SOFTWARE ENGINEERING DEPARTMENT

FALL 2016

SAN JOSÉ STATE UNIVERSITY

CMPE 273 – Enterprise Distributed Systems
Airbnb Simulation With Auction

Team Member	SJSU ID
Disha Sheth	011498240
Harsh Mehta	011429912
Manu Barsainyan	011457472
Bhavan Pandya	011456991
Aishwarya Keerty	011091574

CONTENTS

About Project

The project is a prototype of Airbnb, a web portal for booking a house on-the-go.

About Airbnb

Airbnb is an online marketplace that enables people to list, find, then rent vacation homes for a processing fee. It has over 2,000,000 listings in 34,000 cities and 191 countries. Founded in August 2008 and headquartered in San Francisco California.

Shortly after moving to San Francisco in October 2007, Brian Chesky and Joe Gebbia Created the initial concept for AirBnb & Breakfast during the Industrial Design Conference held by Industrial Designers Society of America. The original site offered short-term living quarters, breakfast, and a unique business networking opportunity for those who were unable to book a hotel in the saturated market.

Airbnb is a peer-to-peer accommodation marketplace that connects hosts (vendors of rooms/accommodations) and travelers via its website. Airbnb enables transactions between these two entities by charging a 'service fee' without directly owning any room by itself.

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.

Individual Member Contributions:

1. **Disha Sheth**
   - Front end and back end development of user dashboard,listings and trips.
   - Home page
   - Search implementation
   - Back end development of Trips module/services.
   - Implementation of RabbitMQ.
   - Designing of database.
   - JMeter Testing.

2. **Harsh Mehta**
   - Implementation of UI Routing.
   - Front end and back end development of property listing.
   - Implementation of checkout.
   - Front end development of Trips module/services.
   - Integration of different modules/services.
   - Implementation of RabbitMQ.
   - Implementation of Redis.
   - Addition of multiple users, hosts and billing.

3. **Manu Barsainyan :**
   - Front end and back end development of billing module/services.
   - Front end and back end development of host dashboard, property listing, image and video upload module/services.
   - Front end and back end development of bidding module/services.
   - Front end and back end development of review module/services.
   - Implementation of RabbitMQ.
   - Mocha Testing.

4. **Bhavan Pandya:**
   - Front end and back end design and development of Admin module.
   - Front end and back end design and development of Admin and Host anlytics module.
   - Generating logic for log analysis.
   - Implementation of RabbitMQ.
   - Designing of database.

5. **Aishwarya Keerty:**
    - Front end and back end design and development of login and register modules/services.
    - Generation of Project Report.

Object Management Policy

1. Requirement Analysis:

Depending on the analysis of the requirements of the project, there are 3 tiers. They are:

a) **The Client tier:** In this tier, the user will interact with the system.

b) **The Middle tier/middleware/messaging system:** In this tier majority of the processing takes place. This tier supports the sending and receiving of messages between distributed systems. Some of the modules in this tier have been implemented using RabbitMQ which is an open source message broker software written in Erlang programming language that implements Advanced Message Queuing Protocol (AMQP).

c) **The Third tier:** This tier comprises of a database to store the state of the entity objects of the system. Some modules in this tier use MySQL as the relational database while others use MongoDB as the NoSQL database to store data.

2. Functional Requirements Considerations:

In this project different modules such as user module, host module, billing module, admin module, trips module have been implemented. Each of the modules has an associated schema which shows how data is stored in the database.

3. Development:

Some of the key features in the project like add property helps user select property in their location and within required price range. Other features like ratings and reviews allows the host and client to rate their experiences. RabbitMQ helps to increase the sustainability when the load is high.

4. Testing the System:

a) Handling “Heavy Weight” resources:
Loading heavyweight resources like images and videos takes a significant of time. Hence, these heavyweight resources are stored in MongoDB. Retrieving images and videos from MongoDB is much more time efficient than retrieving them from MySQL.

b) Policy to decide when to write data into the database:

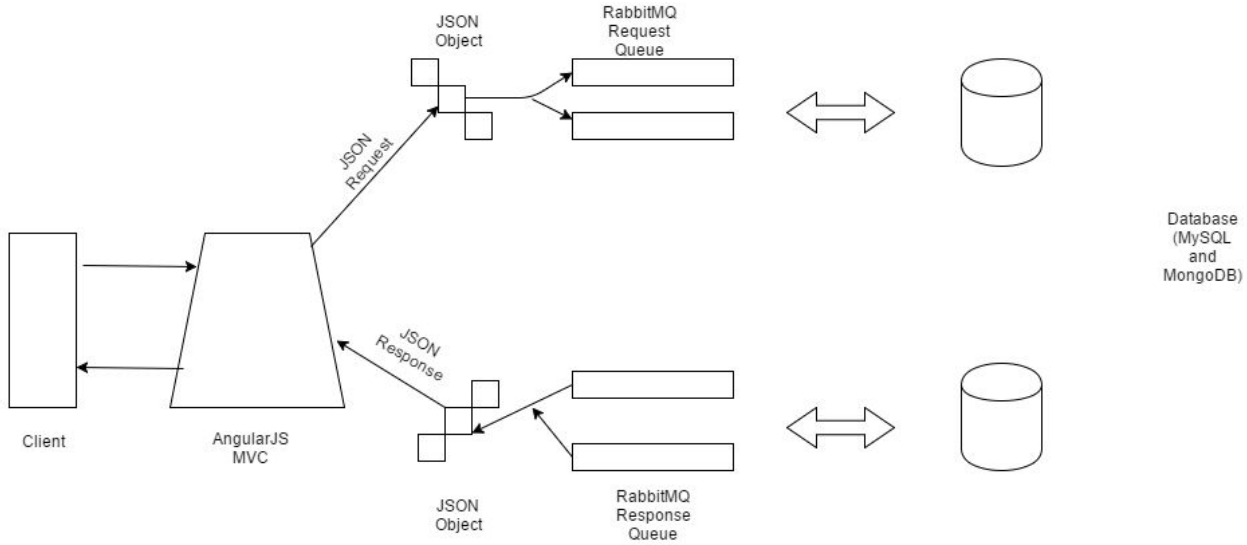
Data has been inserted into database optimally. Mongo has been used to store data like ratings, reviews, property details and sessions. Data relating to User, host, admin and billing details have been stored in MySql. Introduction videos and host images have also been stored in MongoDB for faster query access.

Goal of Project

1) We have to understand the functionality of the REST API and implement Group Project using the nodejs, angularjs, HTML5 and bootstrap.
2) Develop an attitude of working in team and developing a scalable enterprise level application.
3) Understand, optimise and handle the functioning of the actual enterprise application.

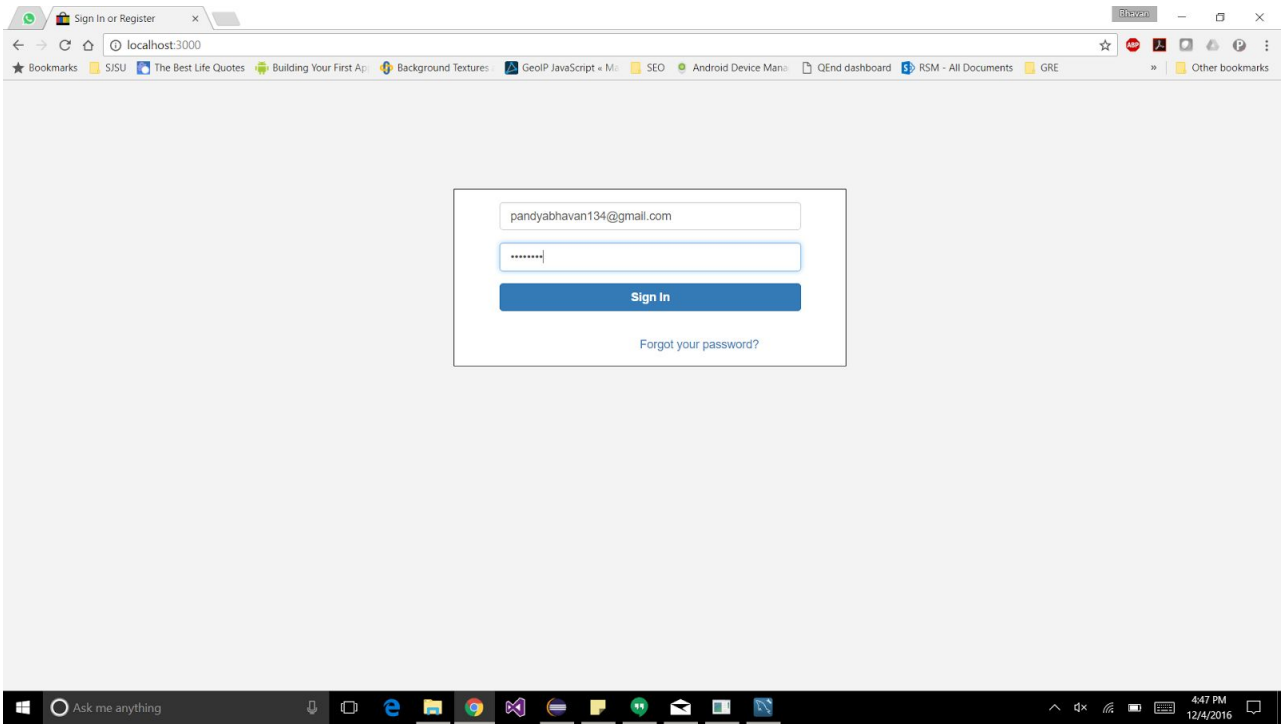
Architecture of Application

Entire n-tier application is developed using MEAN(MongoDB, Express, AngularJS and NodeJS) which is further optimised using MySQL, connection pooling, RabbitMQ and Redis Caching.

