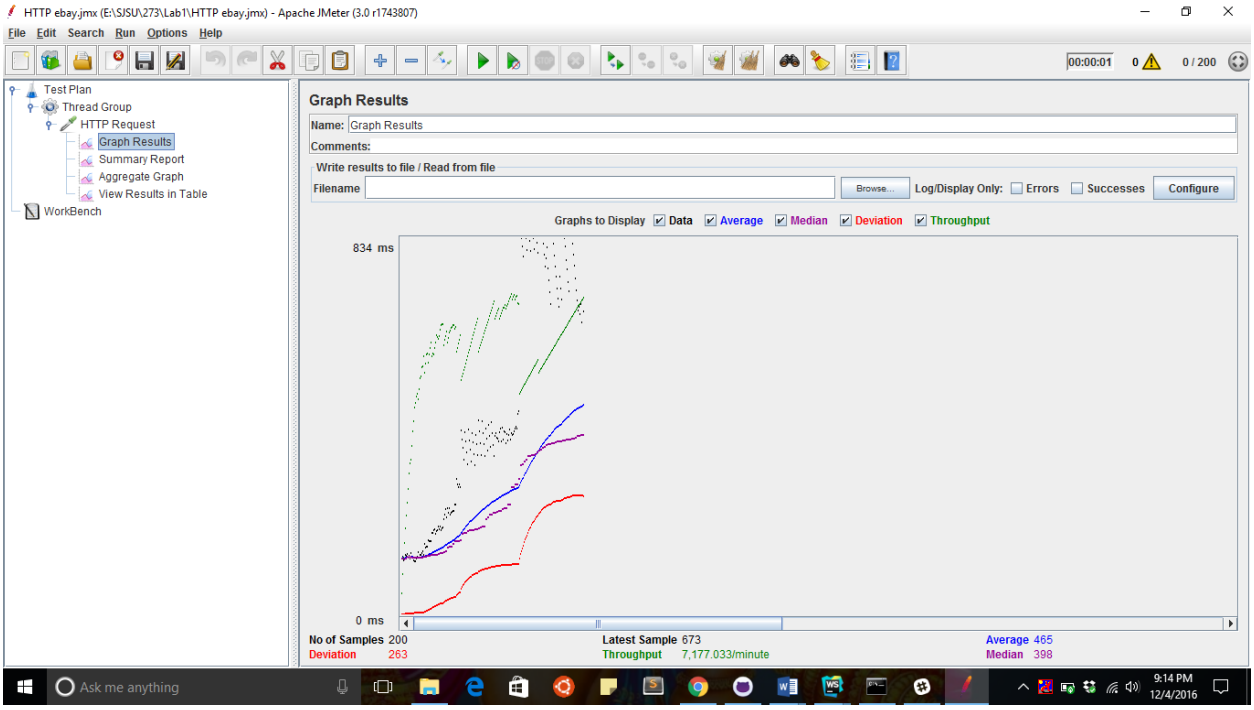
1.Base

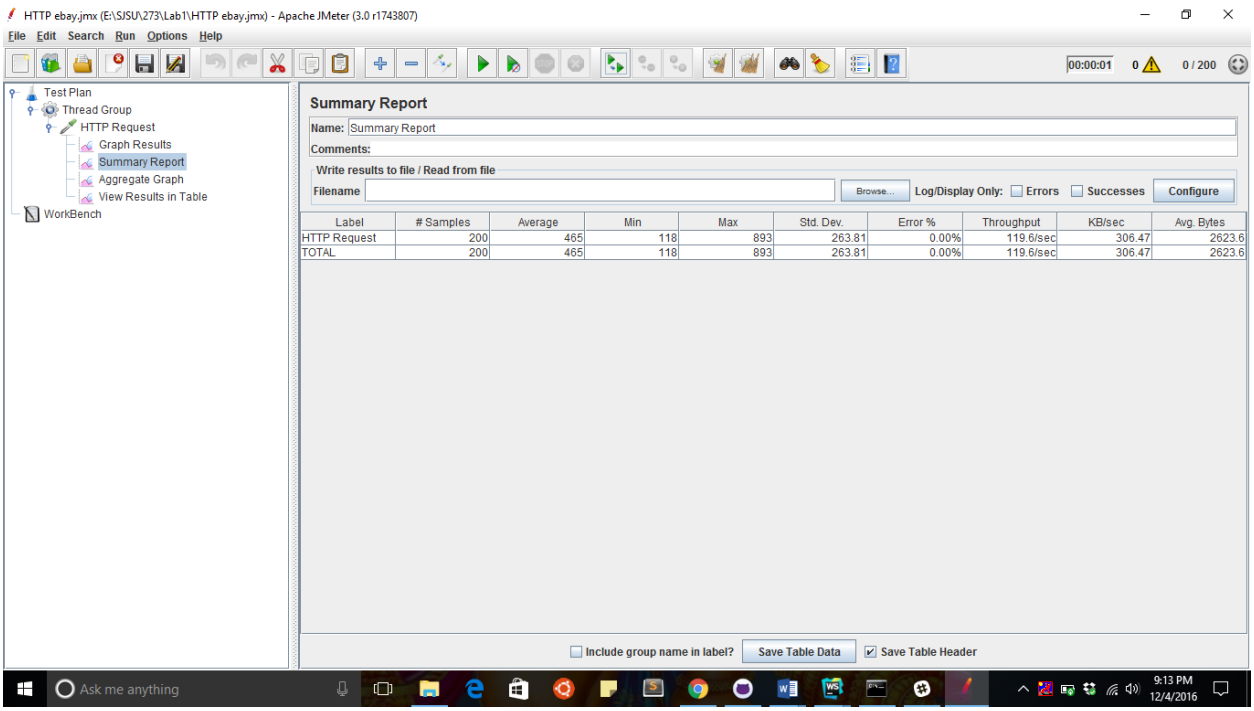
1)100 concurrent users



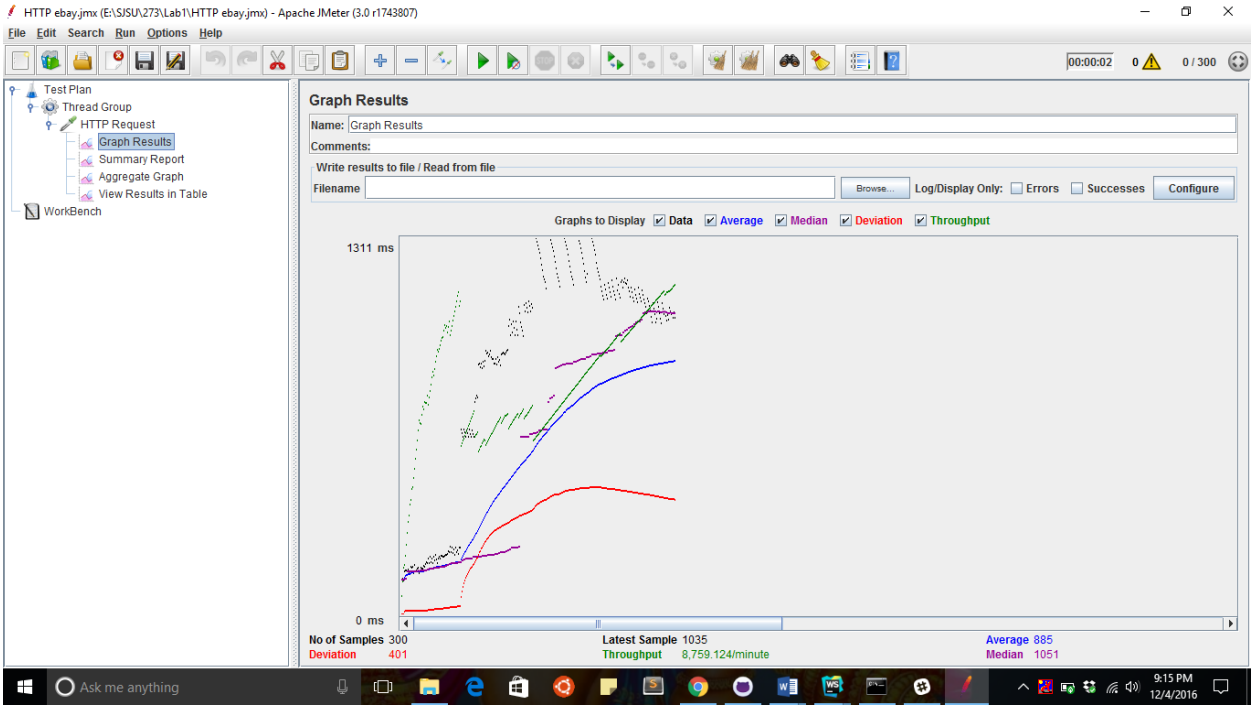


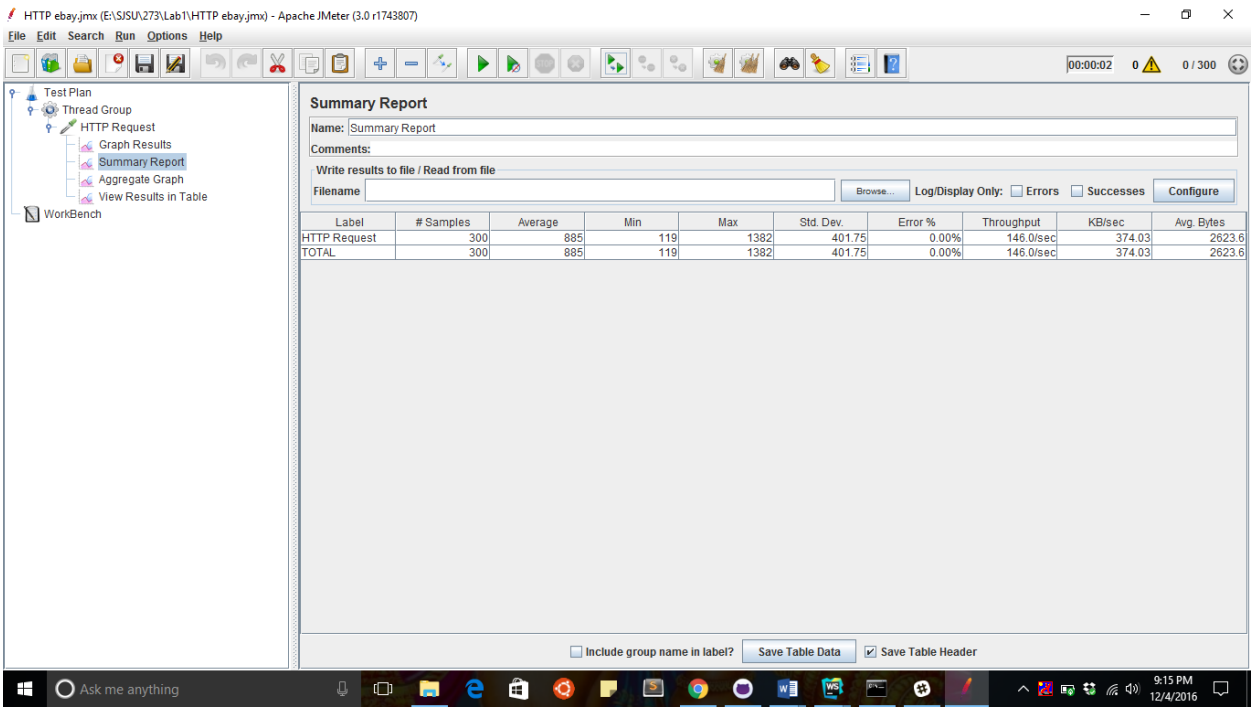
2)200 concurrent users



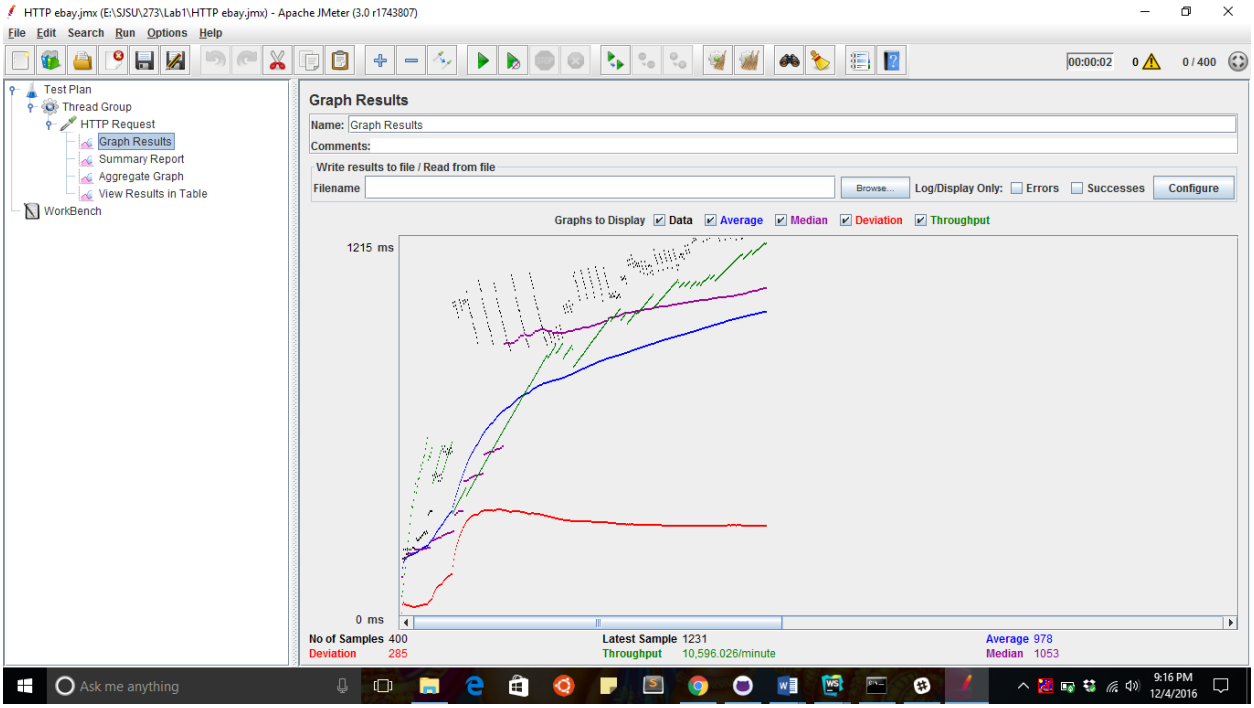


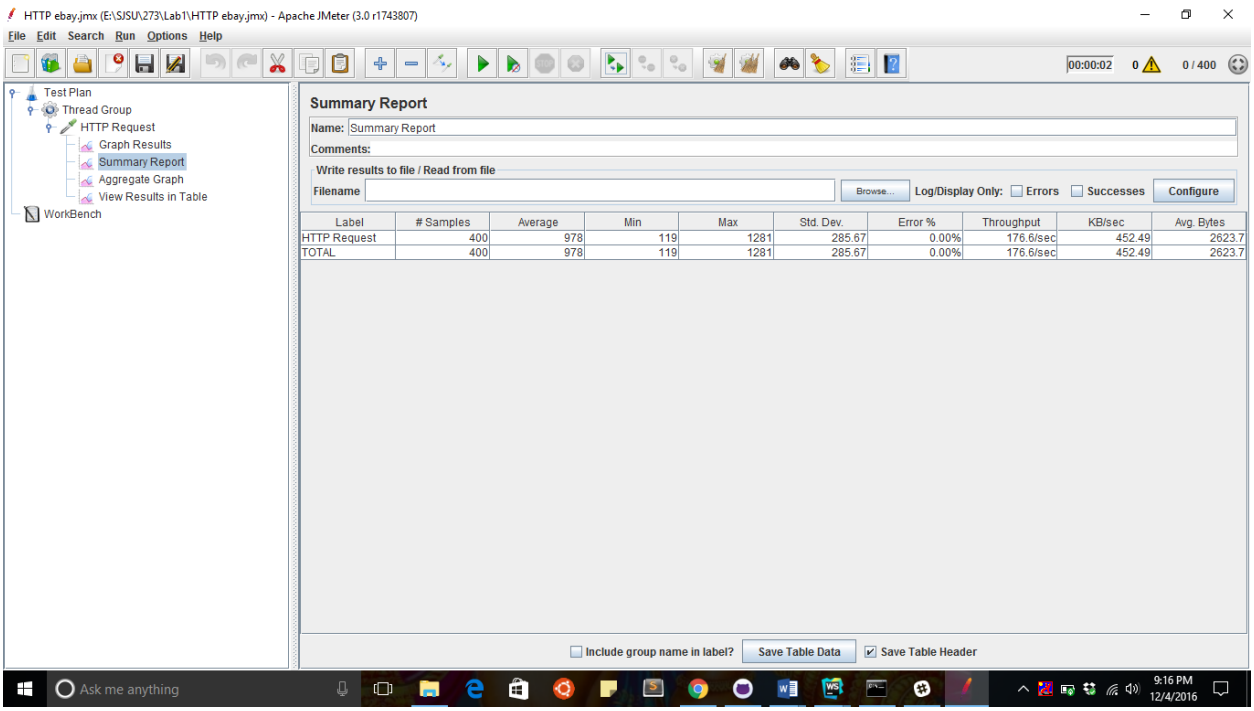
3)300 concurrent users



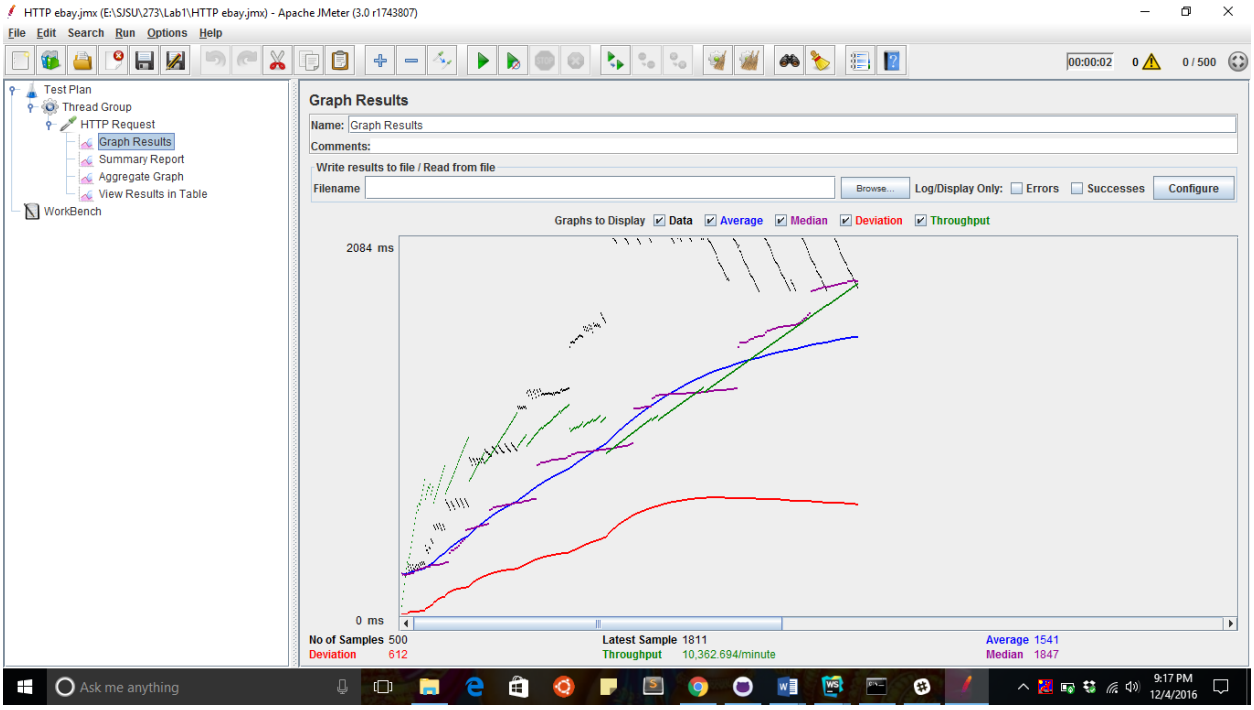


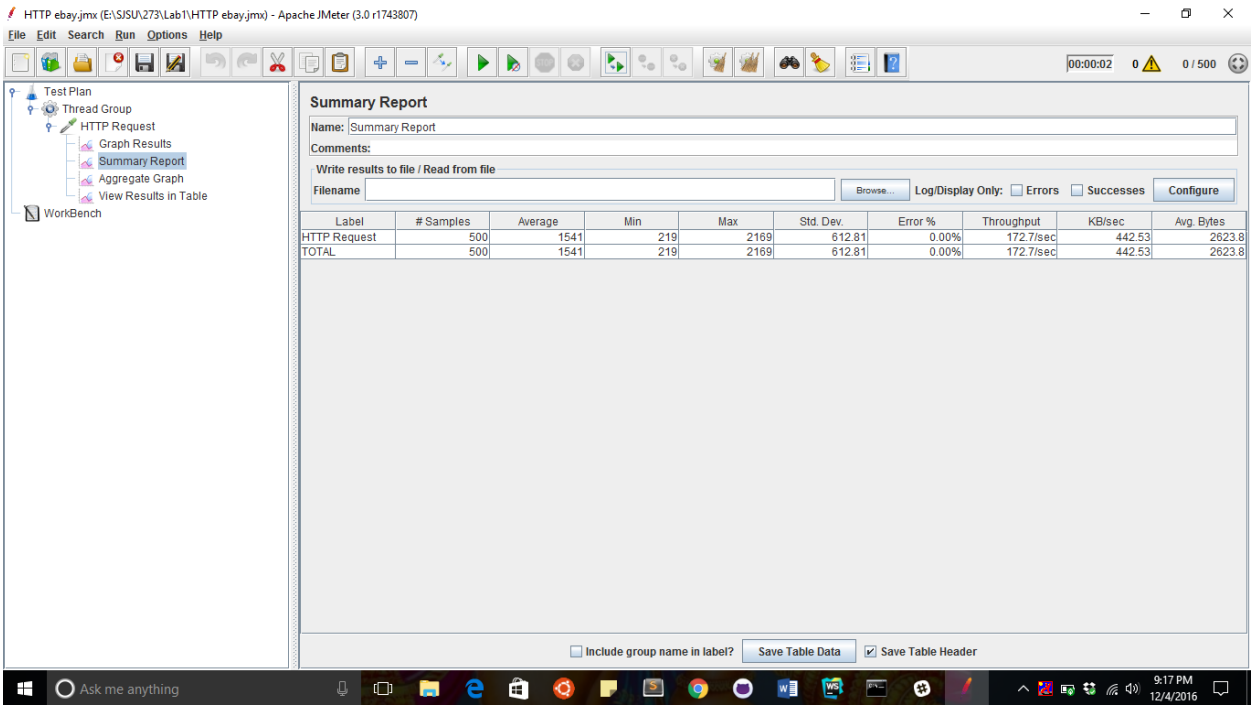
4)400 concurrent users





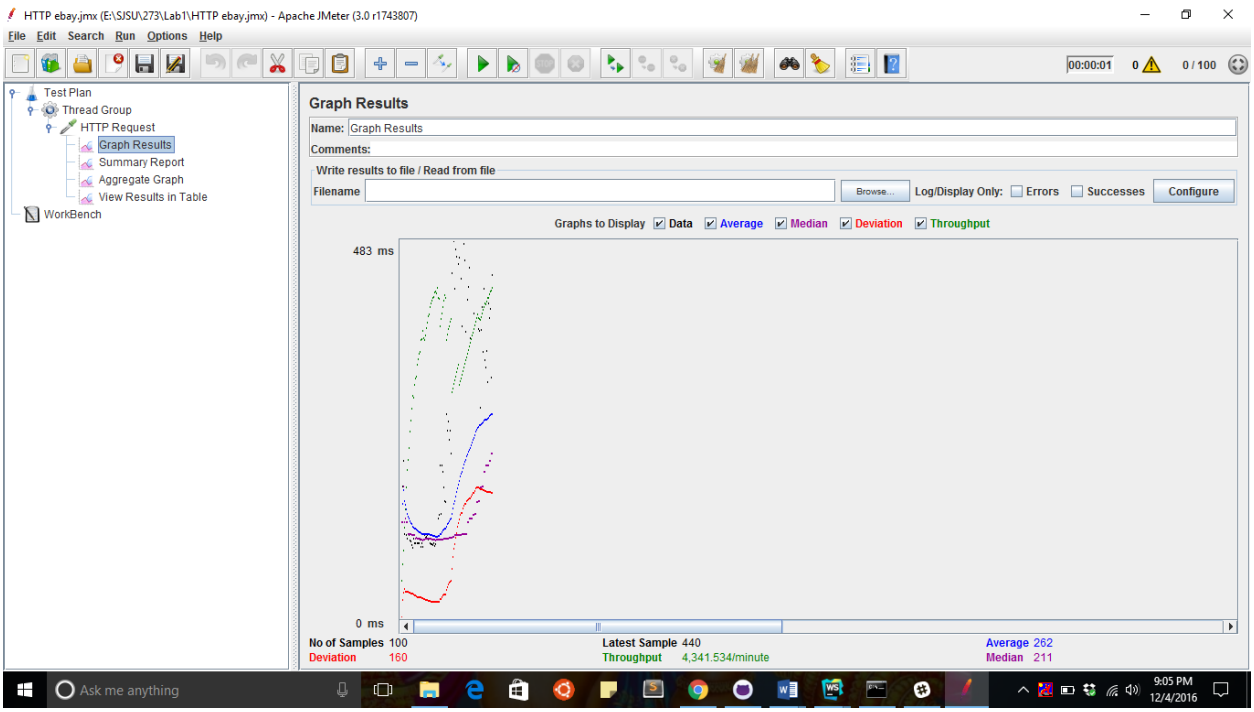
5)500 concurrent users

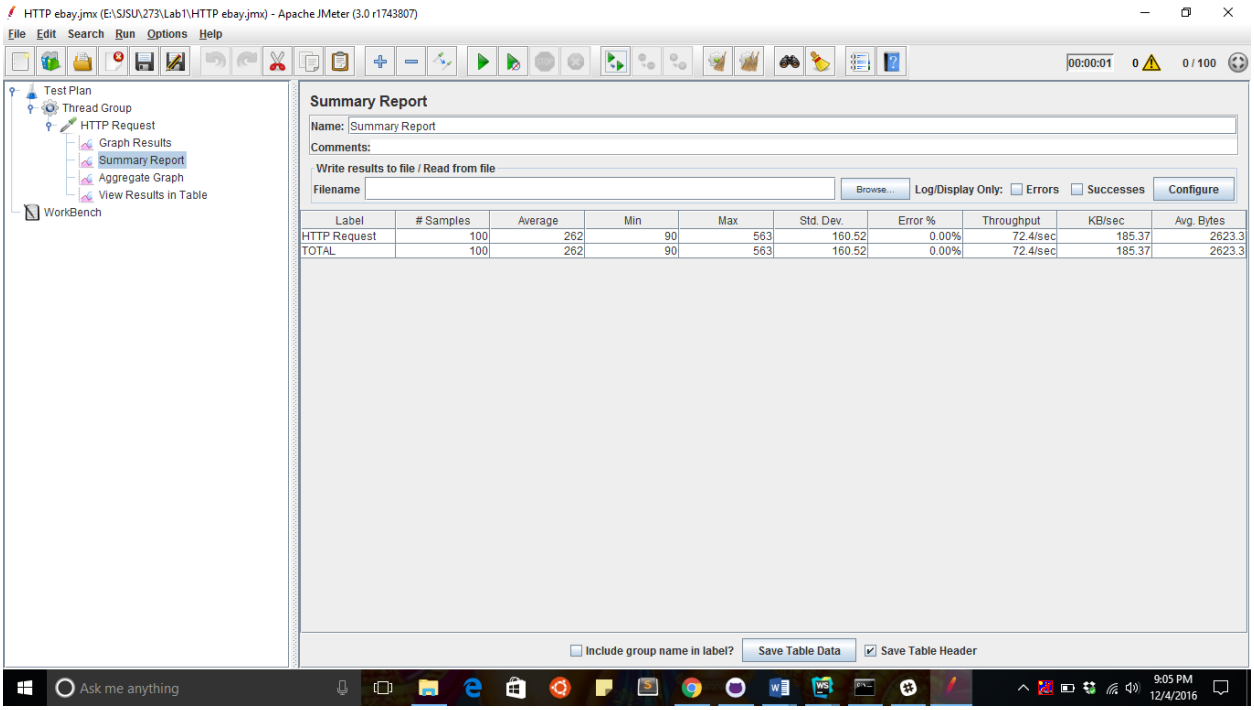




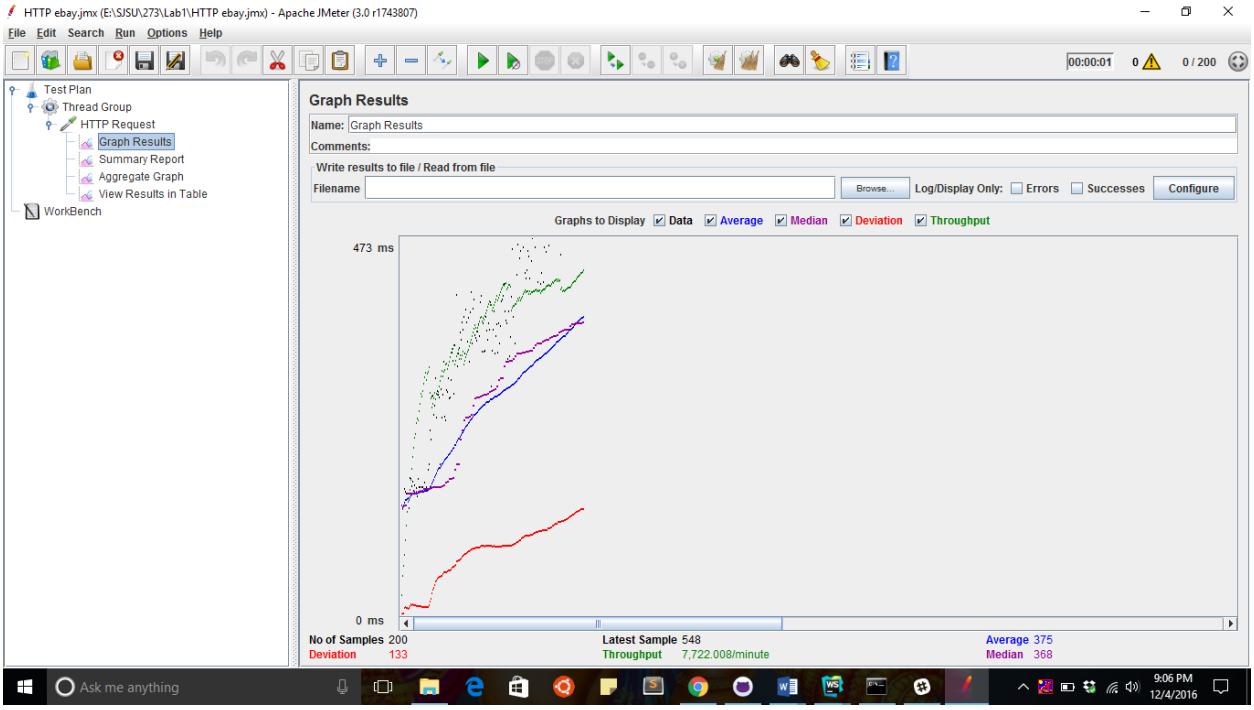
2. Base + Connection Pooling

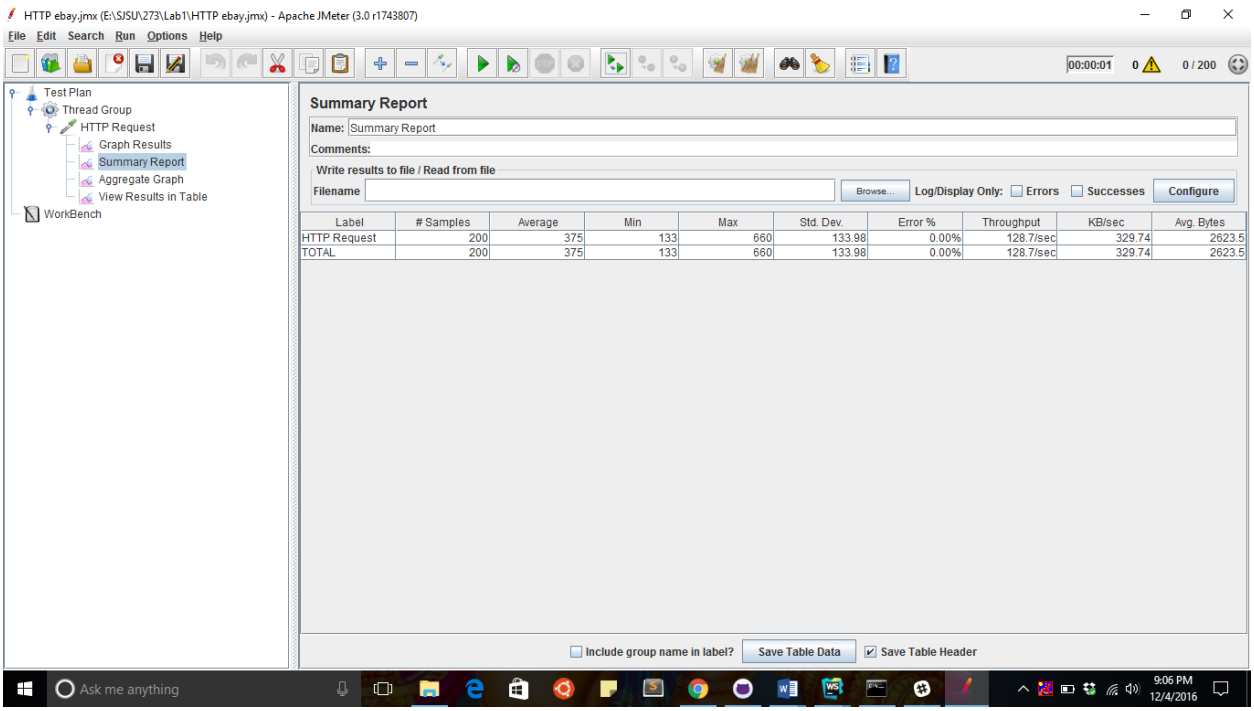
1)100 concurrent users



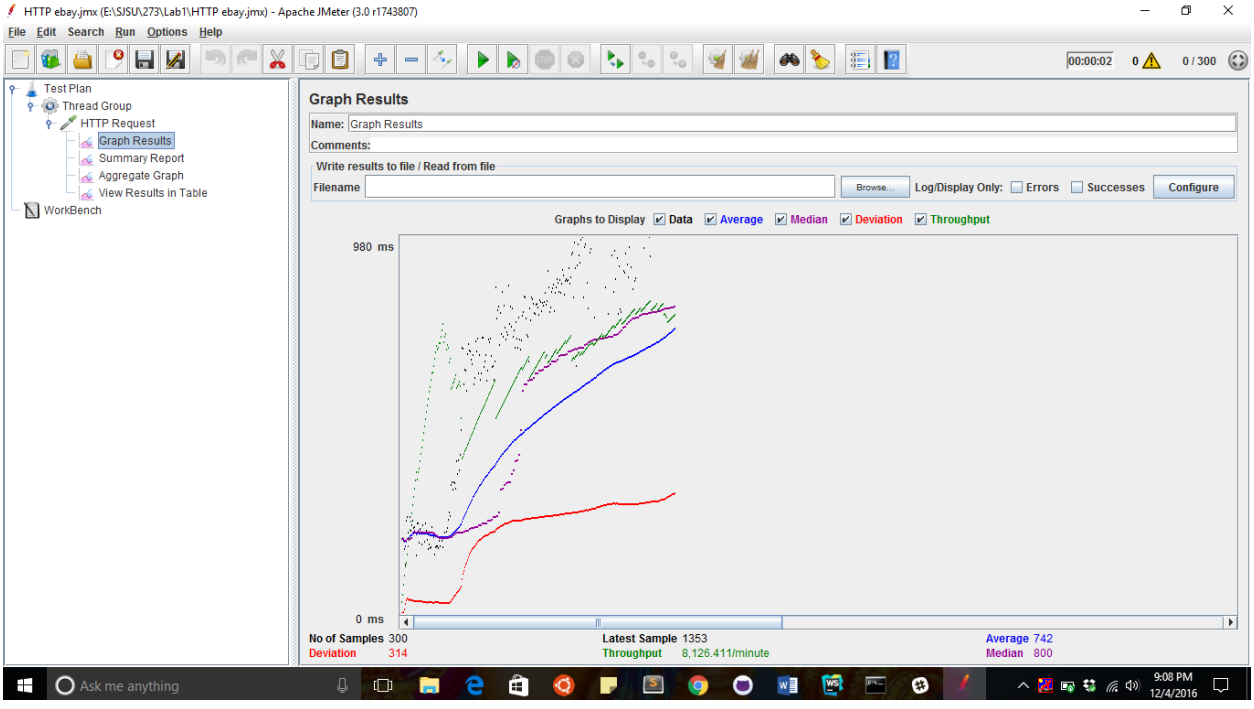


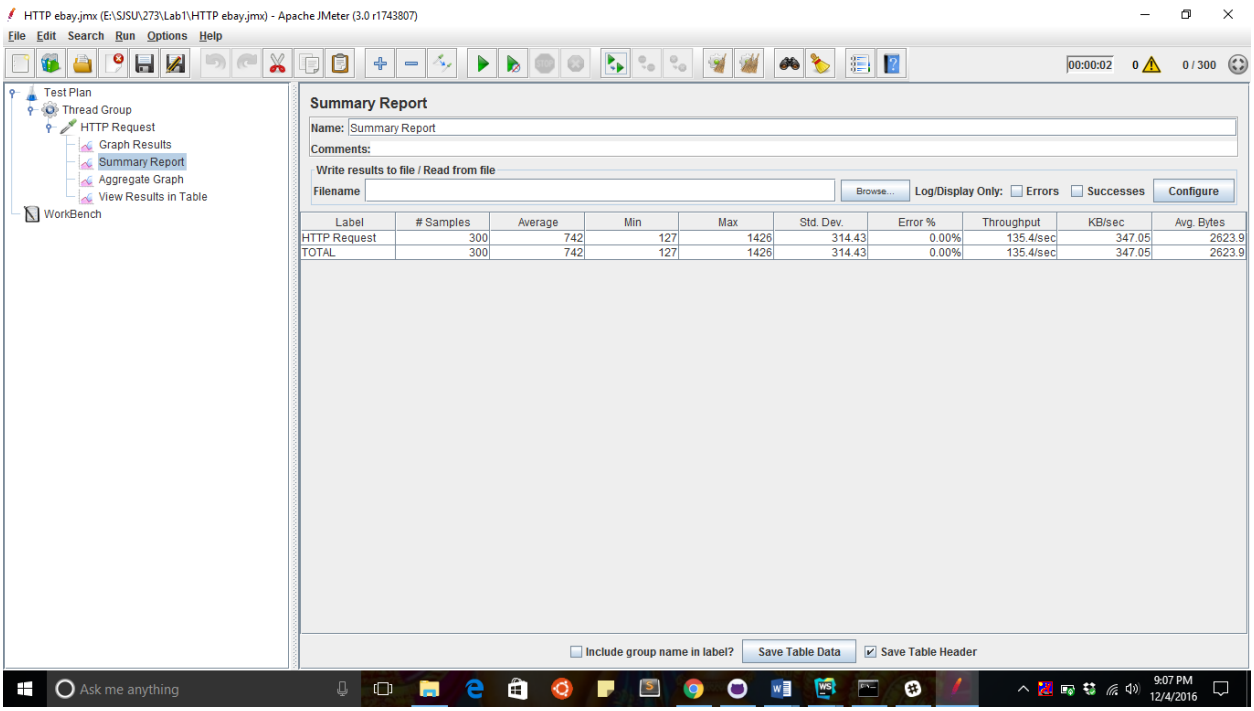
2)200 concurrent users



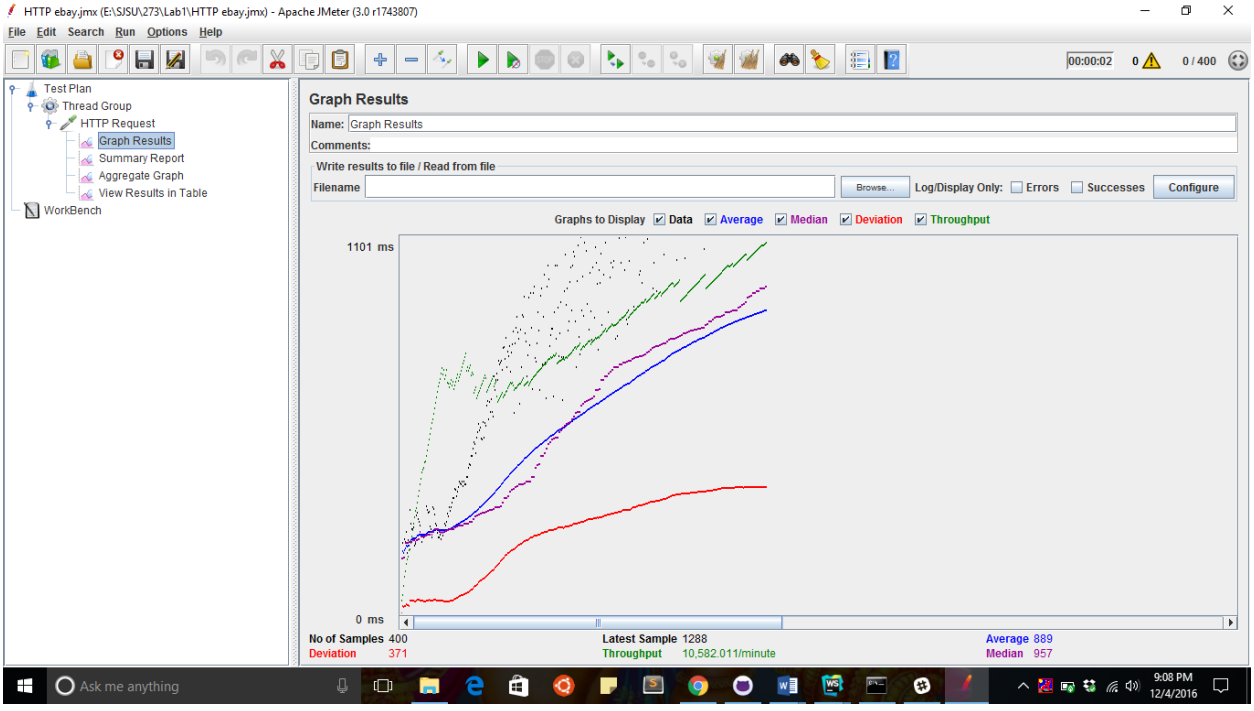


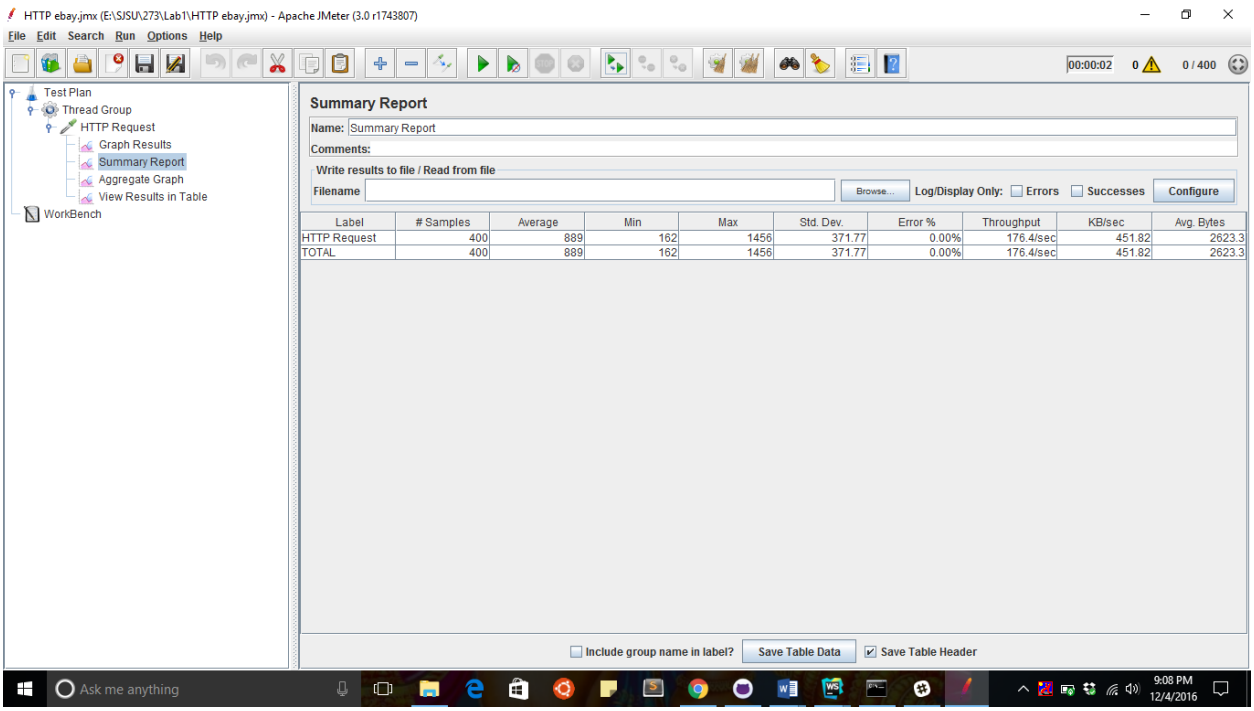
3)300 concurrent users



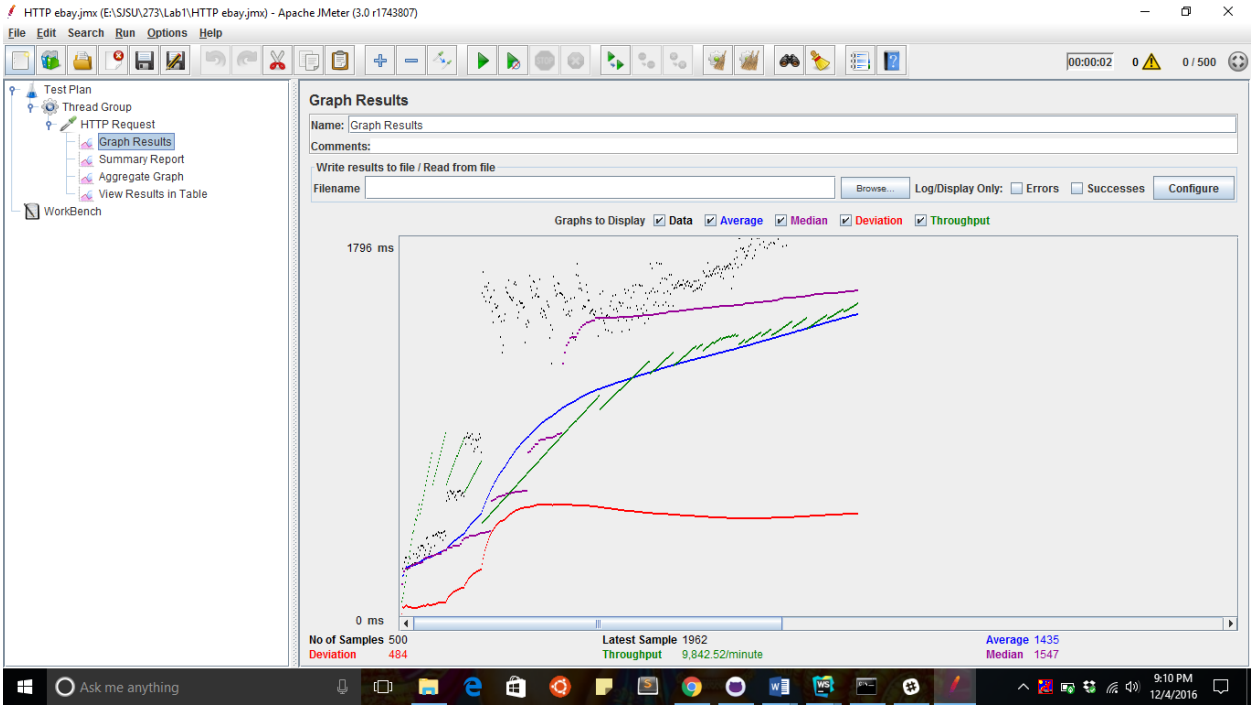


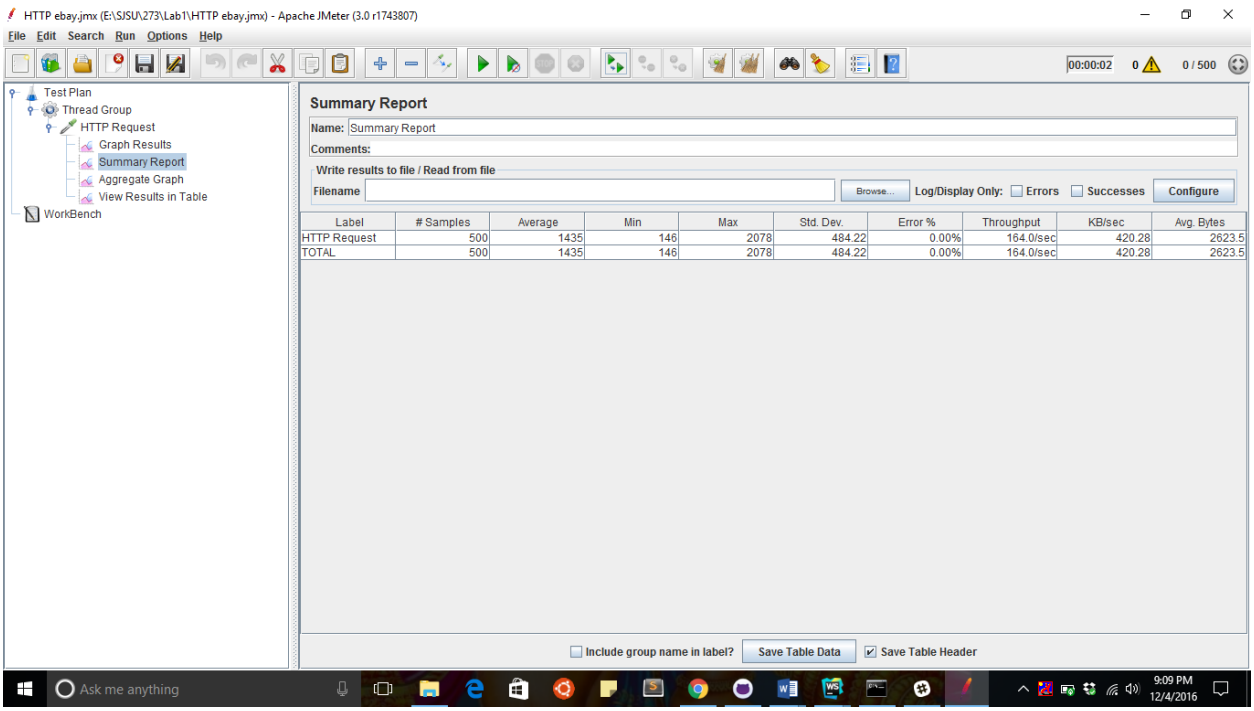
4)400 concurrent users





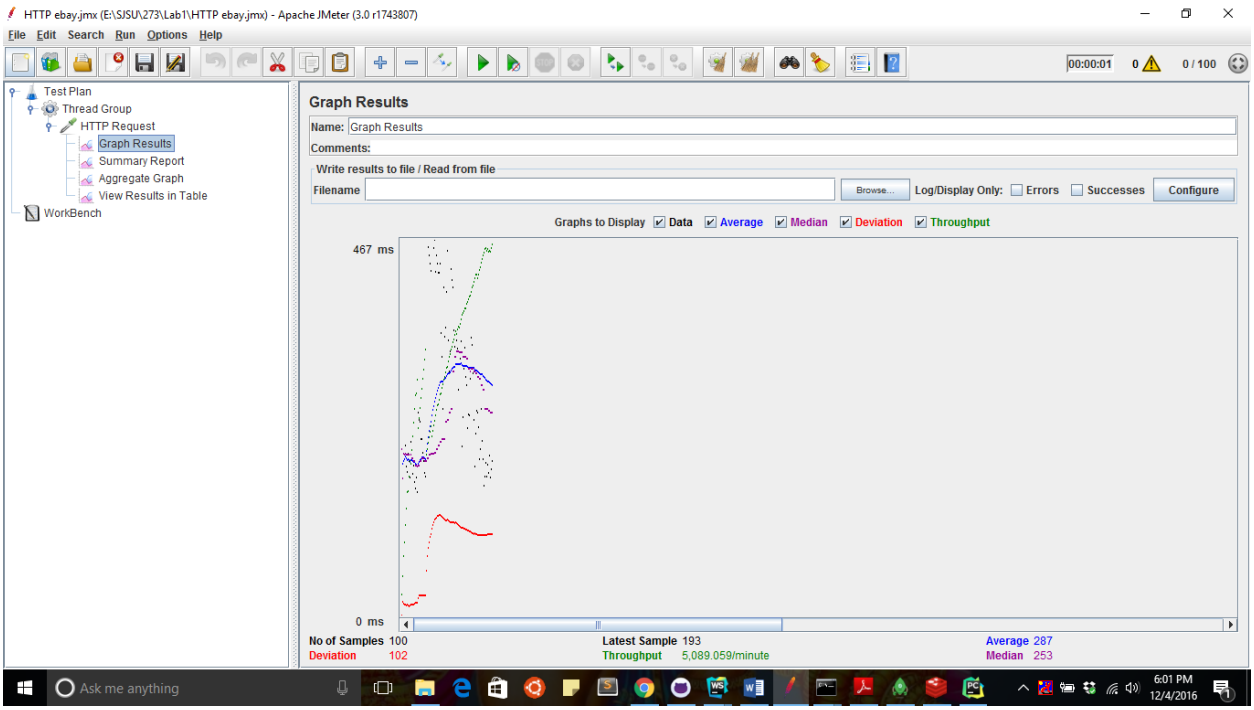
5)500 concurrent Users

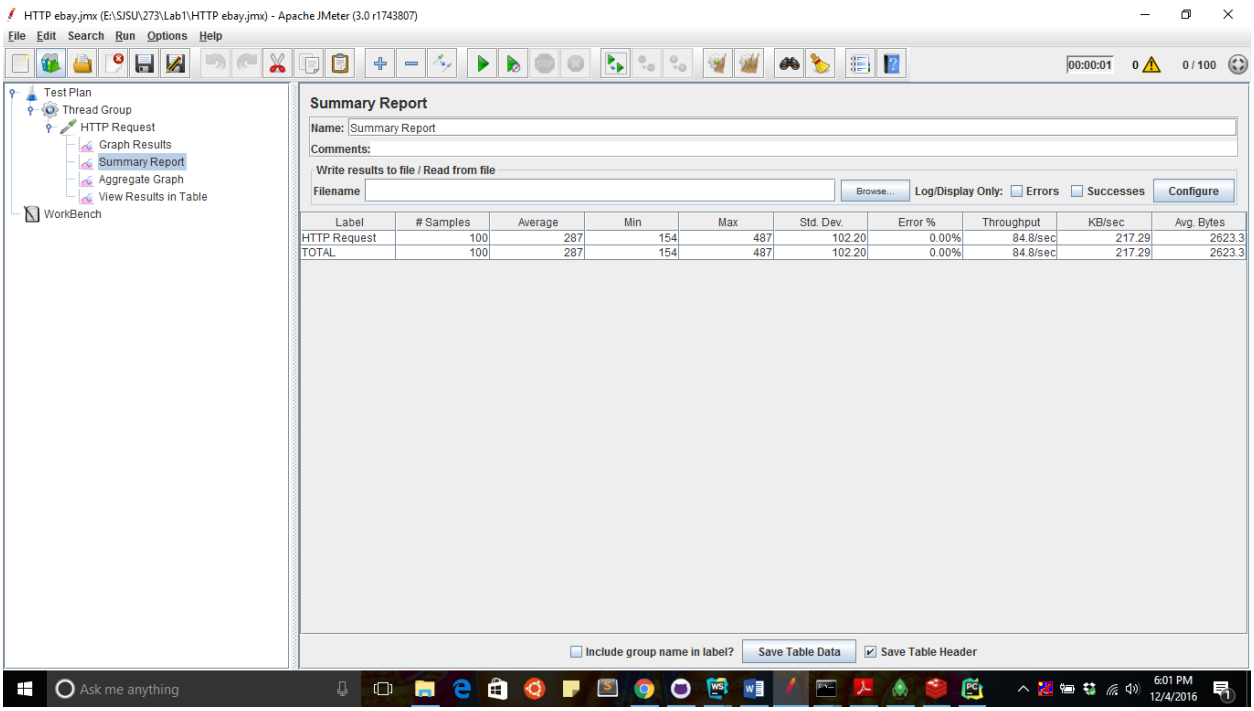




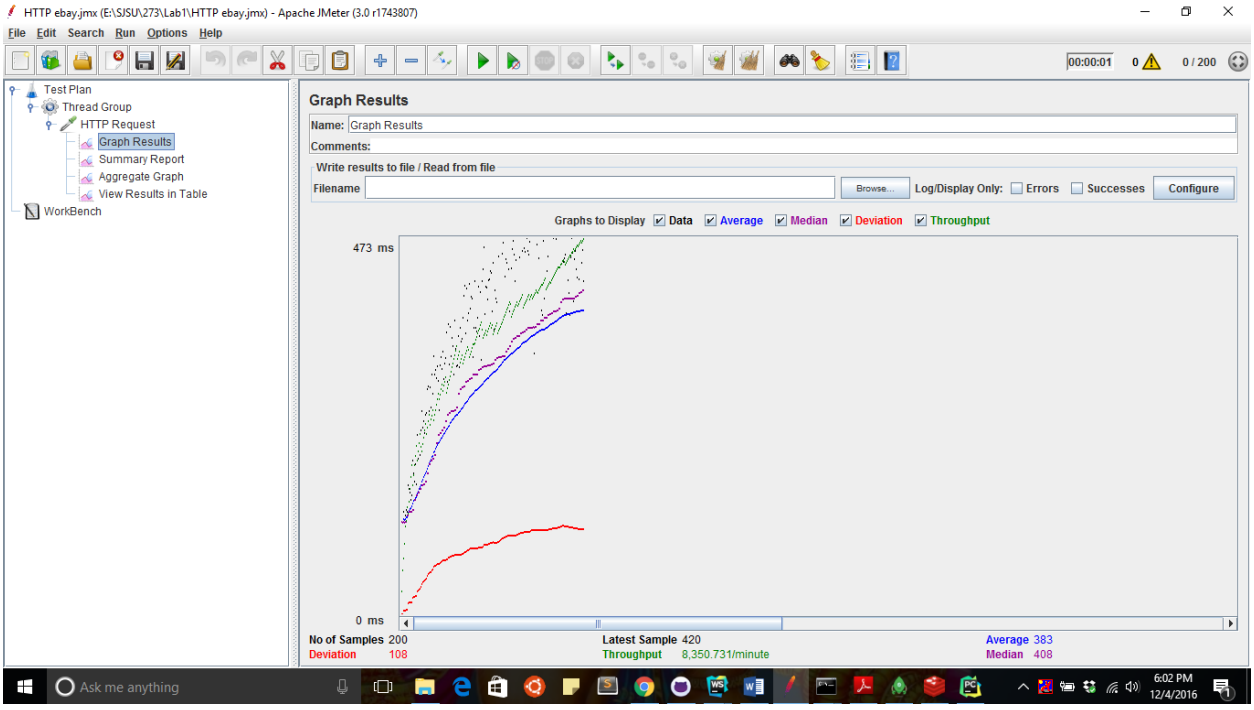
3. Base + RabbitMQ

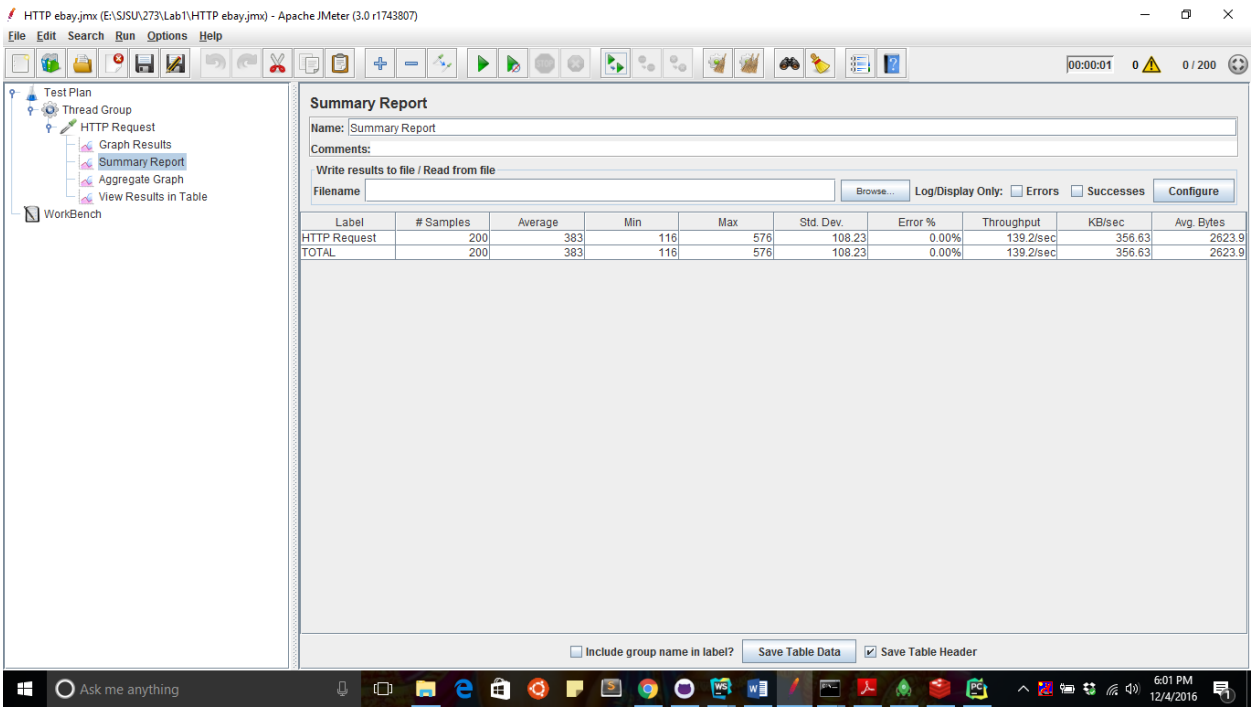