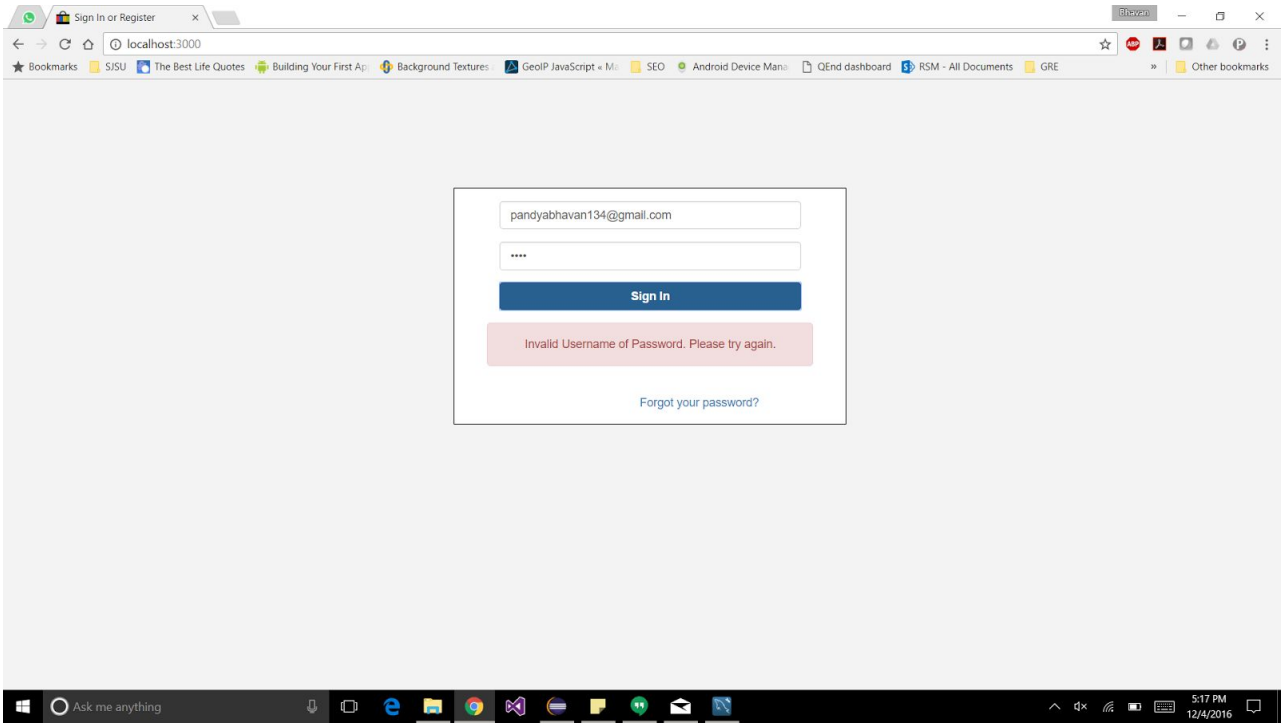


Screen Captures: Various UI Pages

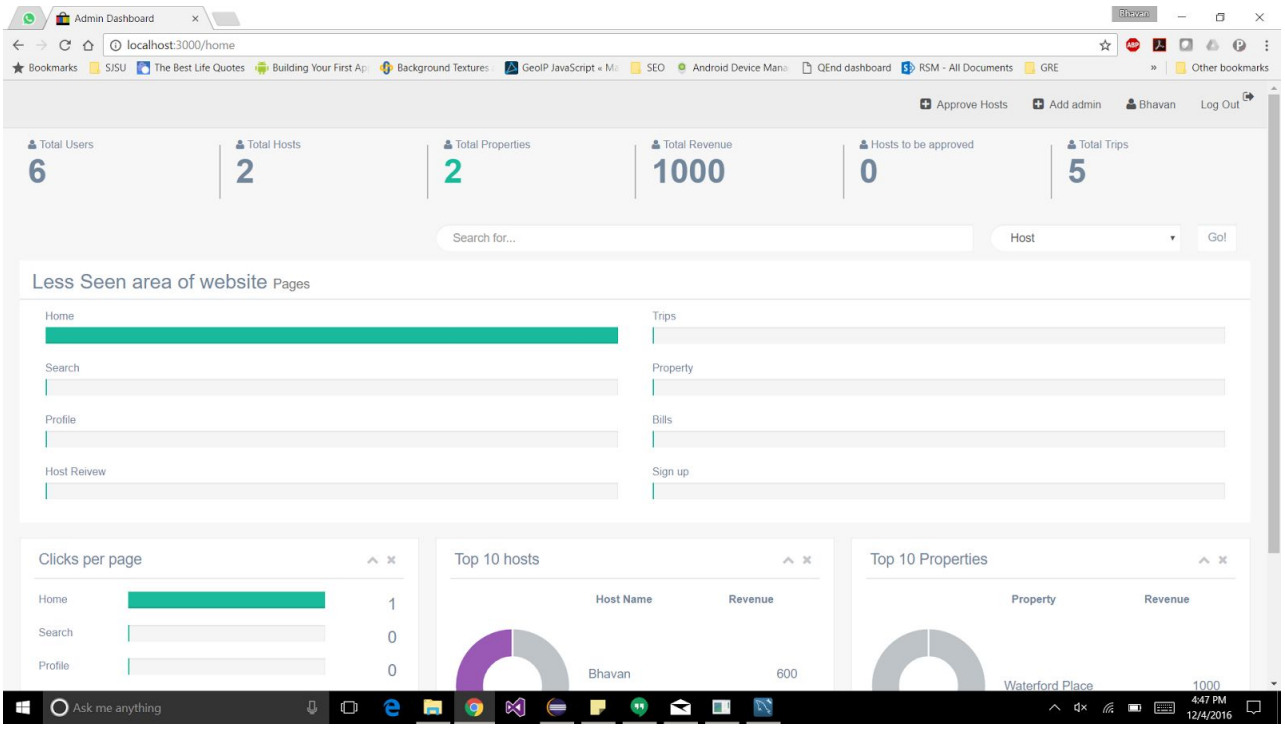
1. Admin Login Page: The admin(s) can log in using their email id and password.

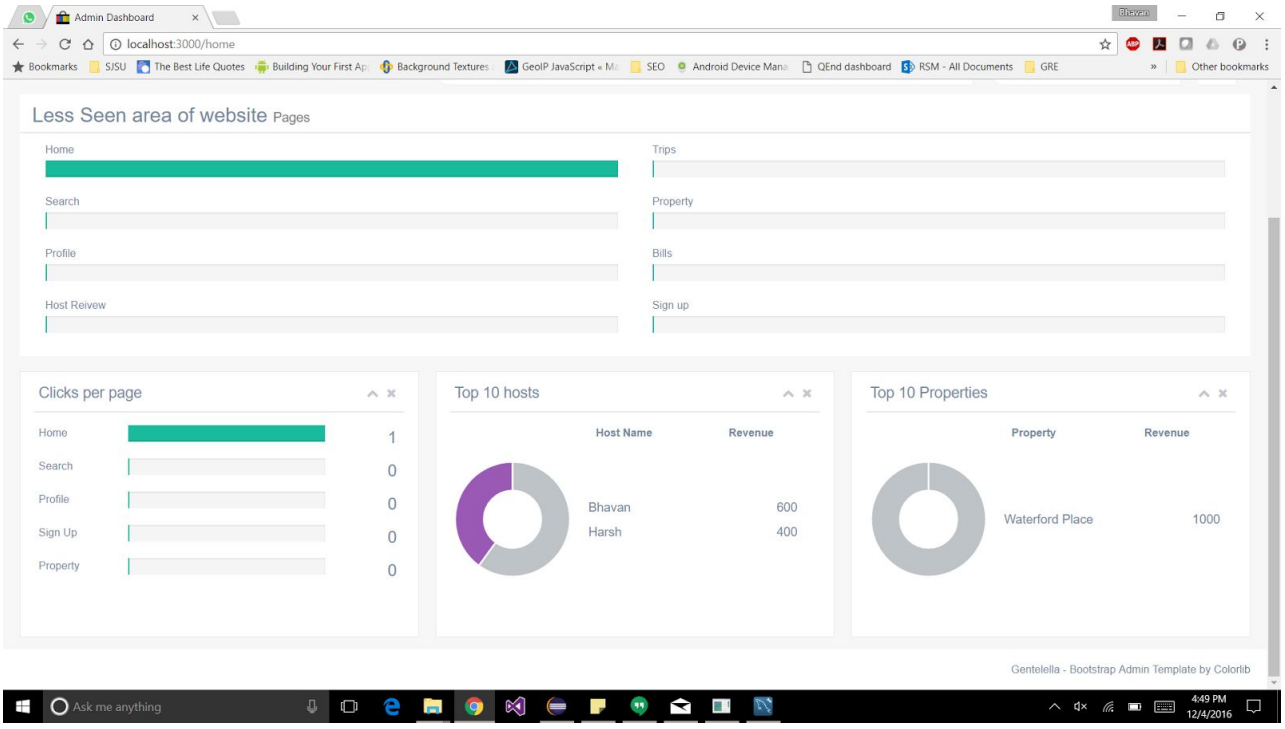


2.Invalid admin validation: Users with invalid details will be displayed error message.