1)100 concurrent users



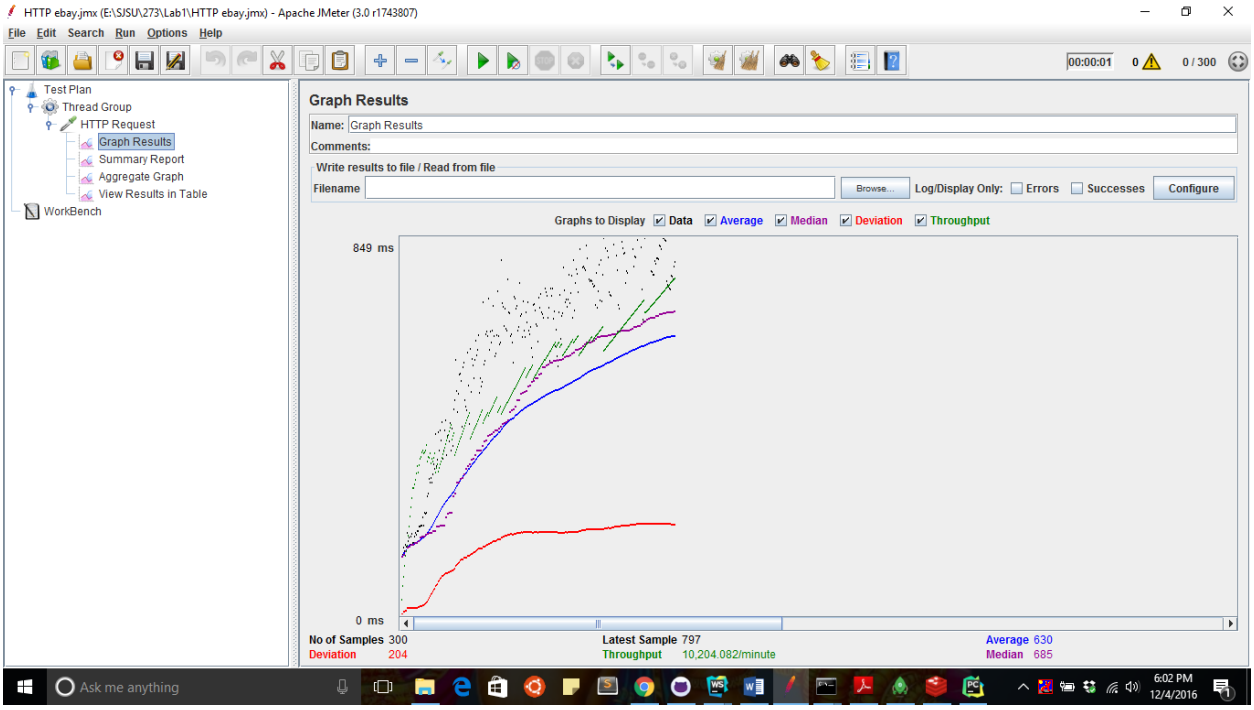


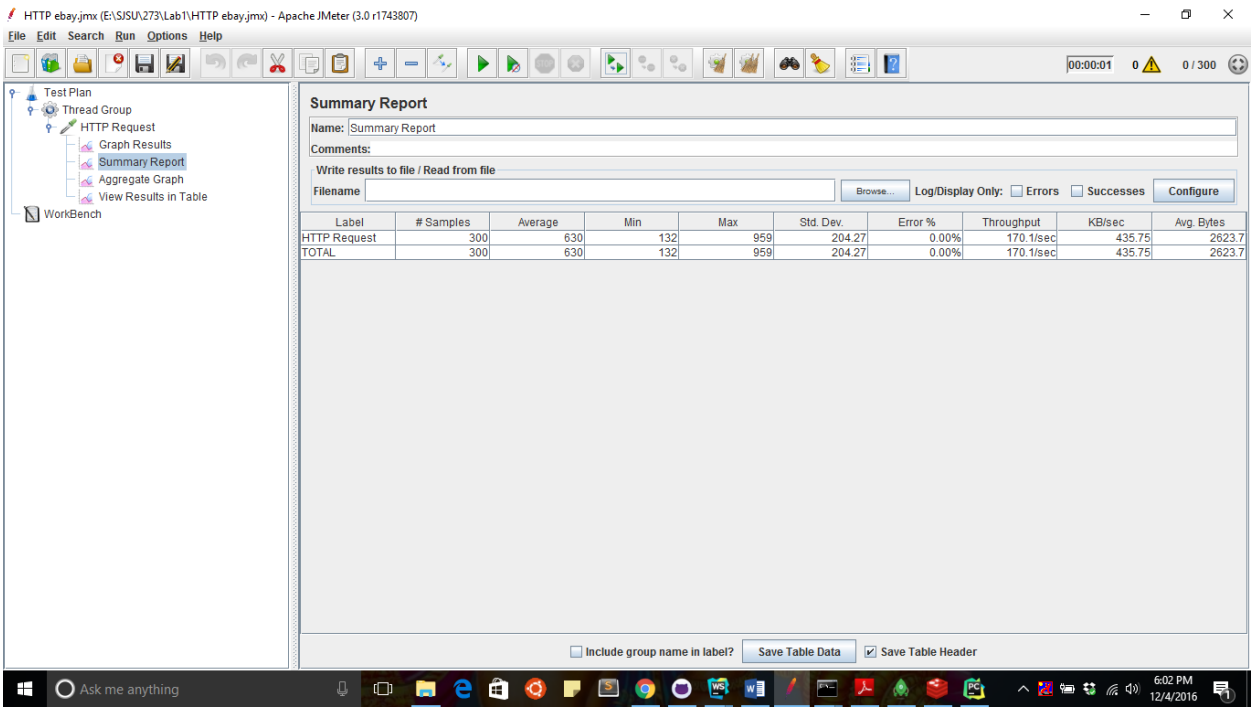
2)200 concurrent users



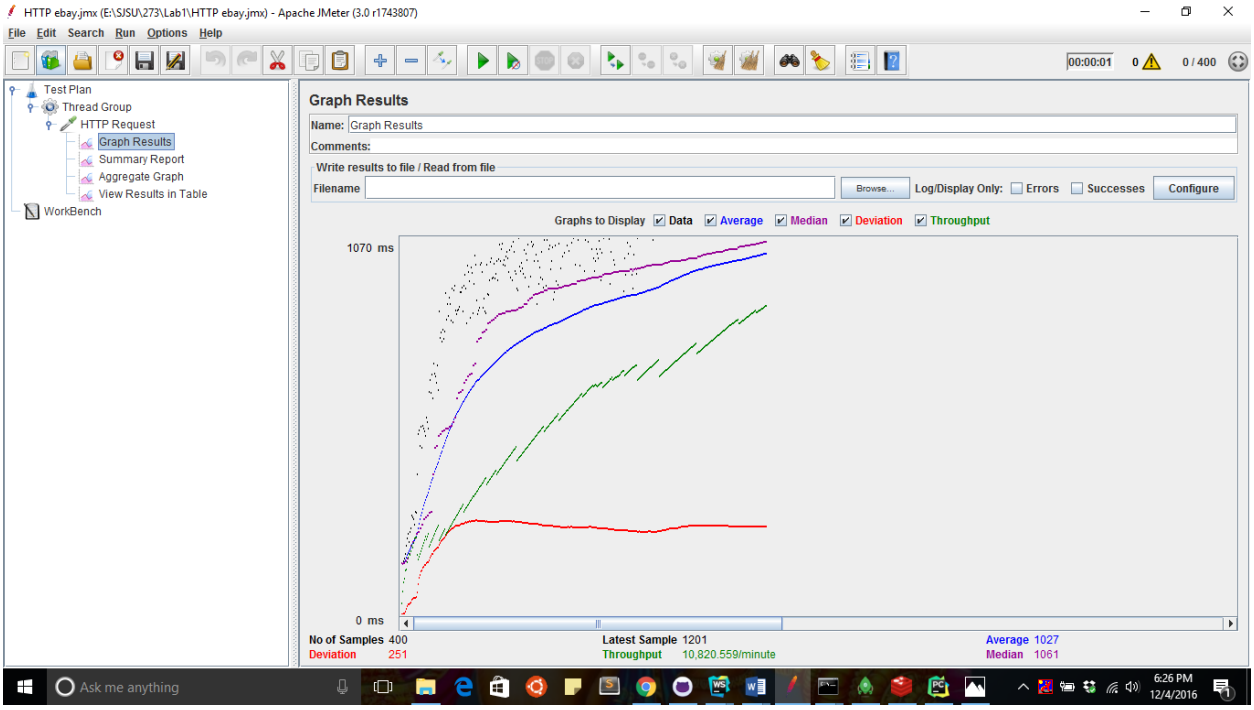


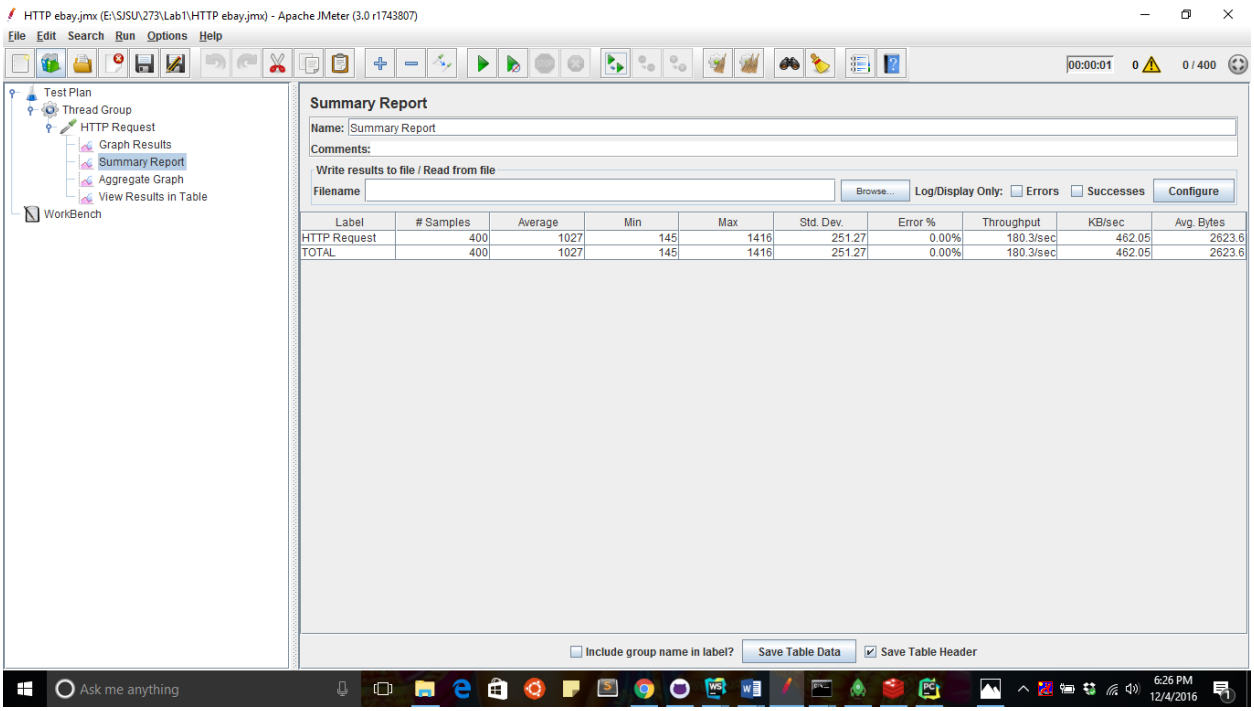
3)300 concurrent users



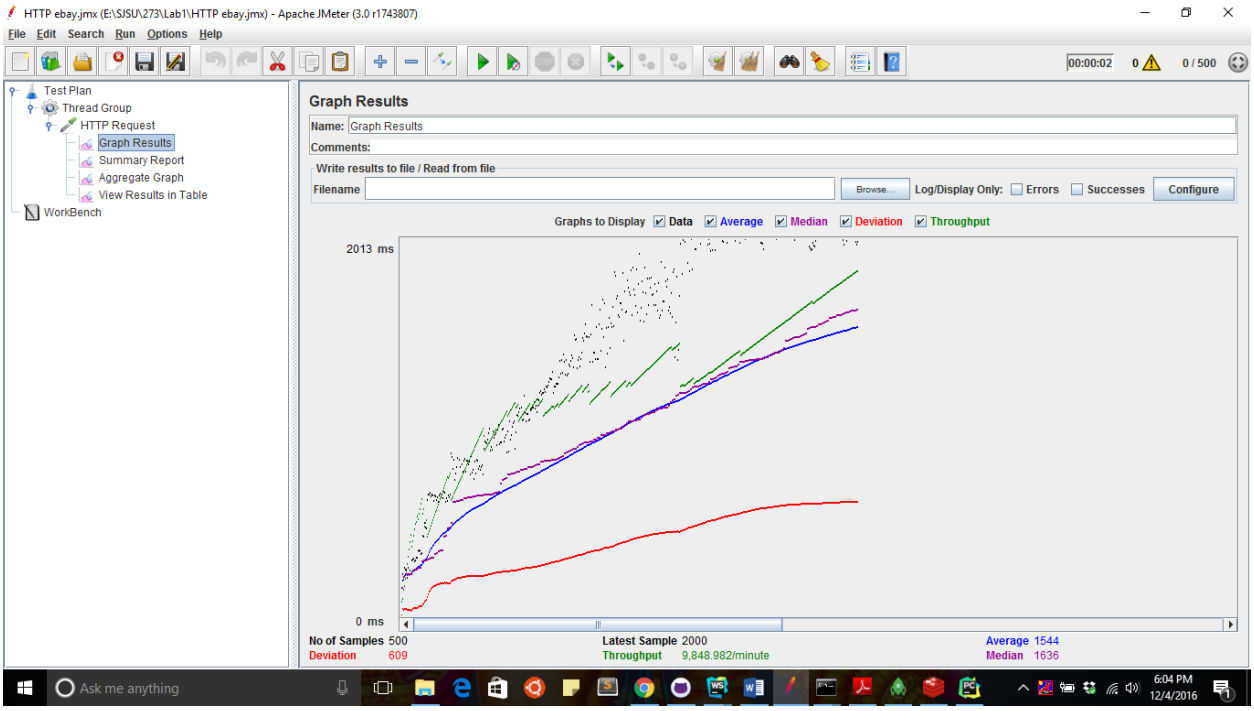


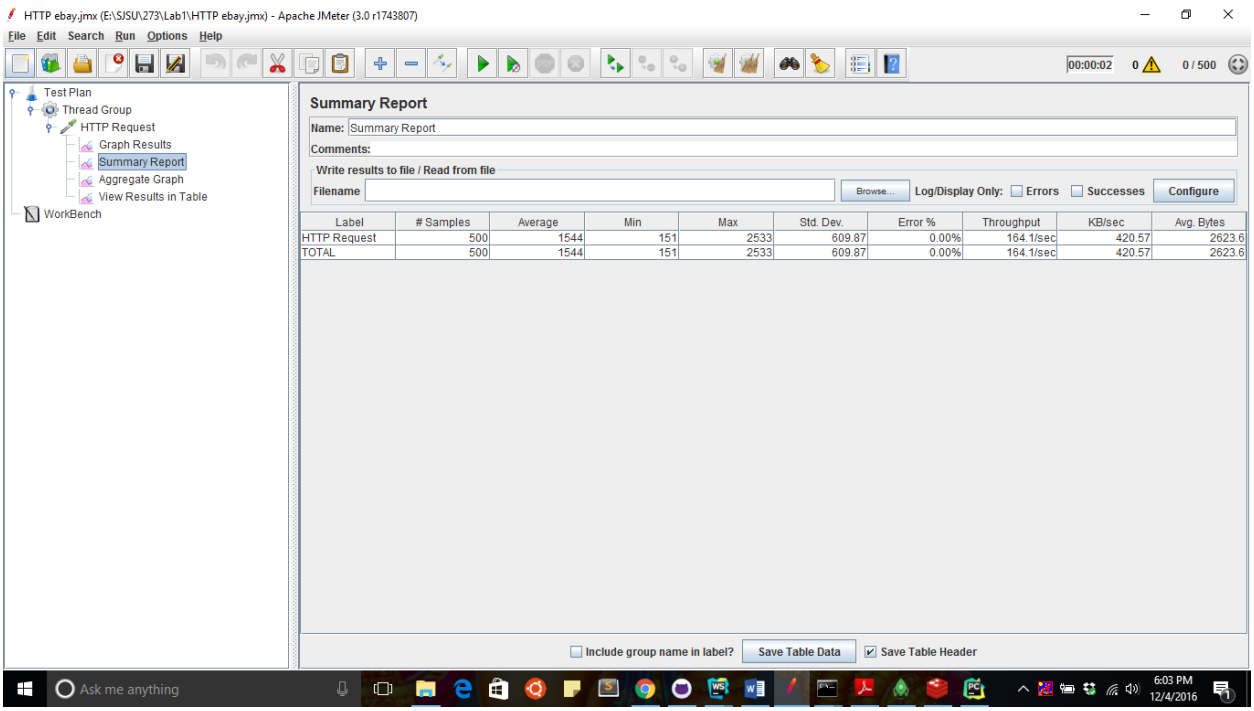
4)400 concurrent users





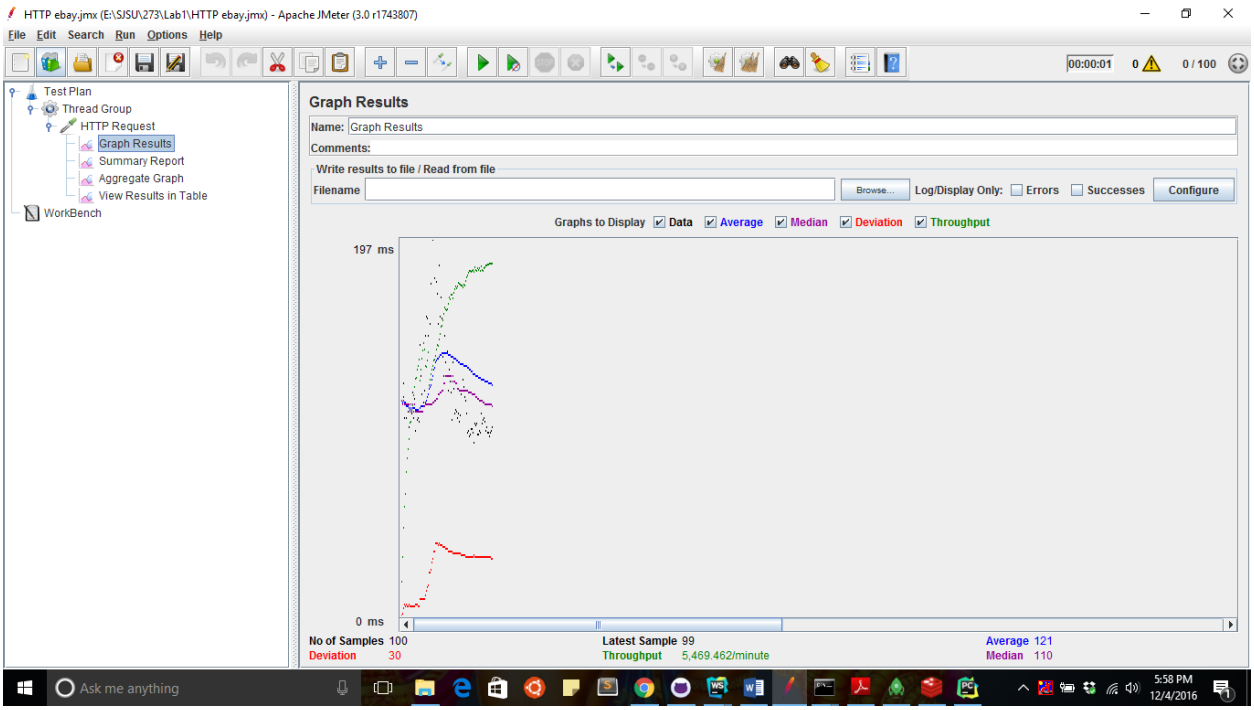
5)500 concurrent Users

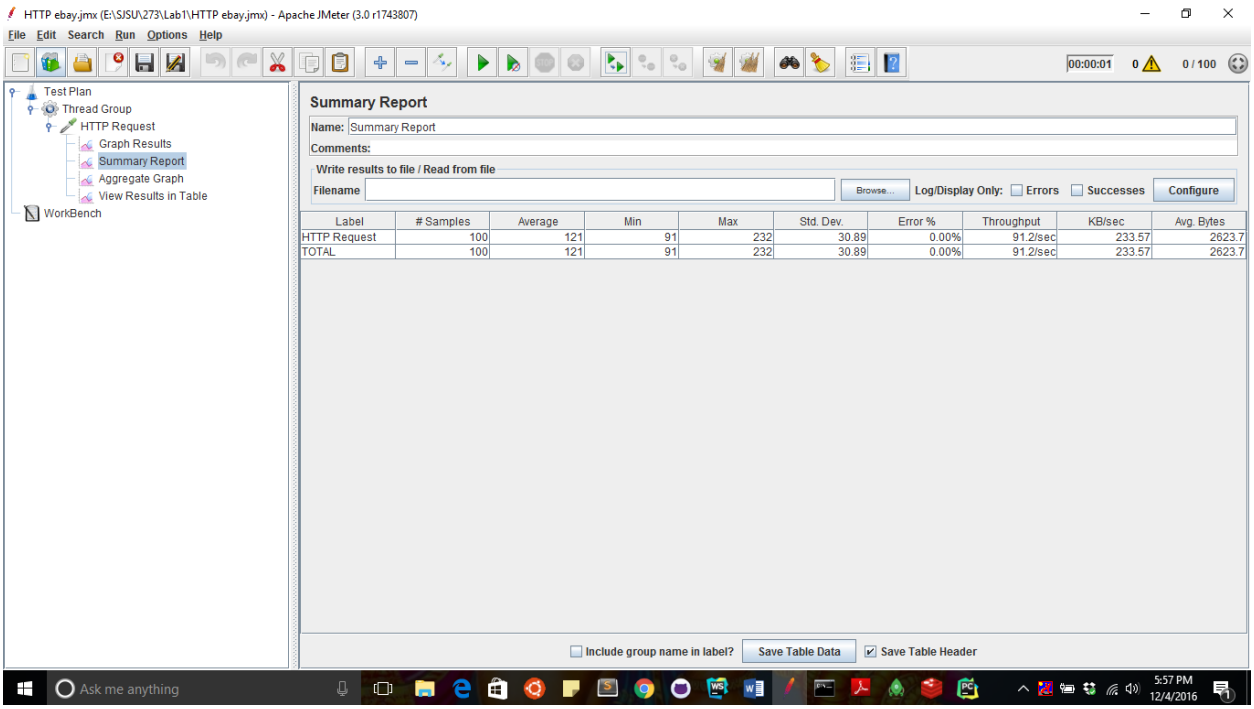




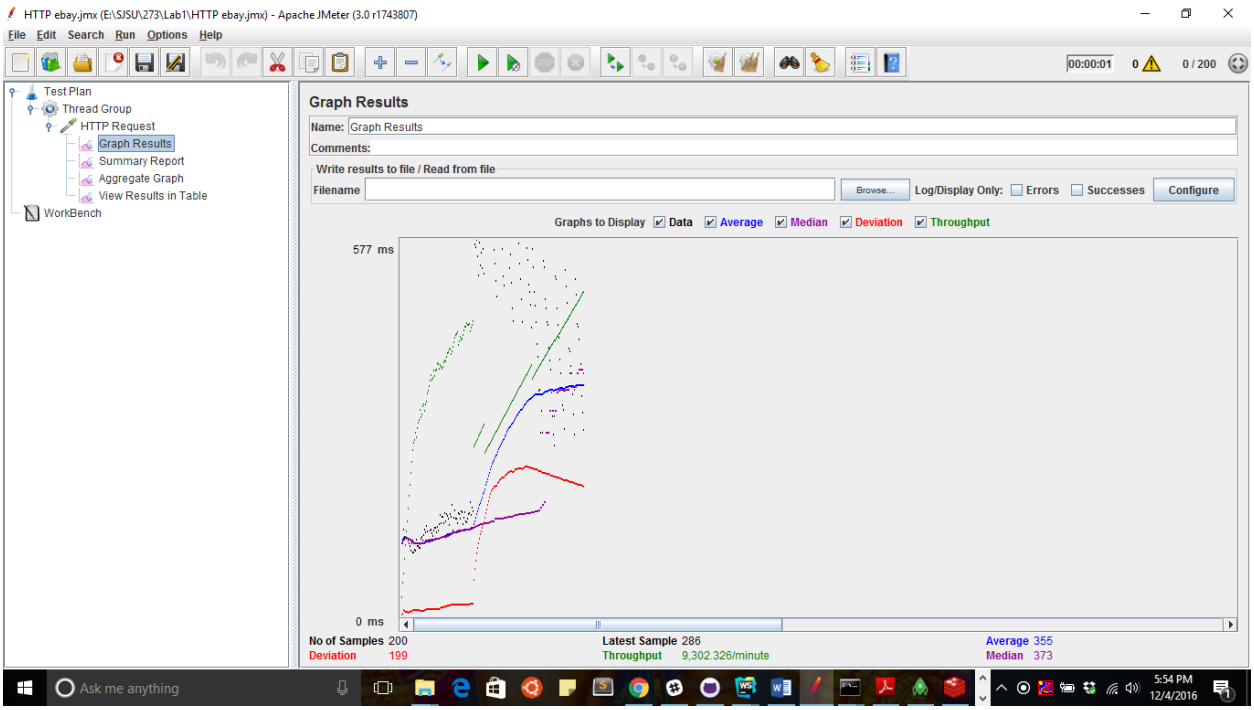
4. Base + RabbitMQ+ Connection Pooling

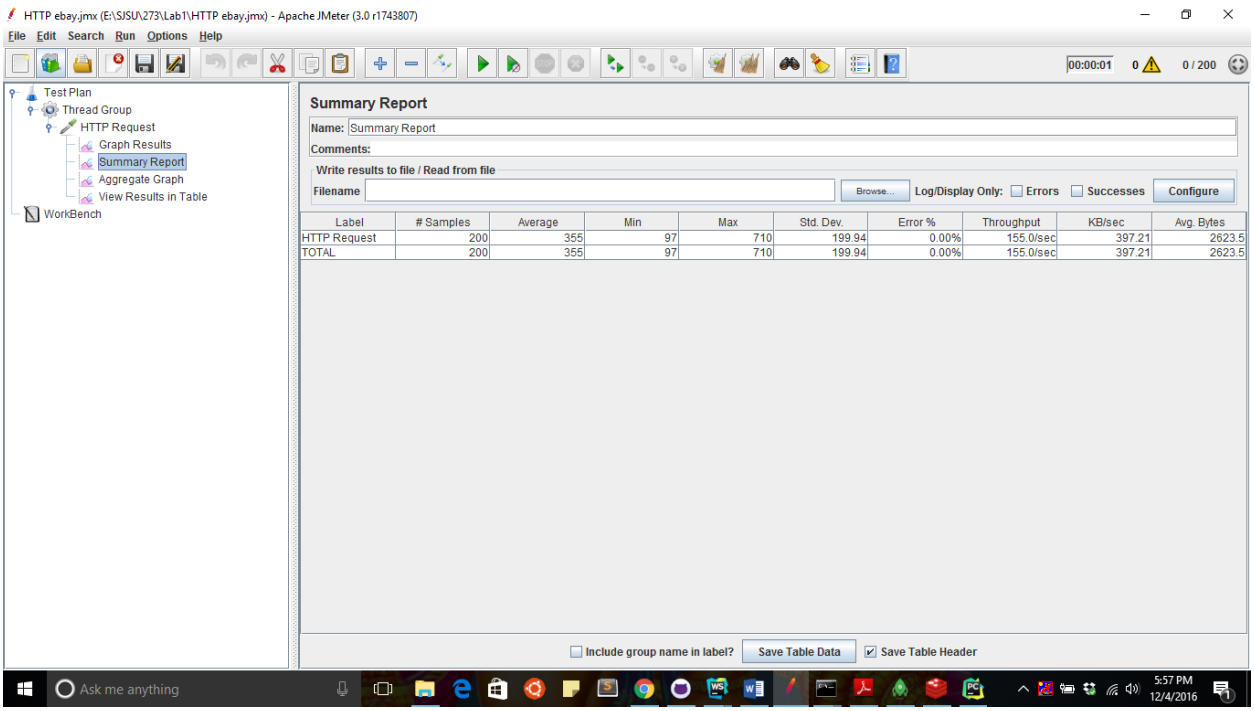
1)100 concurrent users