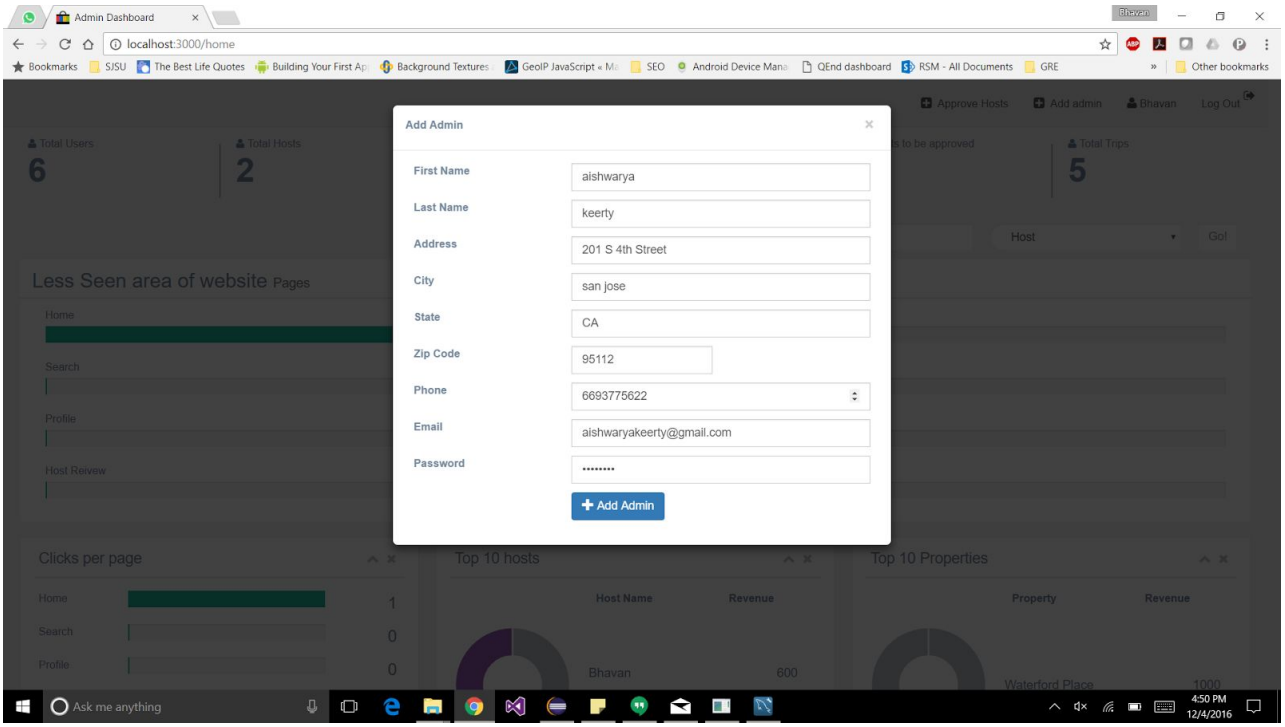


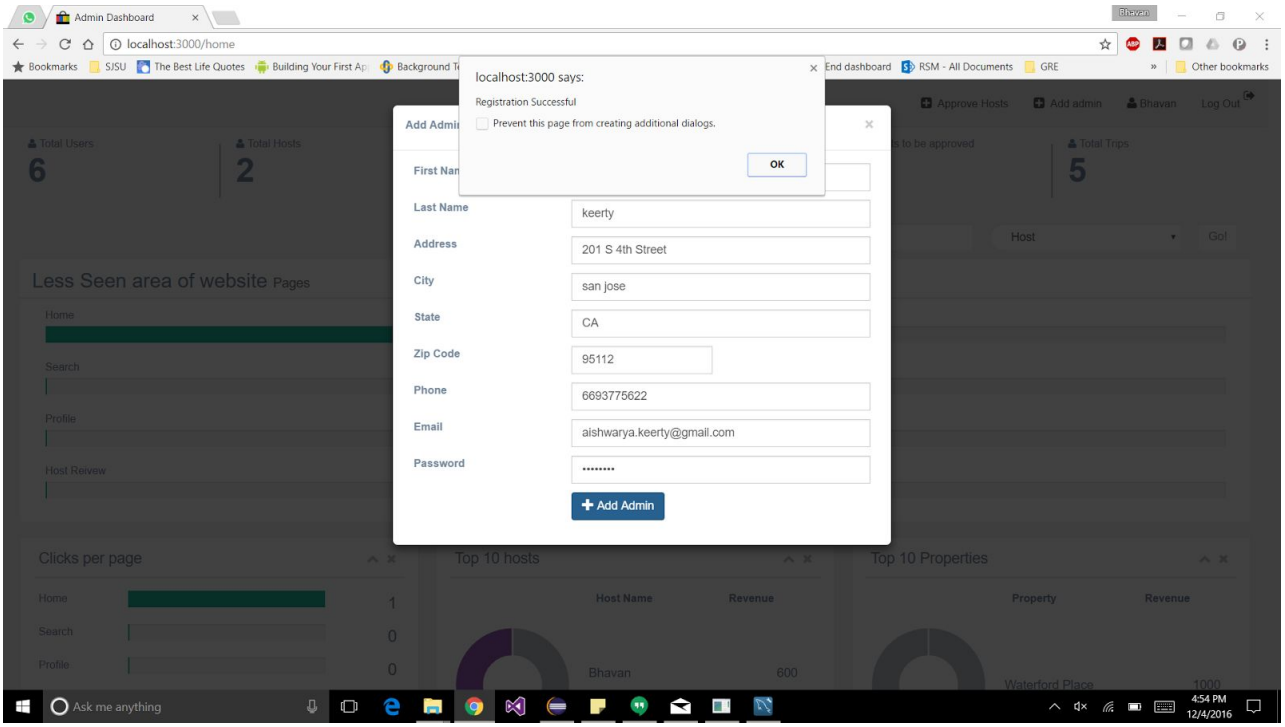
3.Admin's Dashboard: The admin dashboard allows admin to approve hosts, add other admins, check the total revenue, trips, properties, users, clicks per page etc.



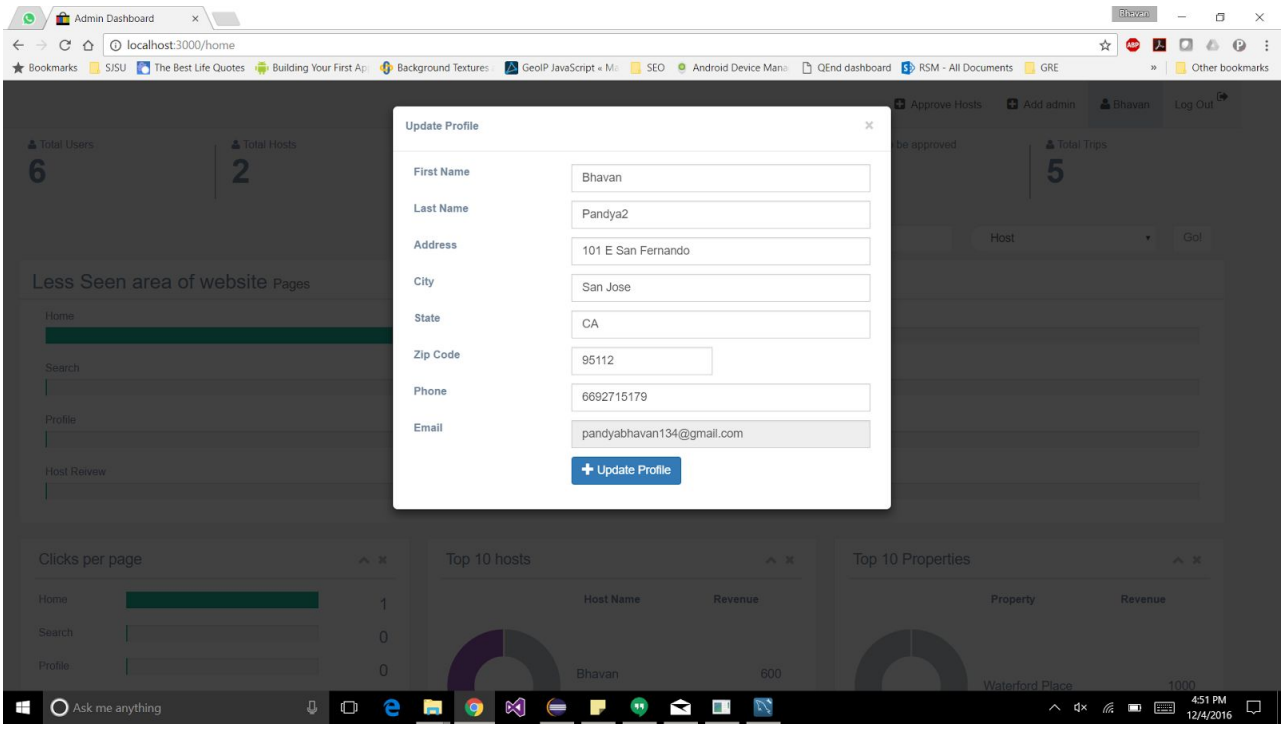


4. Add Admin: An admin can add one or more admins for the website.

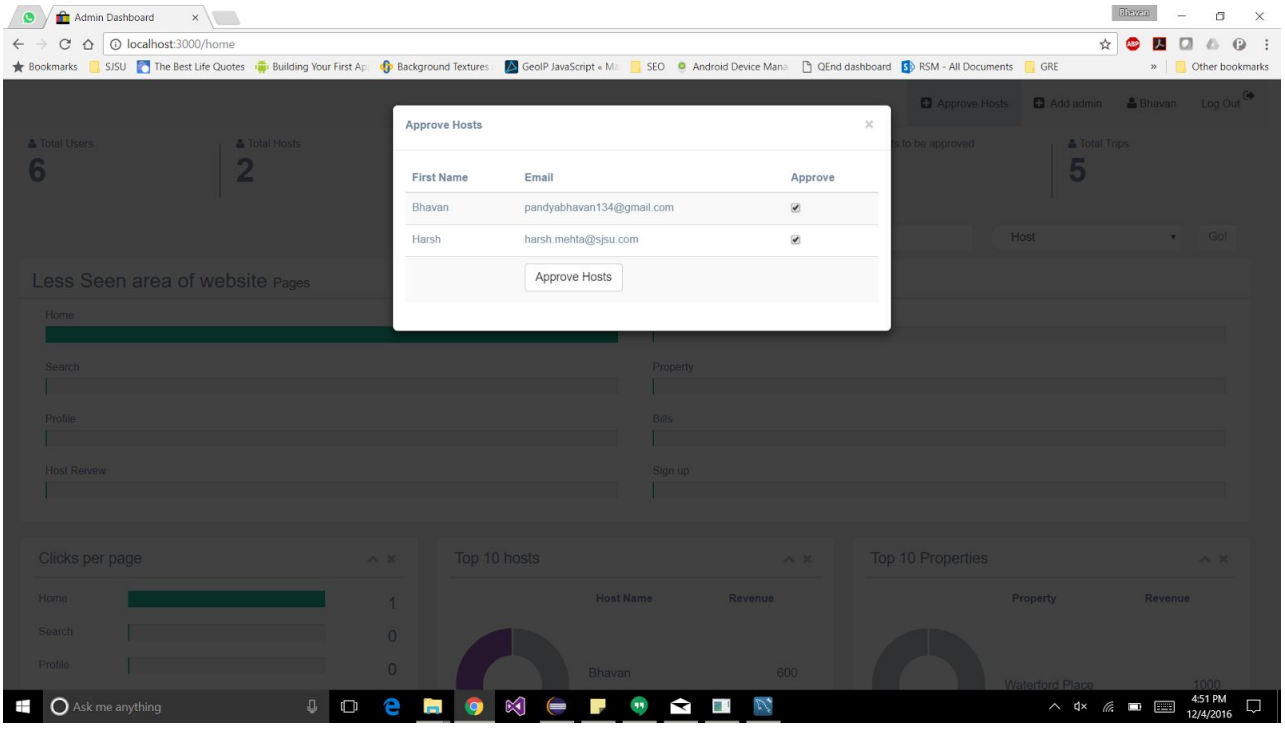


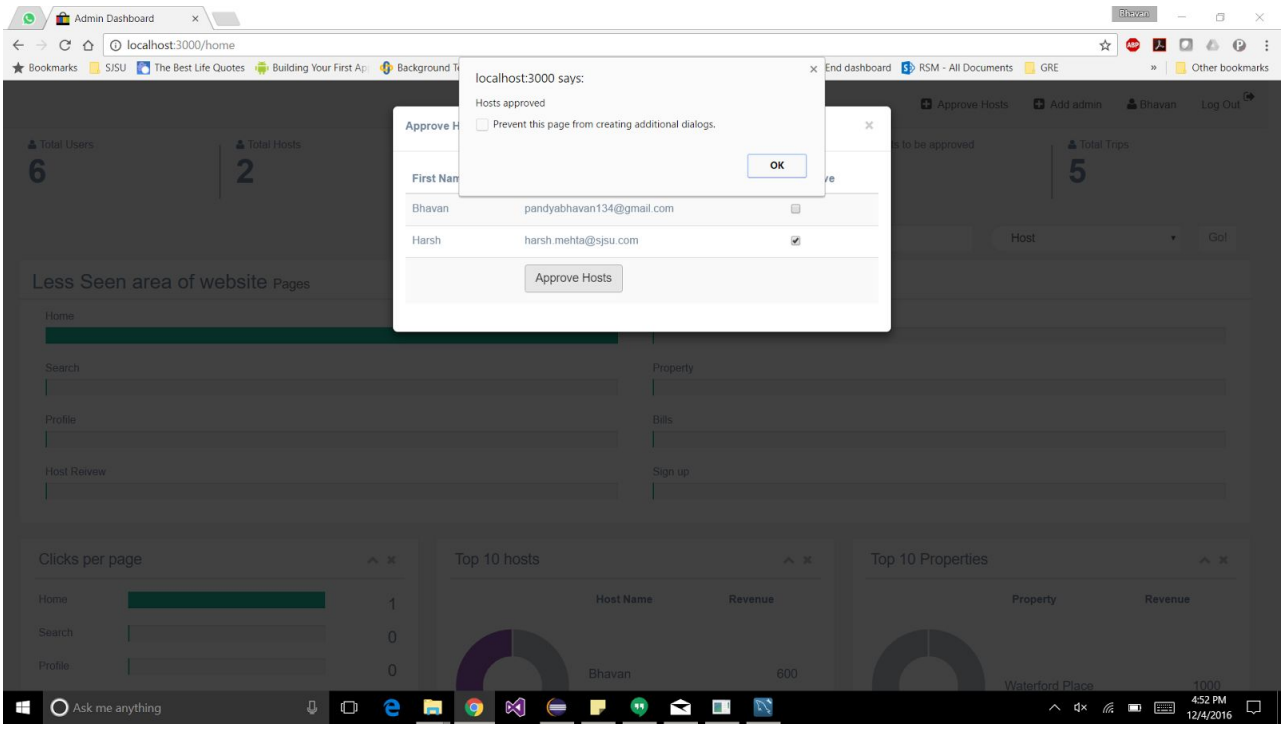


5.Update Profile: An admin can update his profile.

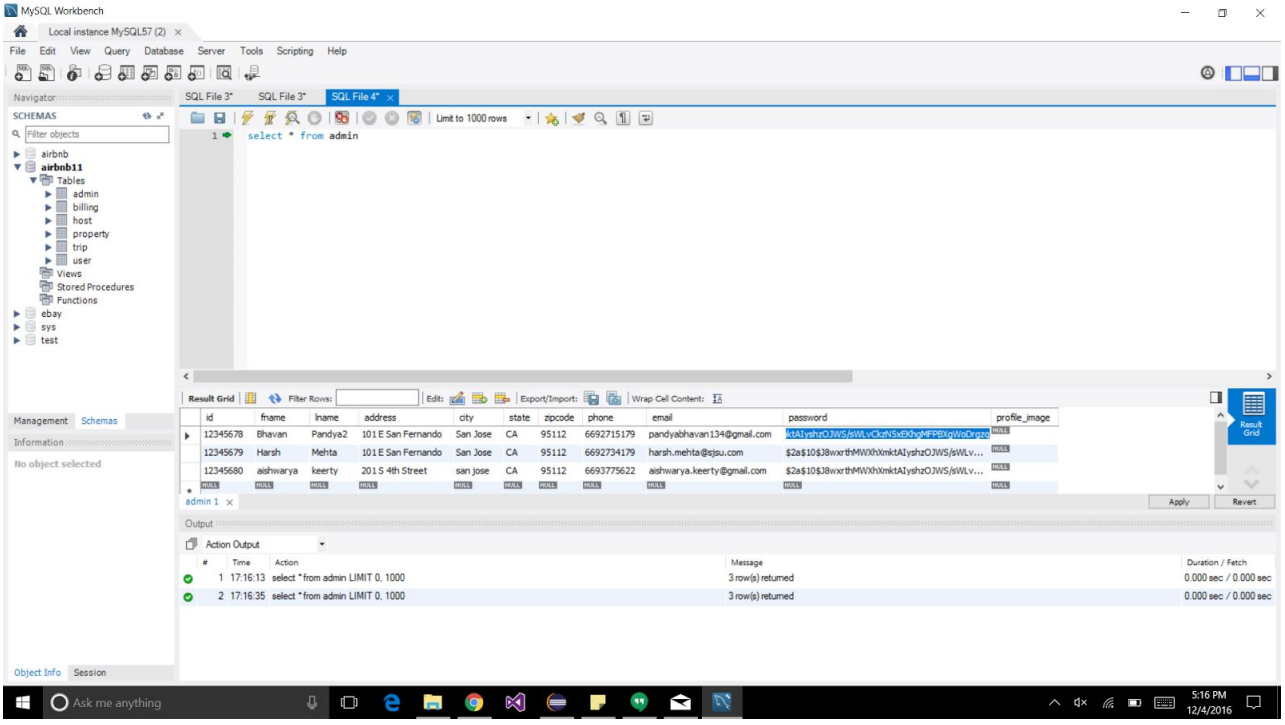


6.Approve Hosts: The admin can approve pending Host requests.