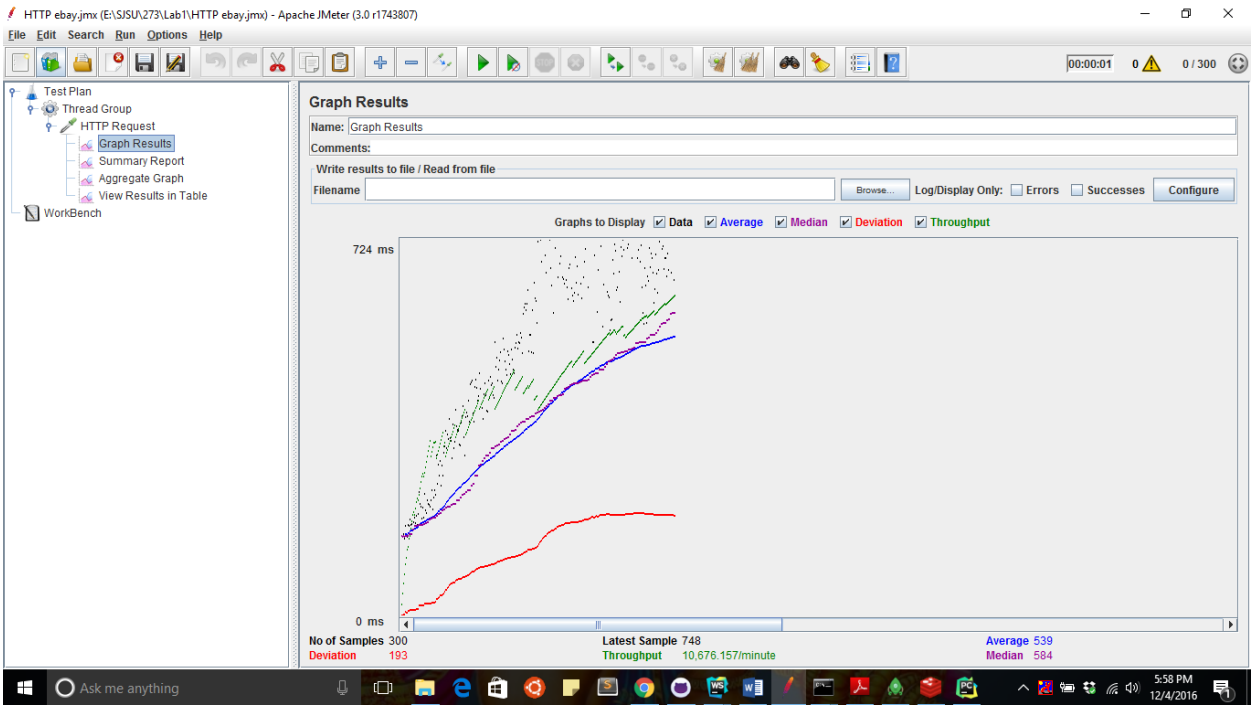


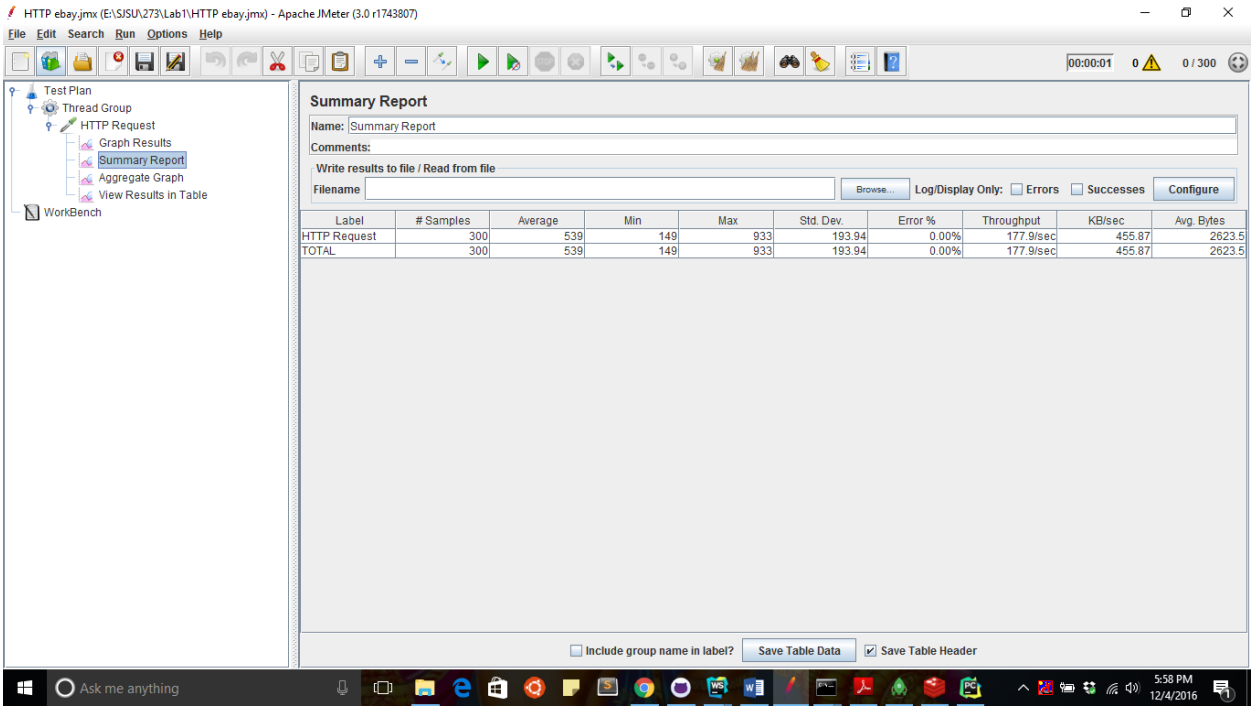
2)200 concurrent users



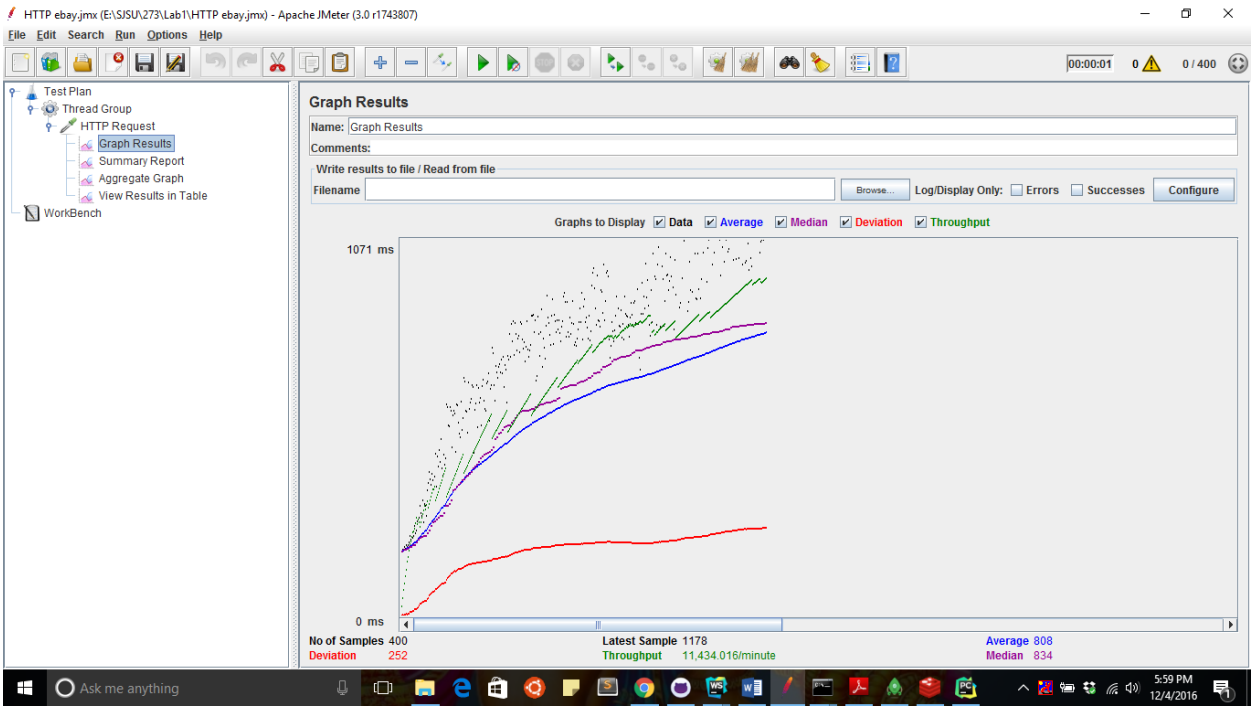


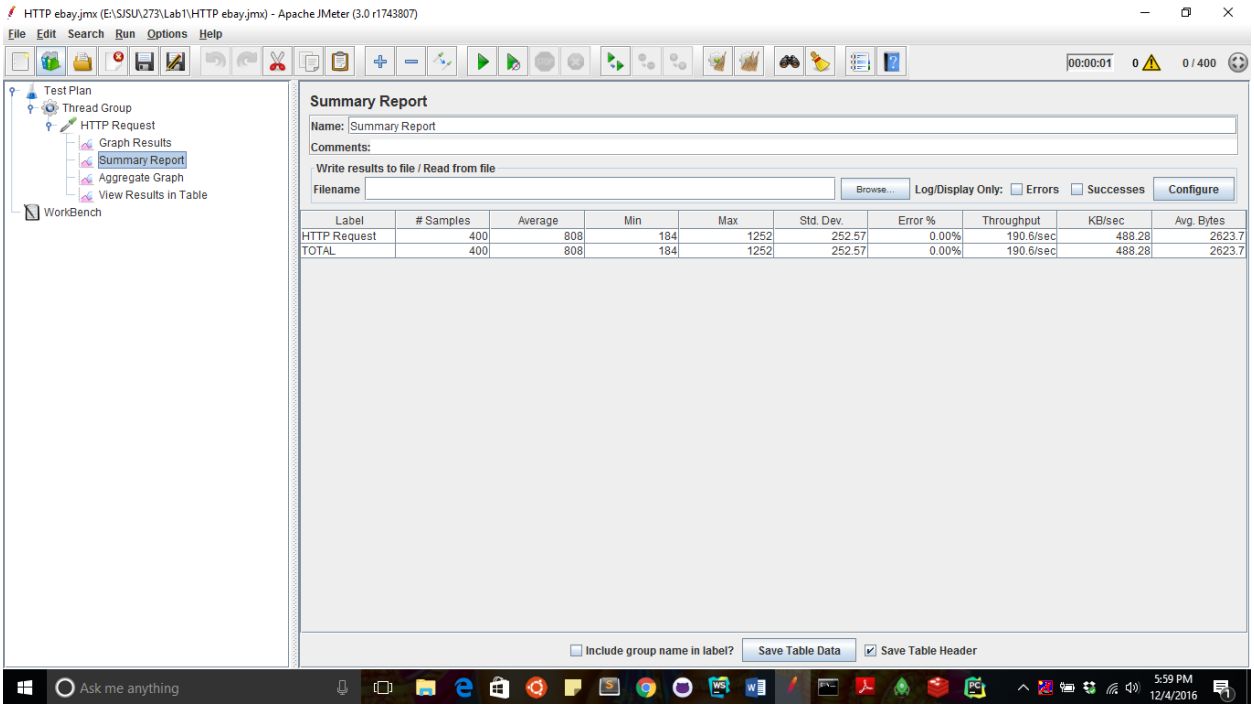
3)300 concurrent users



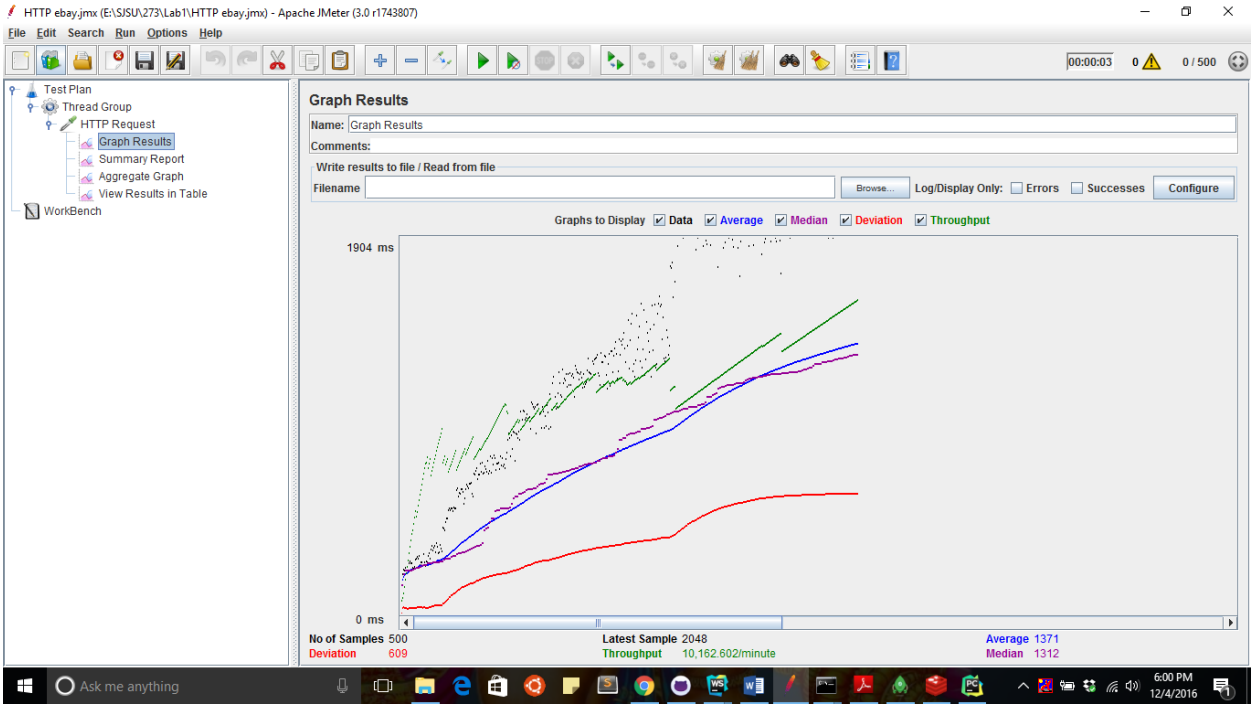


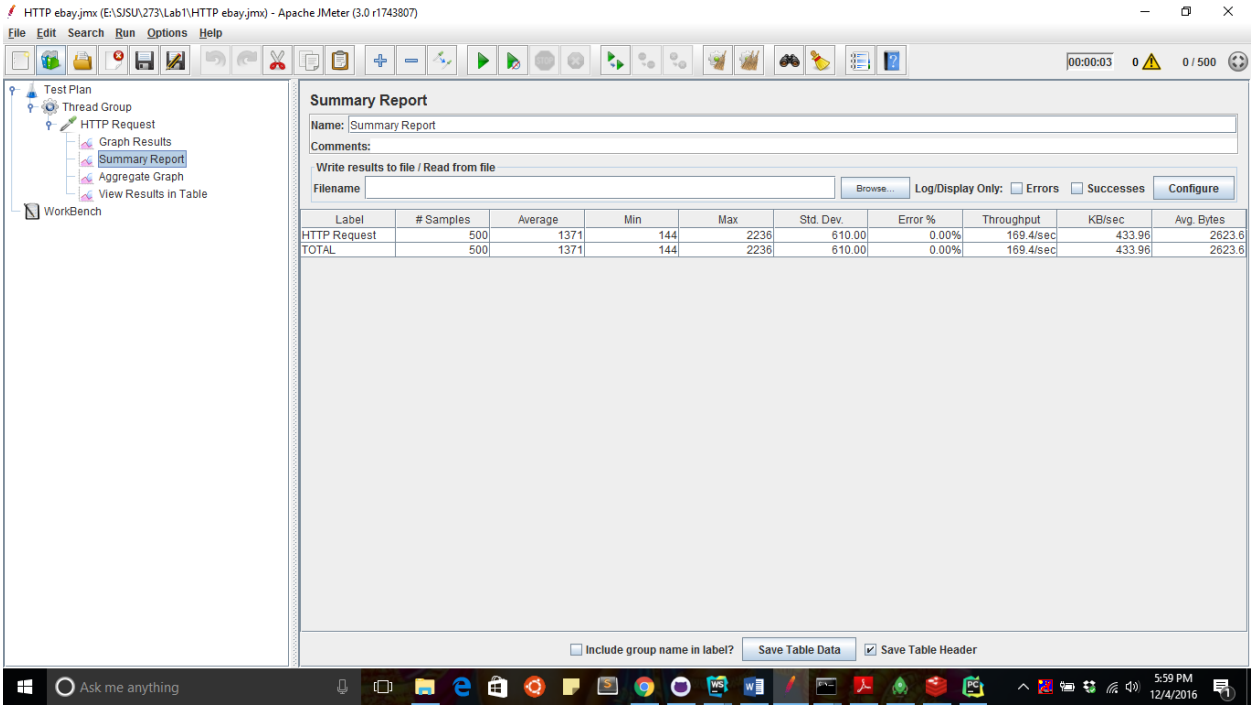
4)400 concurrent users





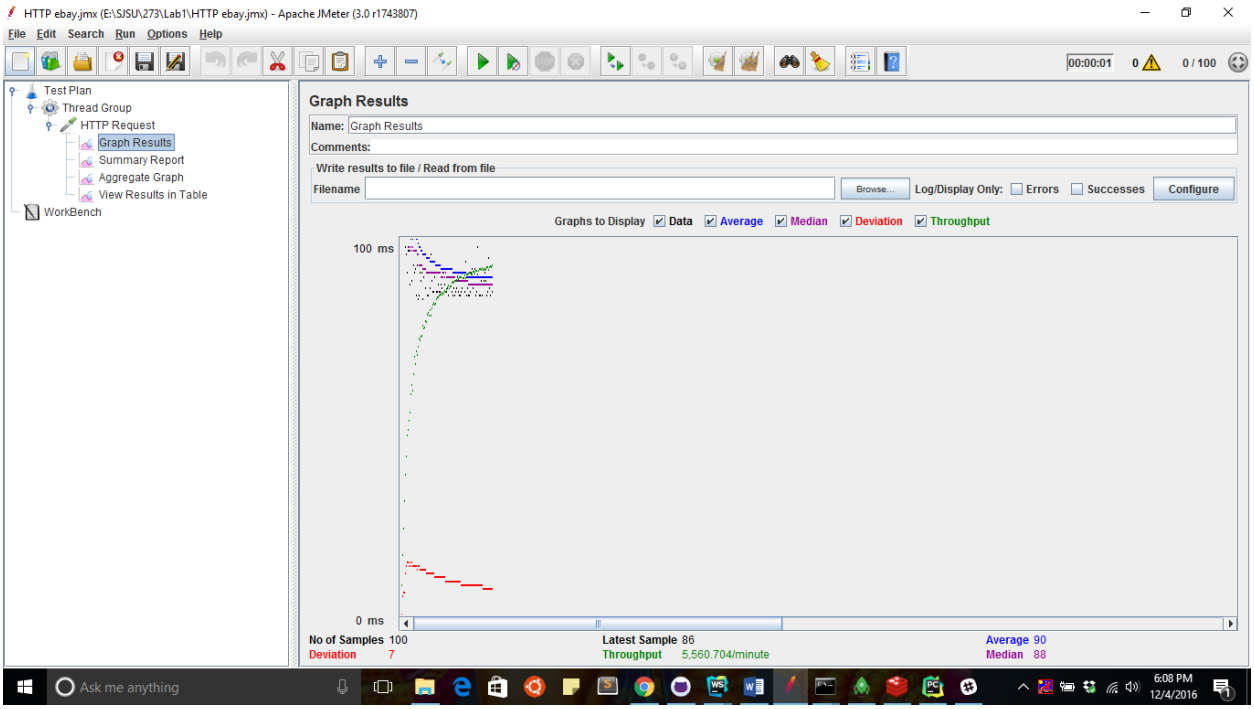
5)500 concurrent Users

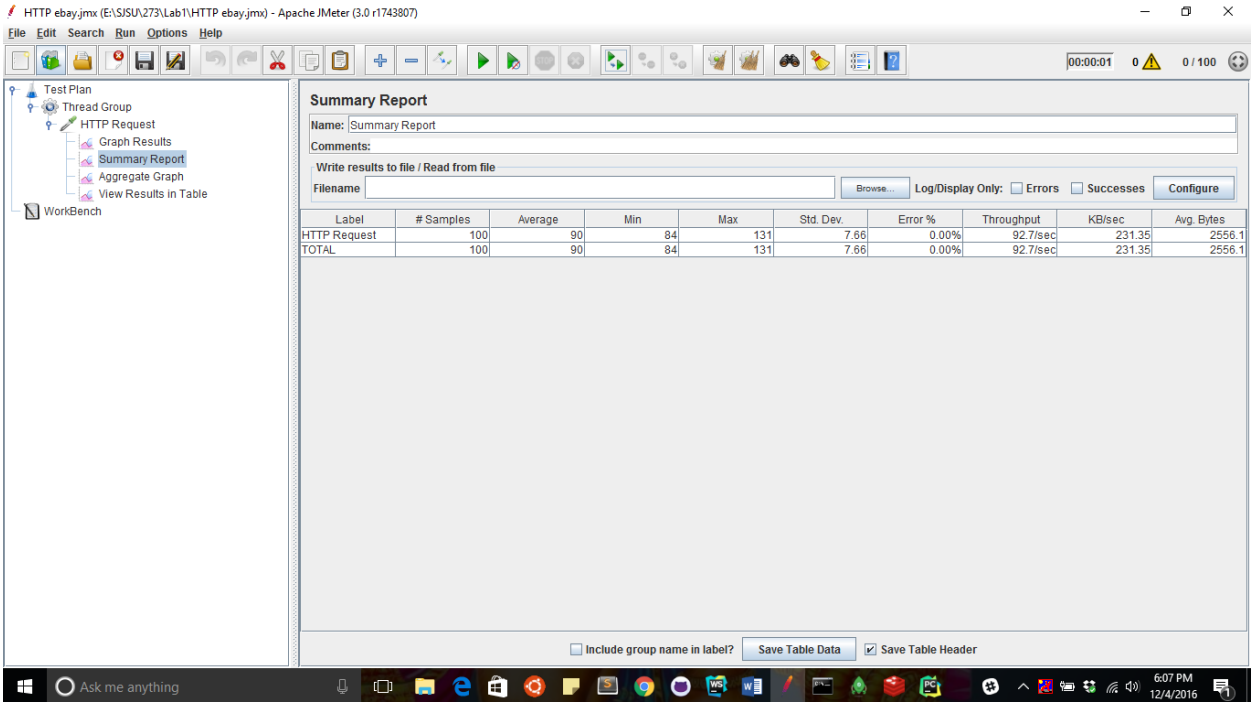

HTTP ebay.jmx (E:\SJSU\273\Lab1\HTTP ebay.jmx) - Apache JMeter (3.0 r1743807)
File Edit Search Run Options Help
00:00:03 0 0 / 500
Test Plan
Thread Group
HTTP Request