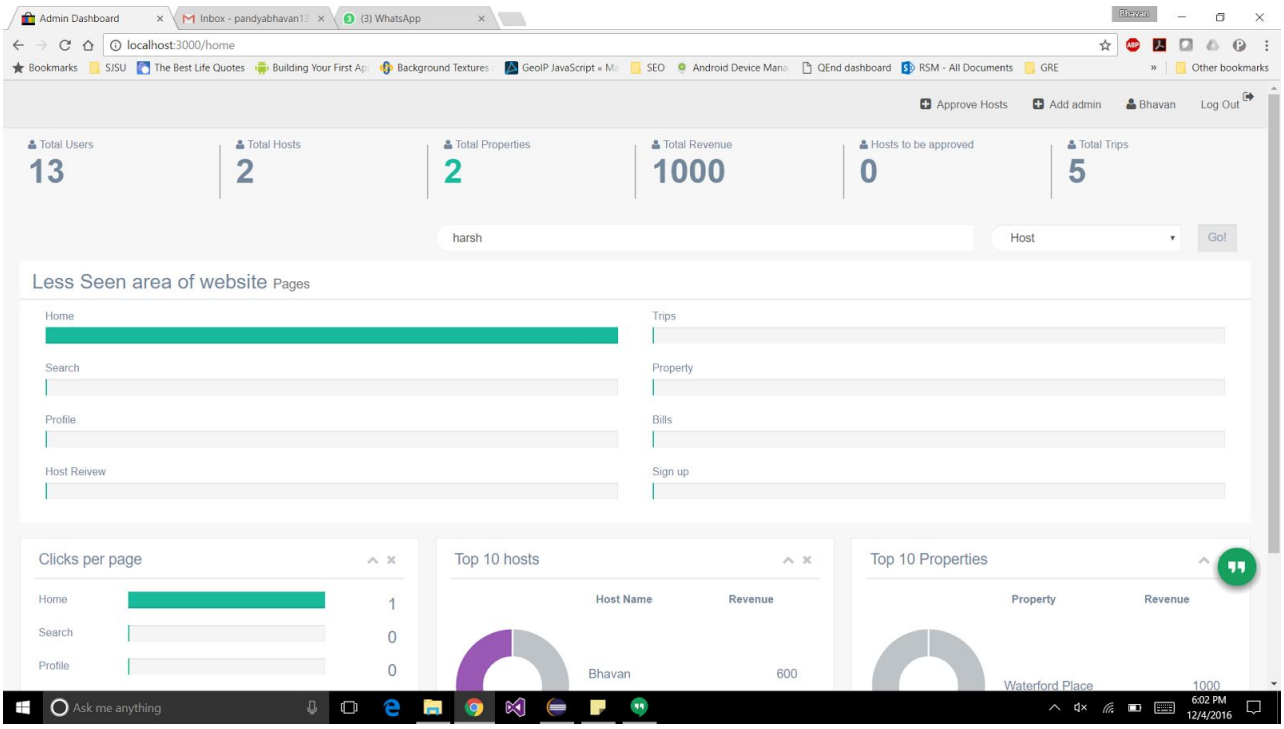




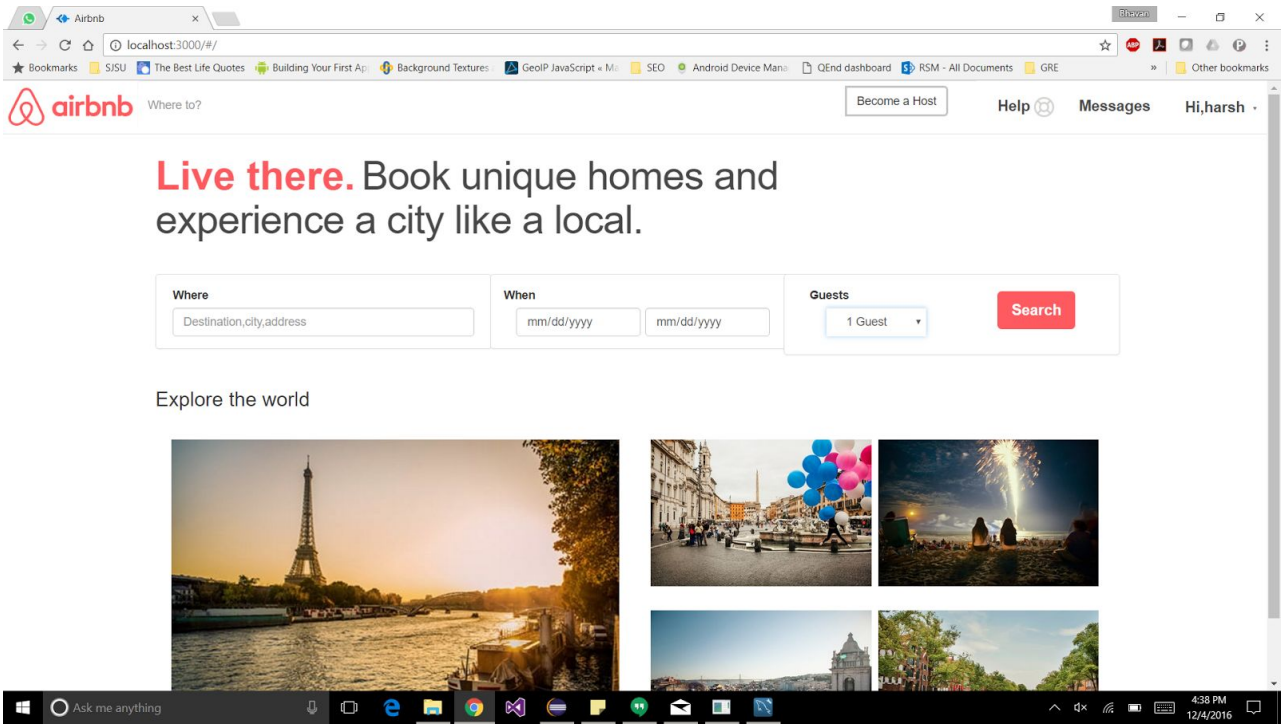
7.Admin Database:

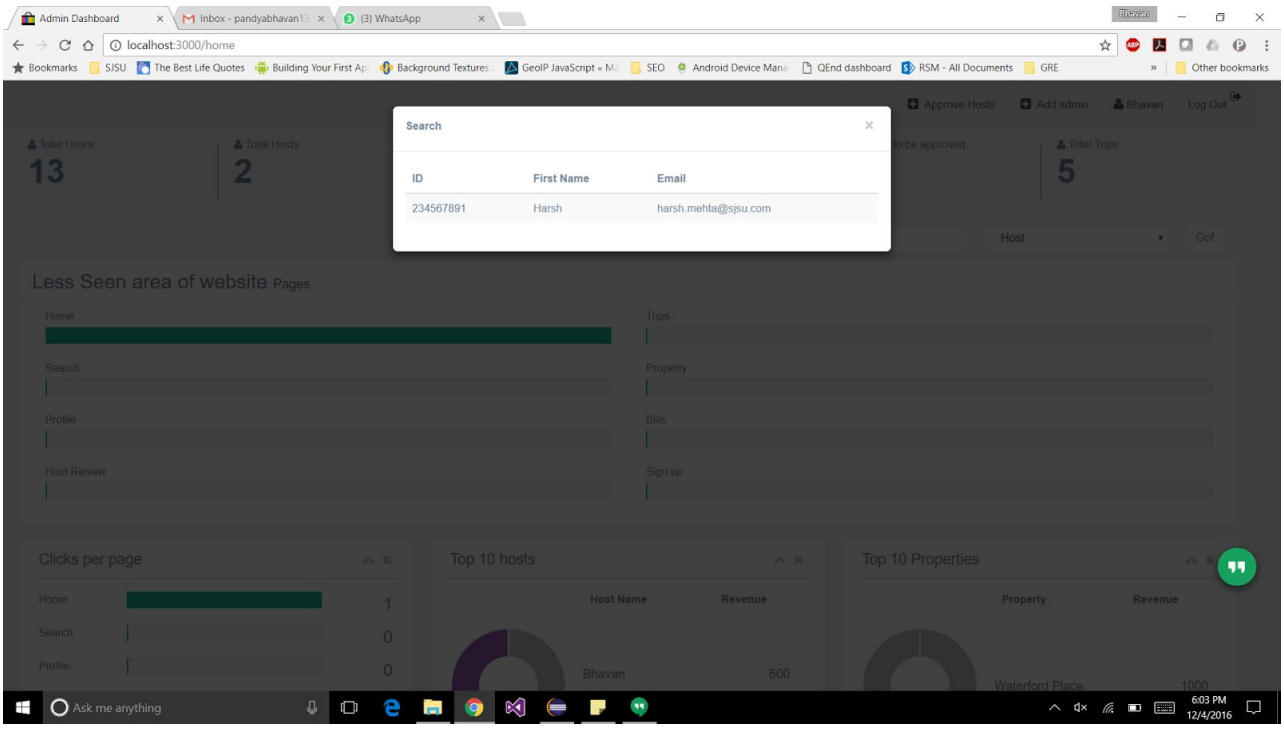


8.Search: The admin can search for hosts and bills in the admin dashboard.