Graph Results
Summary Report
Aggregate Graph
View Results in Table
WorkBench
Graph Results
Name: Graph Results
Comments:
Write results to file / Read from file
Filename
Browse...
Log/Display Only: Errors Successes
Configure
Graphs to Display Data Average Median Deviation Throughput
1904 ms
0 ms
No of Samples 500
Latest Sample 2048
Average 1371
Deviation 609
Throughput 10,162.602/minute
Median 1312
Ask me anything
6:00 PM
12/4/2016


HTTP ebay.jmx (E:\SJSU\273\Lab1\HTTP ebay.jmx) - Apache JMeter (3.0 r1743807)
File Edit Search Run Options Help
00:00:03 0 0 / 500
Test Plan
Thread Group
HTTP Request
Graph Results
Summary Report
Aggregate Graph
View Results in Table
WorkBench
Summary Report
Name: Summary Report
Comments:
Write results to file / Read from file
Filename
Browse...
Log/Display Only: Errors Successes
Configure
Label # Samples Average Min Max Std. Dev. Error % Throughput KB/sec Avg. Bytes
HTTP Request 500 1371 144 2236 610.00 0.00% 169.4/sec 433.96 2623.6
TOTAL 500 1371 144 2236 610.00 0.00% 169.4/sec 433.96 2623.6
Include group name in label?
Save Table Data
Save Table Header
Ask me anything
5:59 PM
12/4/2016

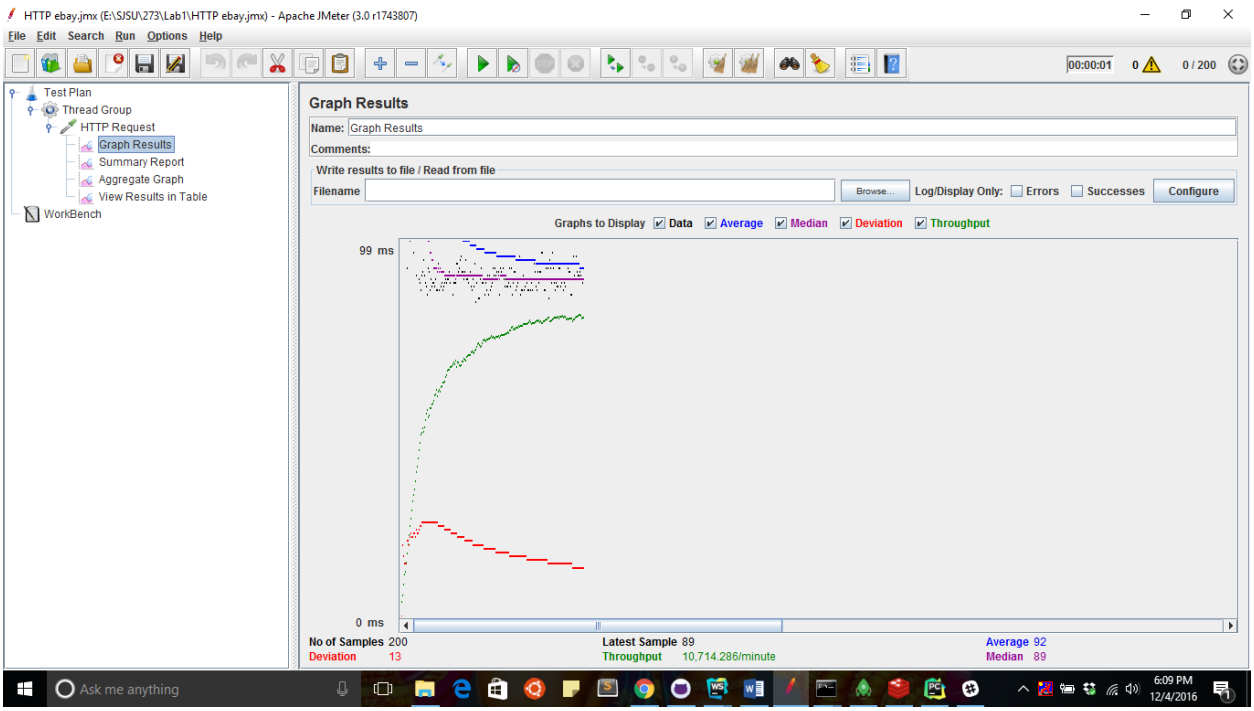
5. Base + RabbitMQ + Connection Pooling + Redis