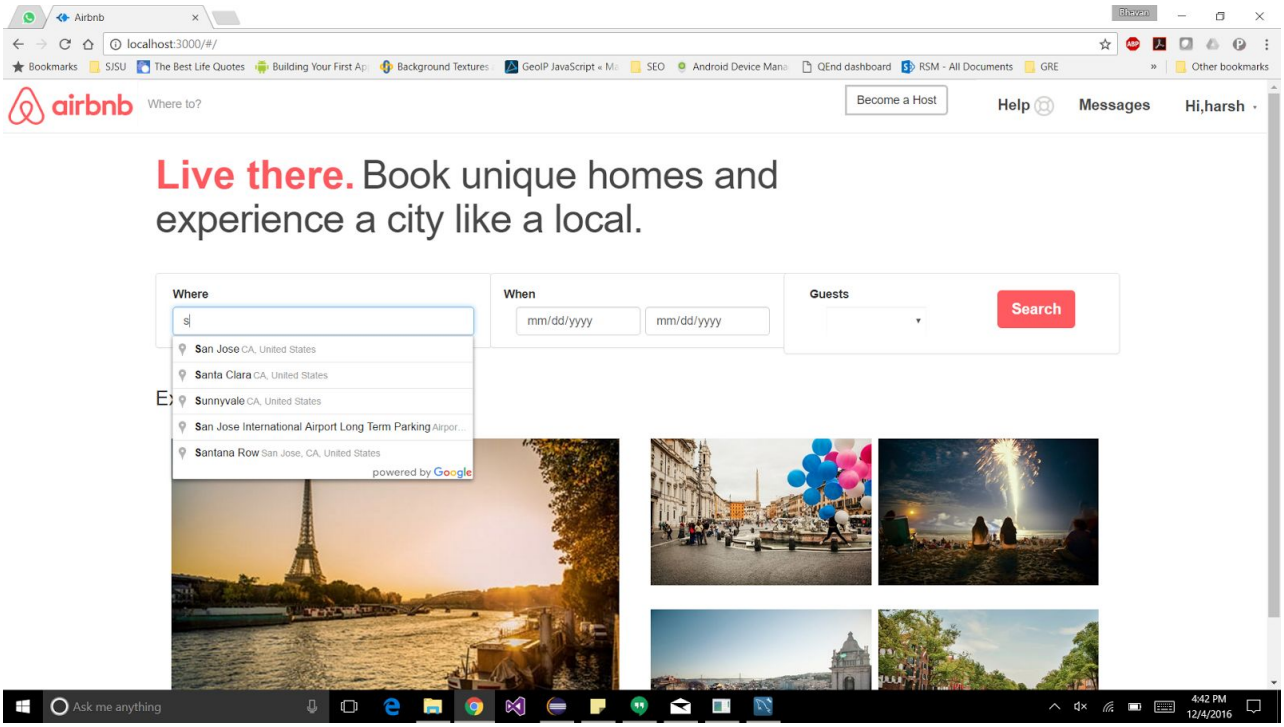


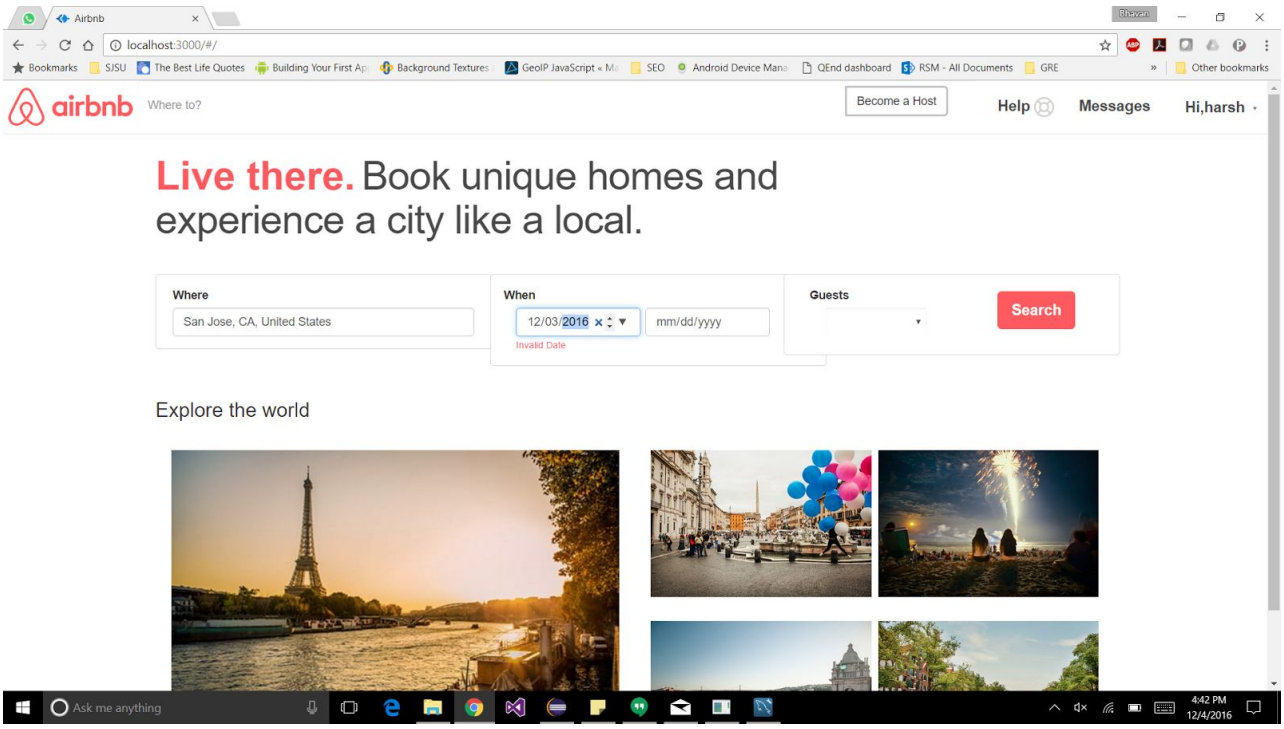
9. Home Page: The home page allows the login and signup of users and hosts.

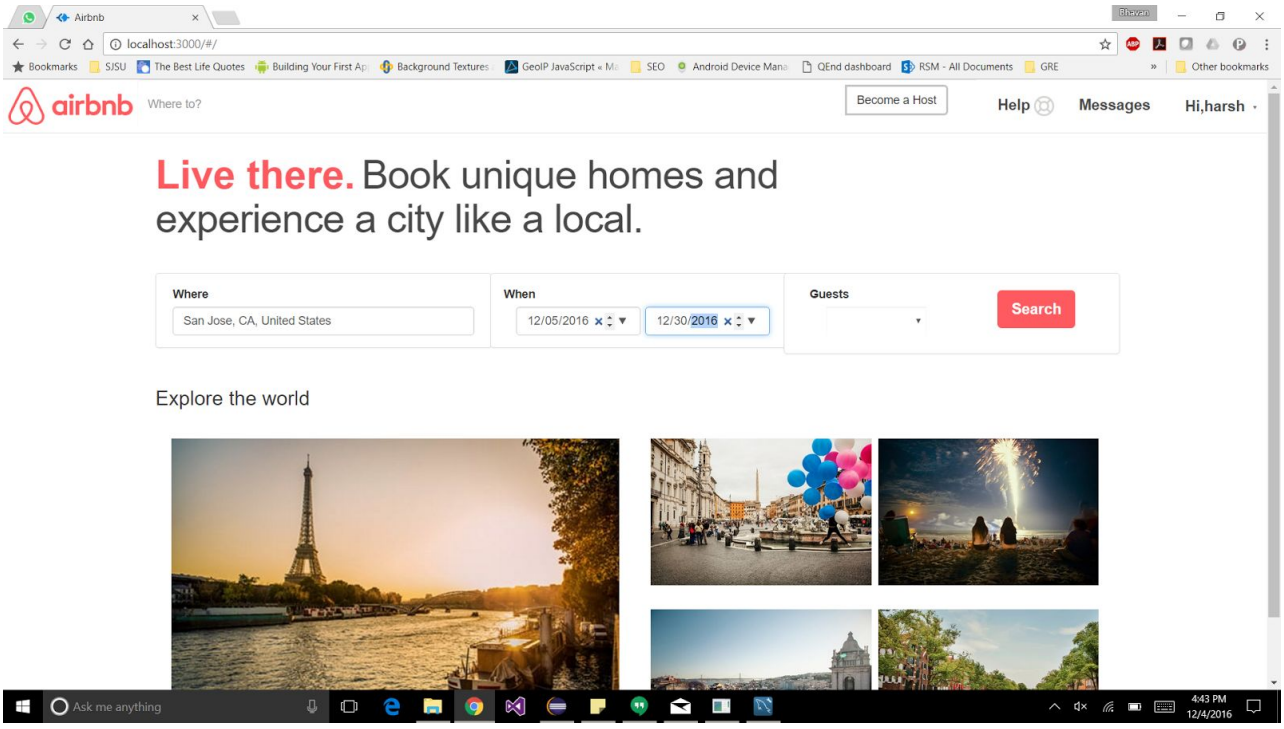


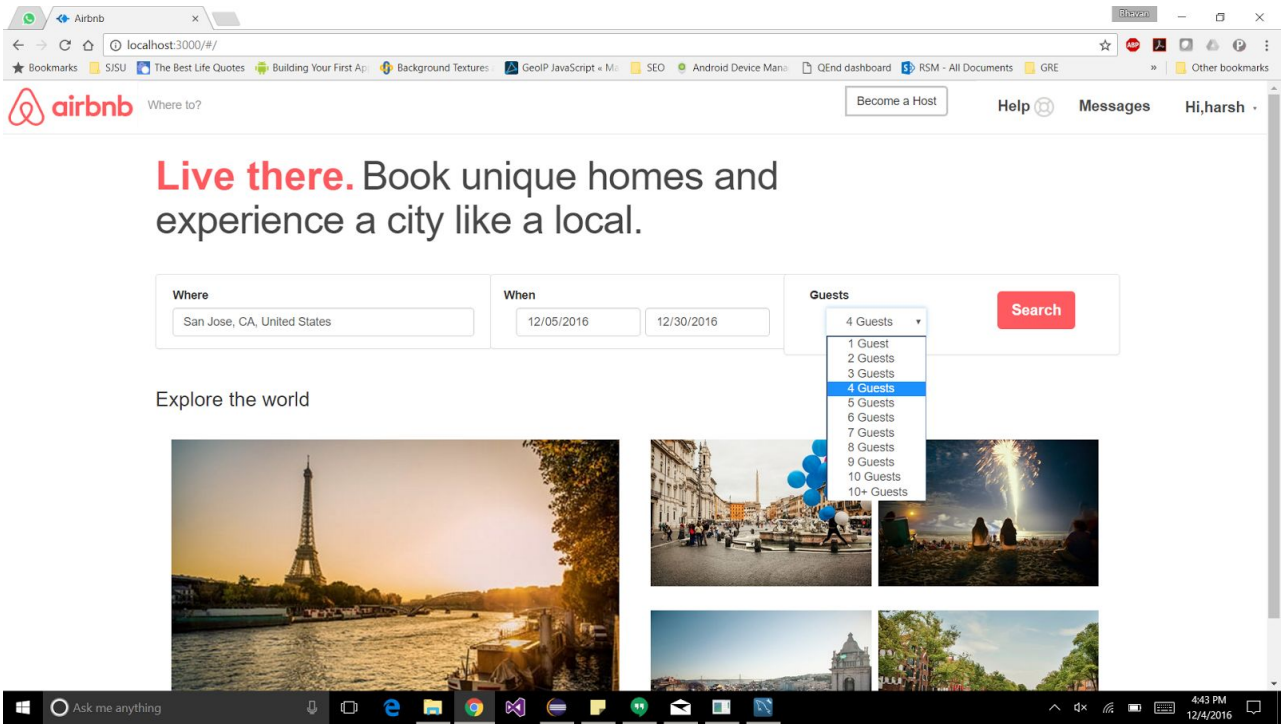


10. Home page validations:

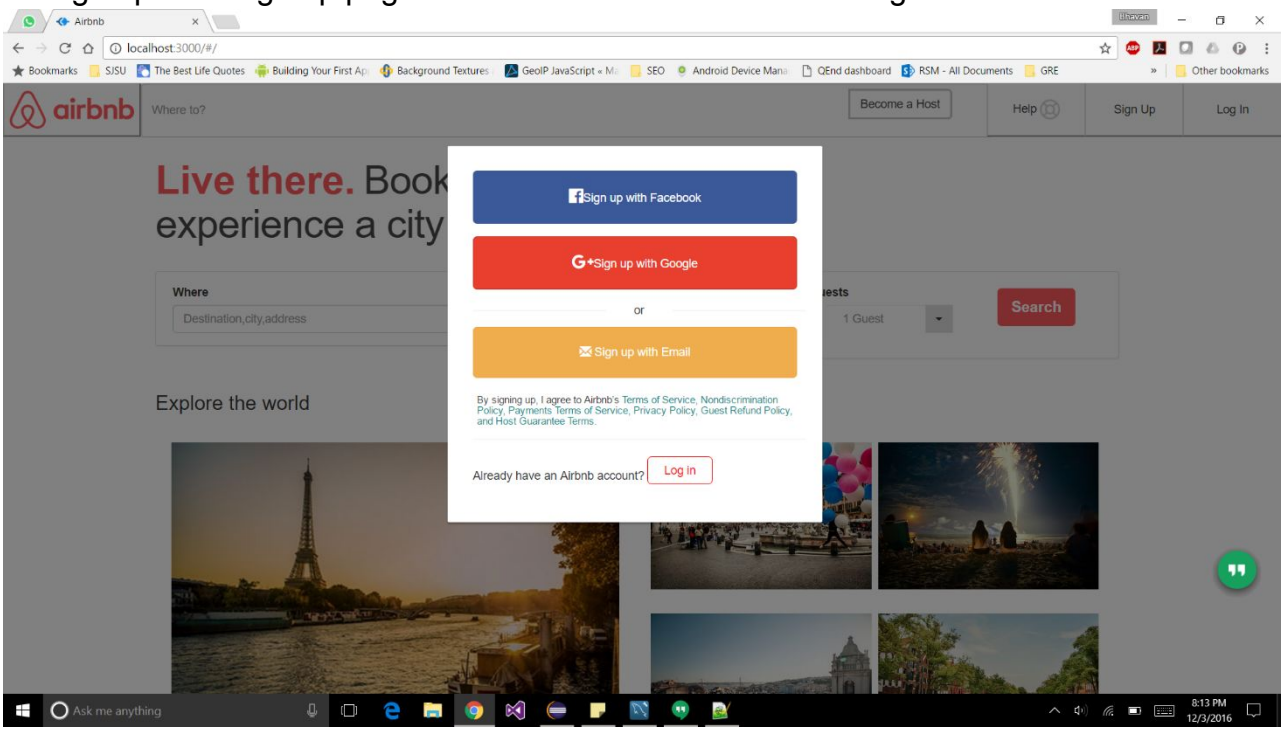



Airbnb
localhost:3000/#/
airbnb
Where to?
Become a Host
Help
Messages
Hi,harsh
Live there. Book unique homes and experience a city like a local.
Where
San Jose, CA, United States
When
12/03/2016
mm/dd/yyyy
Invalid Date
Guests
Search
Explore the world
Ask me anything
4:42 PM
12/4/2016

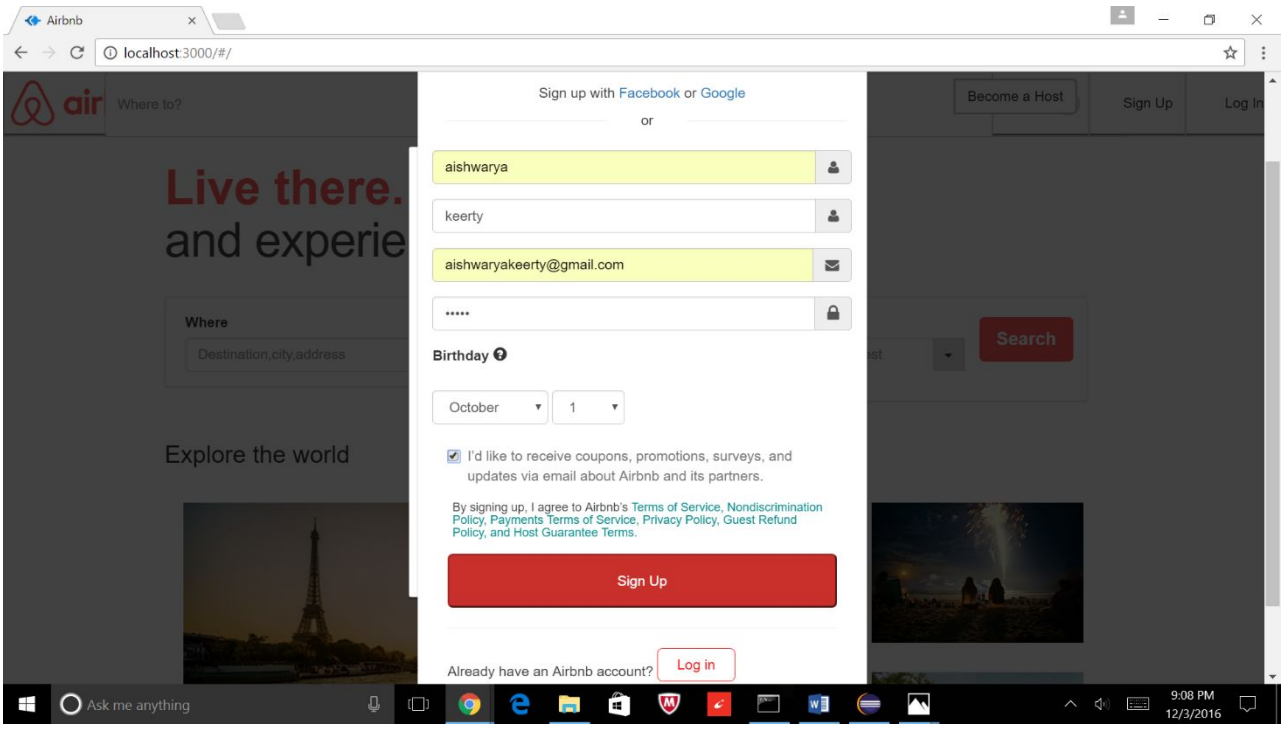

Airbnb
localhost:3000/#/
airbnb
Where to?
Become a Host
Help
Messages
Hi,harsh
Live there. Book unique homes and experience a city like a local.
Where
San Jose, CA, United States
When
12/05/2016
12/30/2016
Guests
Search
Explore the world
Ask me anything
4:43 PM
12/4/2016