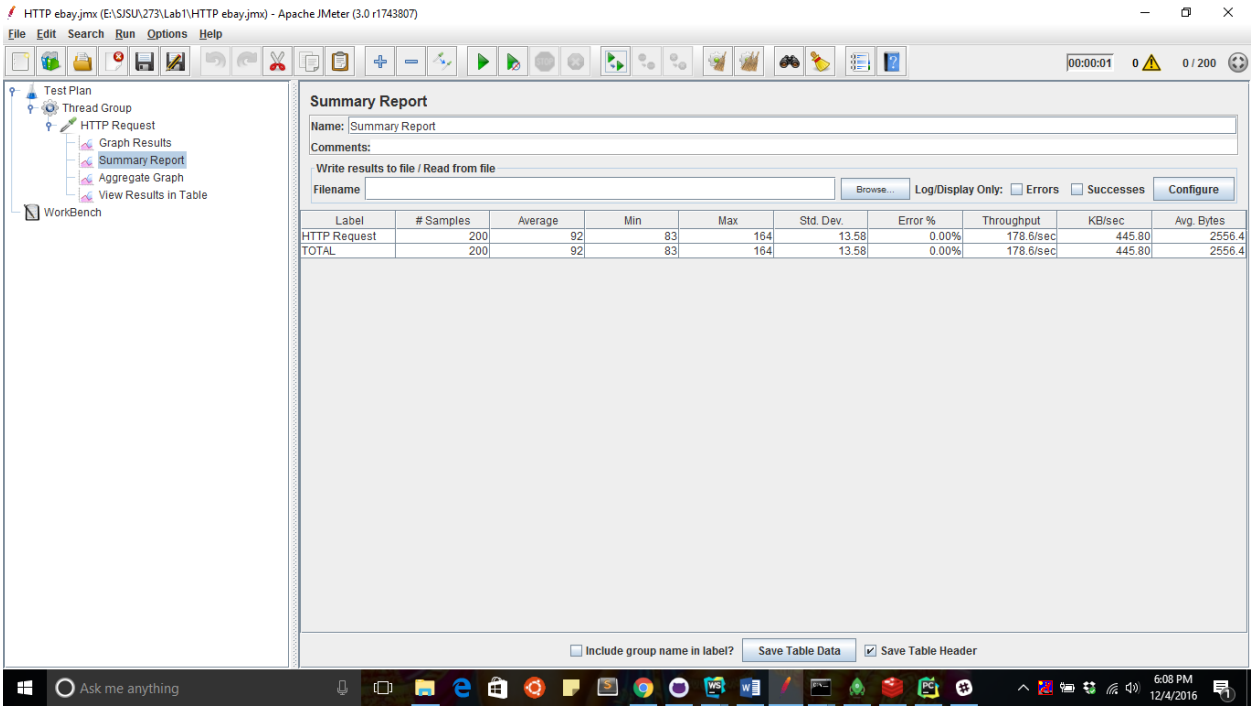
1)100 concurrent users



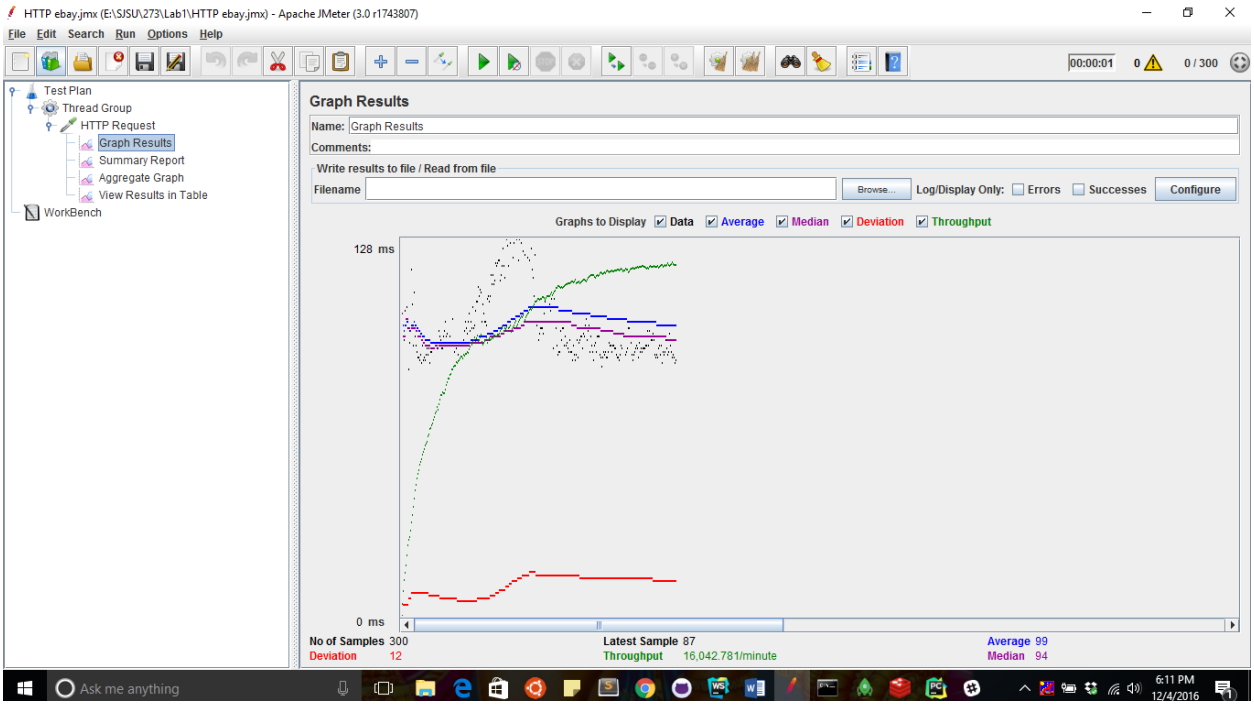


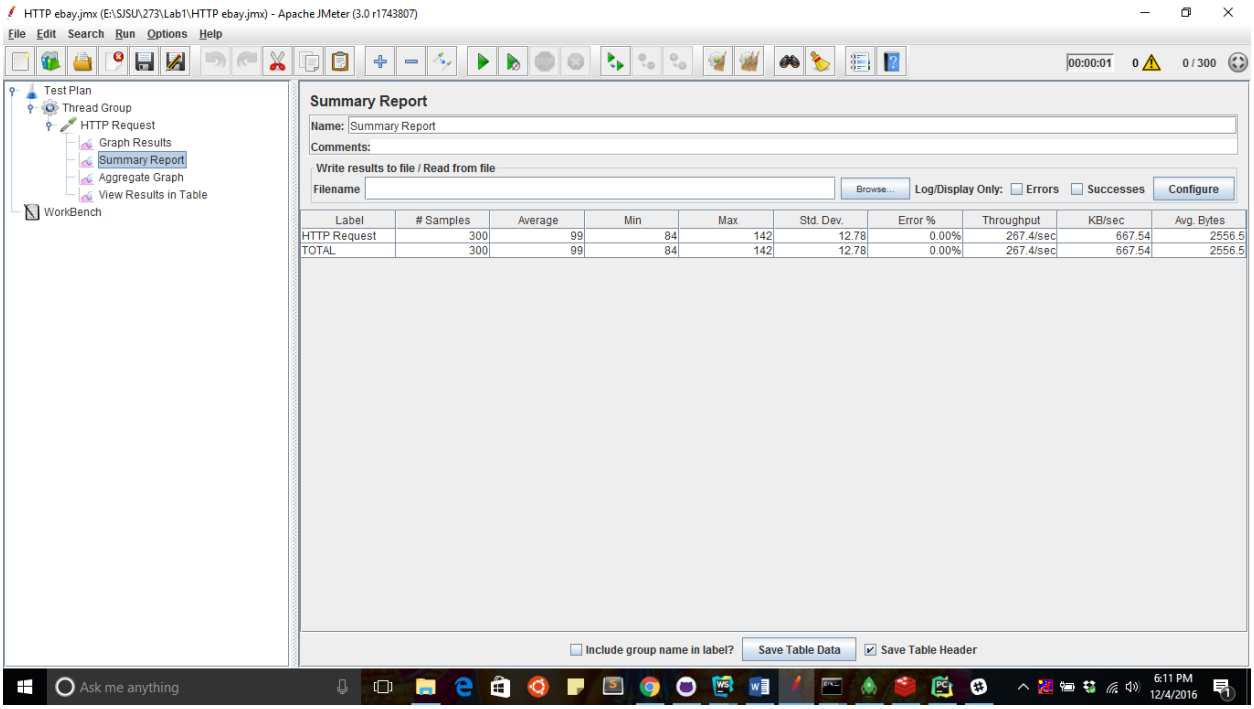
2)200 concurrent users



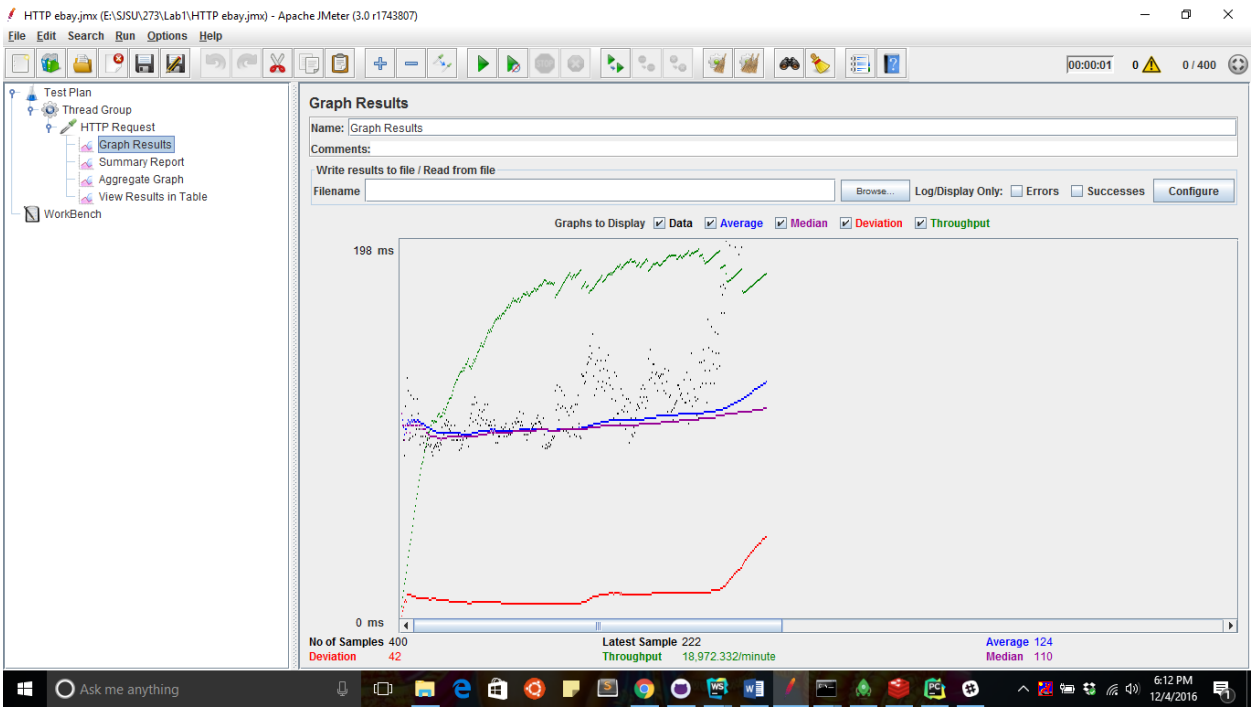


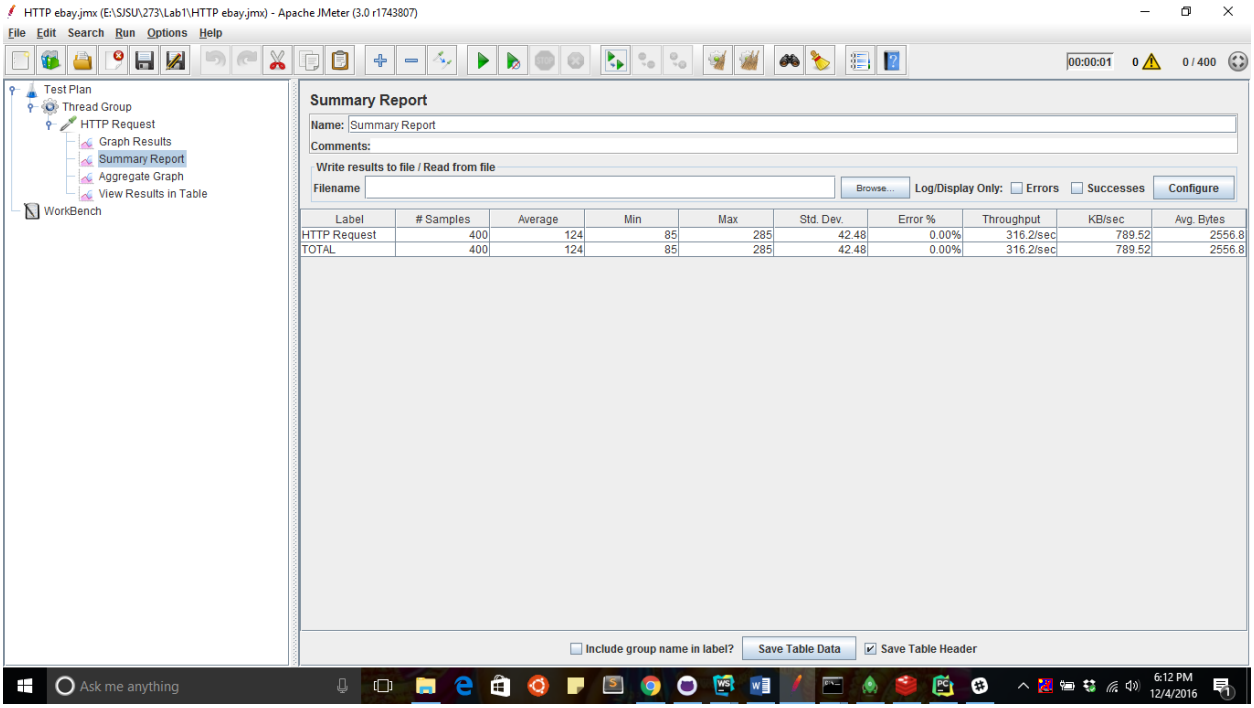
3)300 concurrent users



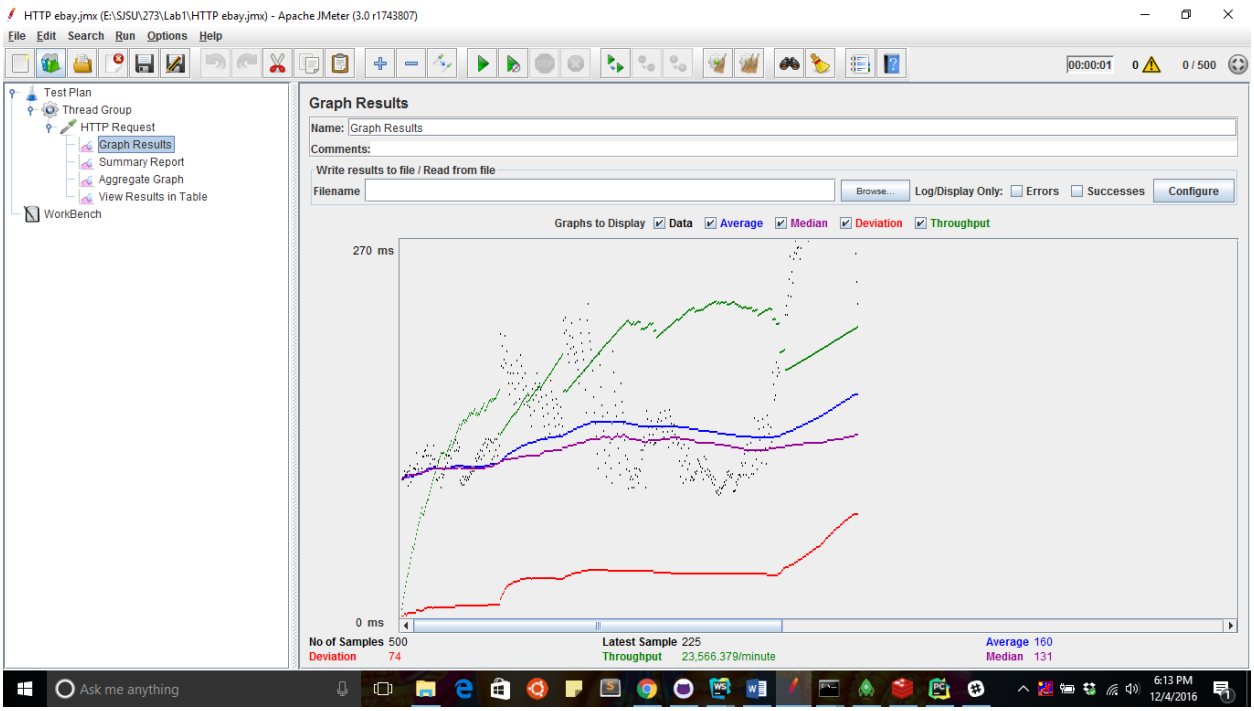


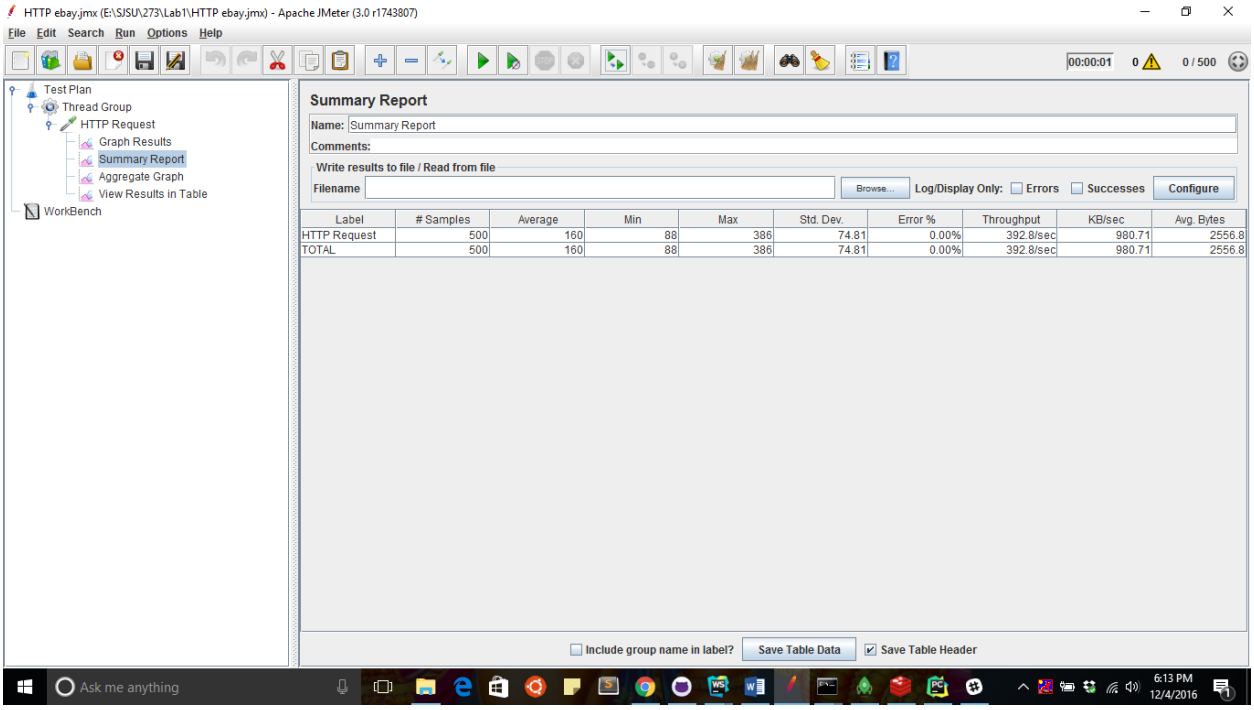
4)400 concurrent users





5)500 concurrent Users

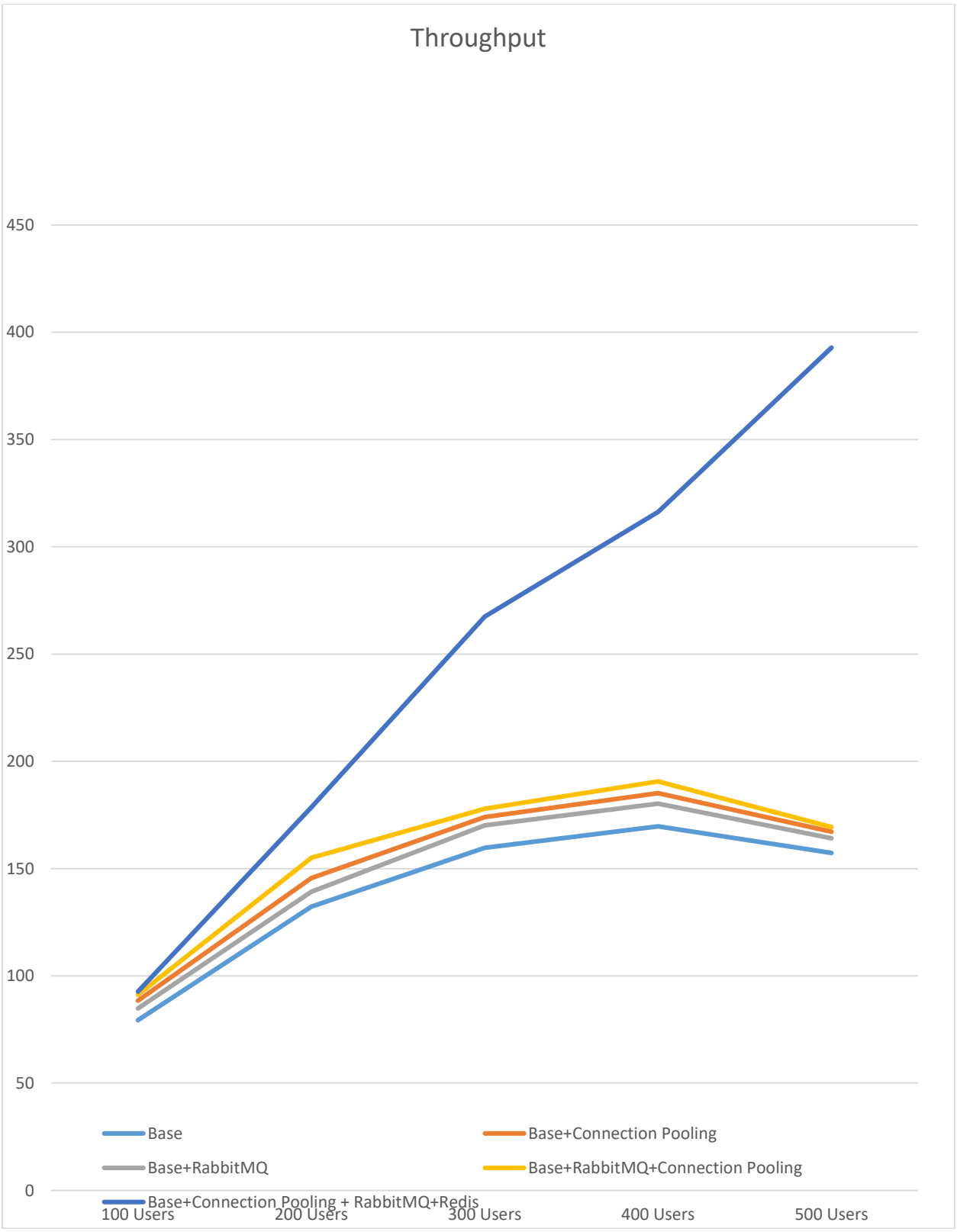




Performance Analysis

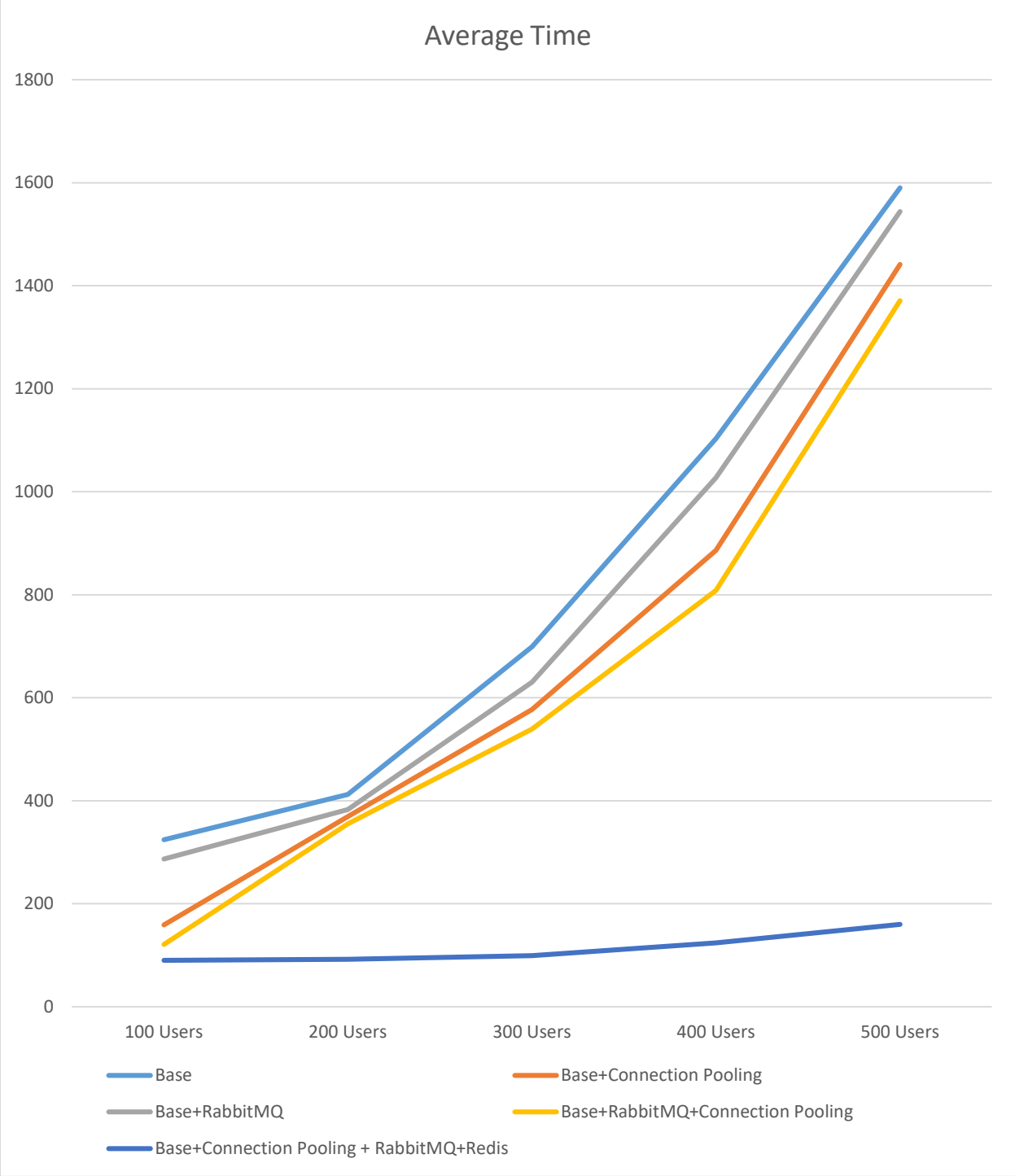
1. Throughput Analysis

-Y axis shows throughput per second
-X axis shows number of concurrent users


Throughput
450
400
350
300
250
200
150
100
50
0
Base
Base+Connection Pooling
Base+RabbitMQ
Base+RabbitMQ+Connection Pooling
Base+Connection Pooling + RabbitMQ+Redis
100 Users
200 Users
300 Users
400 Users
500 Users

2.Average Time Analysis

-Y axis shows Average Time in seconds
-X axis shows number of concurrent users



Observations and Lessons learnt from the Project

Airbnb Application helped us to develop an efficient, robust and a scalable web application.

- We developed RESTful stateless services using MEAN stack.
- The connection pooling in the MongoDB database helped us to optimize the performance.
- RabbitMQ helped us in providing scalability and handling multiple concurrent requests at the same time.
- We used Redis for cache management.
- We learned ELK stack to develop the Analytics module.
- JMeter and Mocha helped us to continuously analyze the performance in different scenarios for all the APIs.