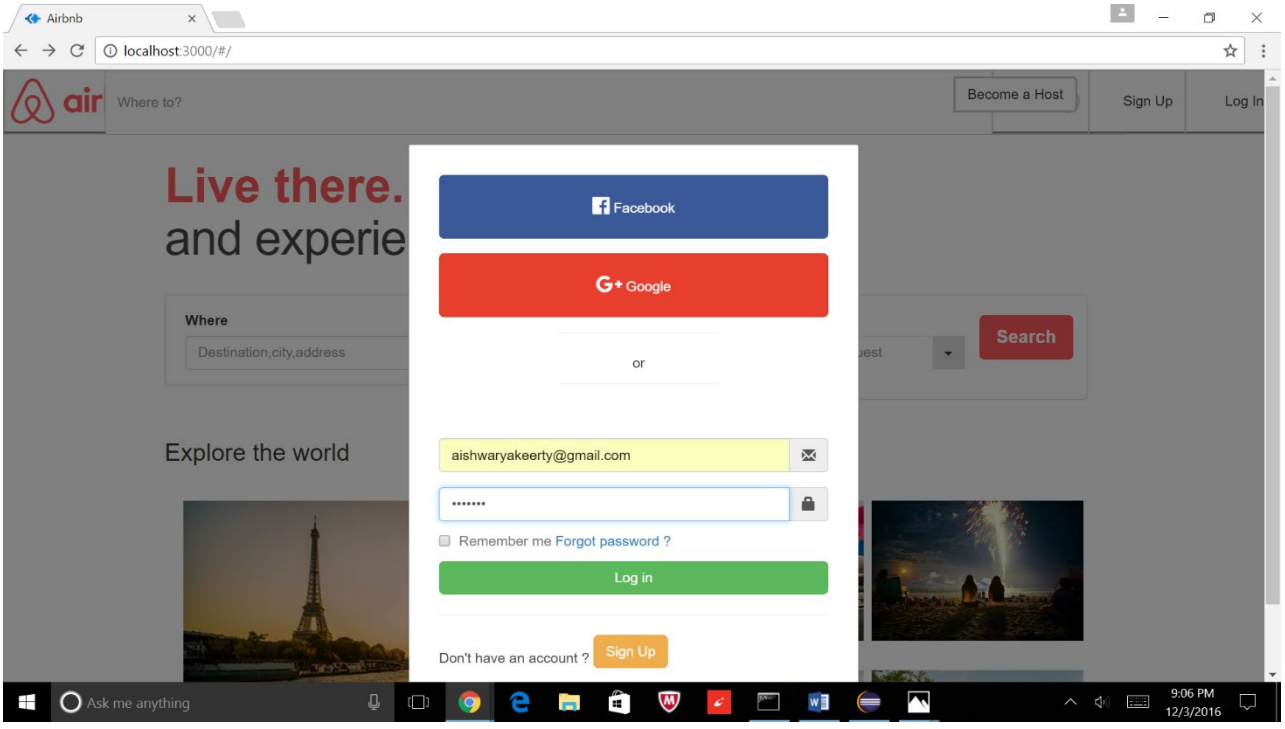
11 Sign up: The sign up page allows new hosts and users to register.



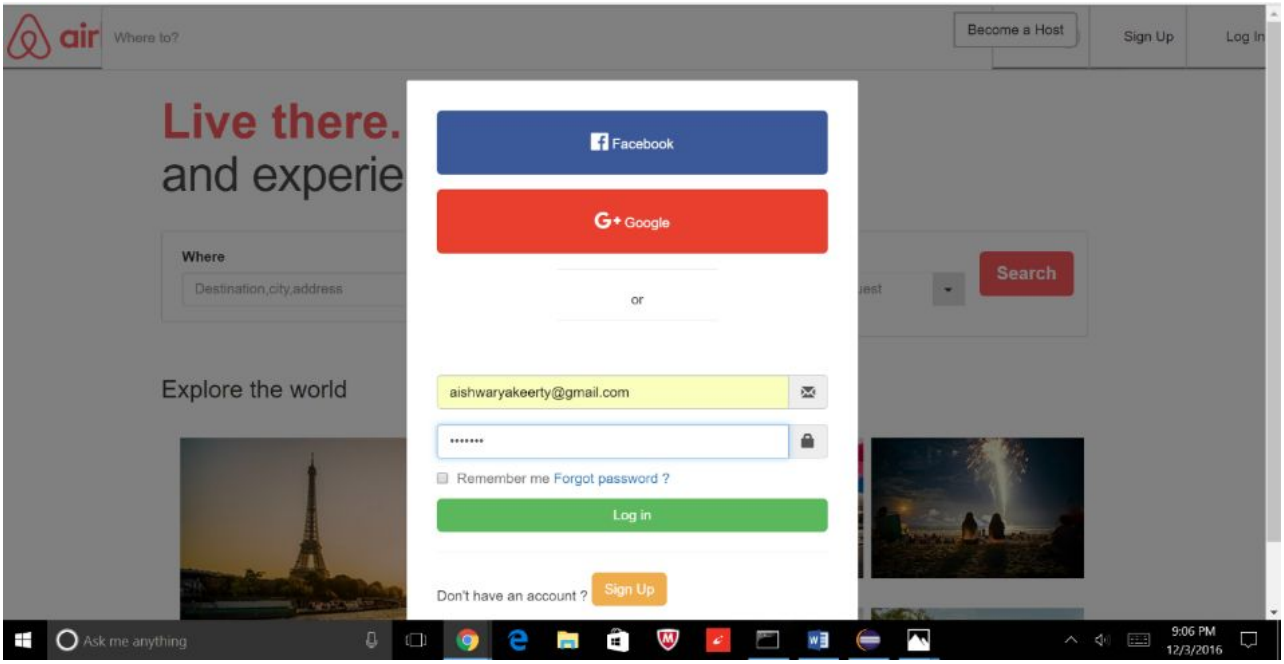
12.Sign up with Email: This allows the user/ host to sign up using Email.



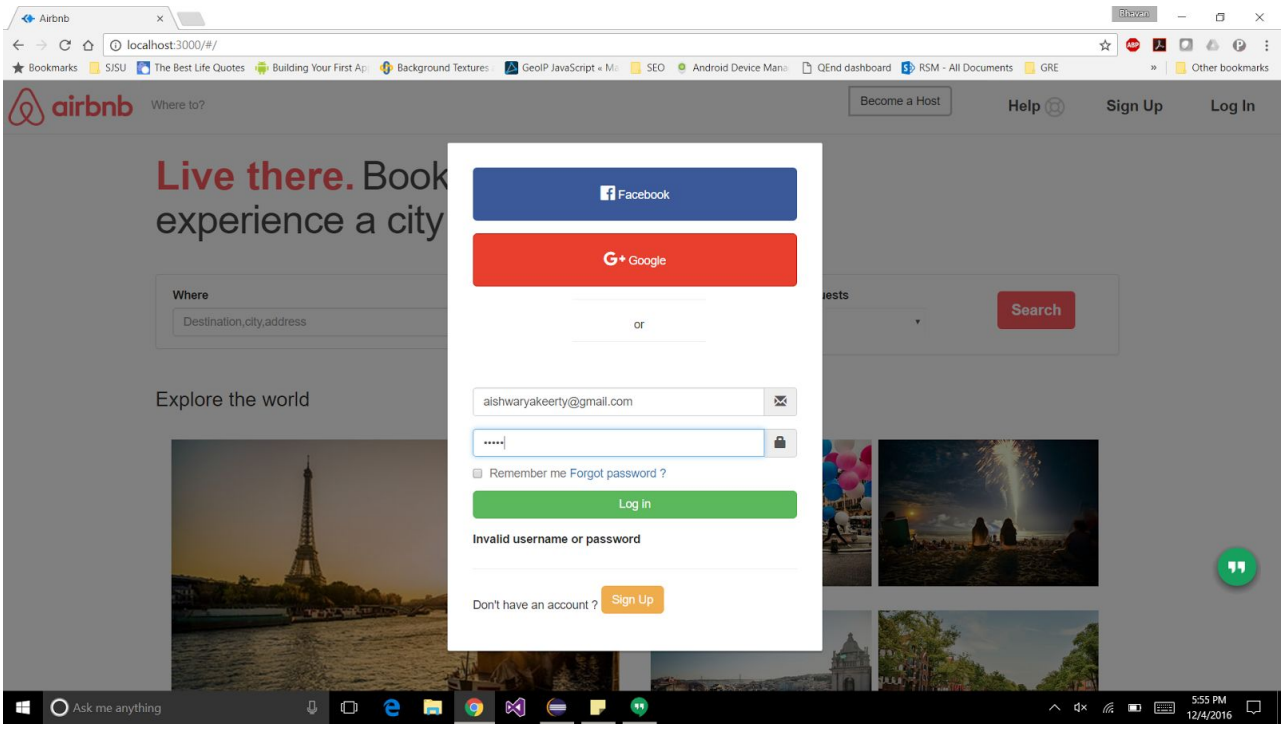
13.Login: The log in page allows existing user and hosts to log in.



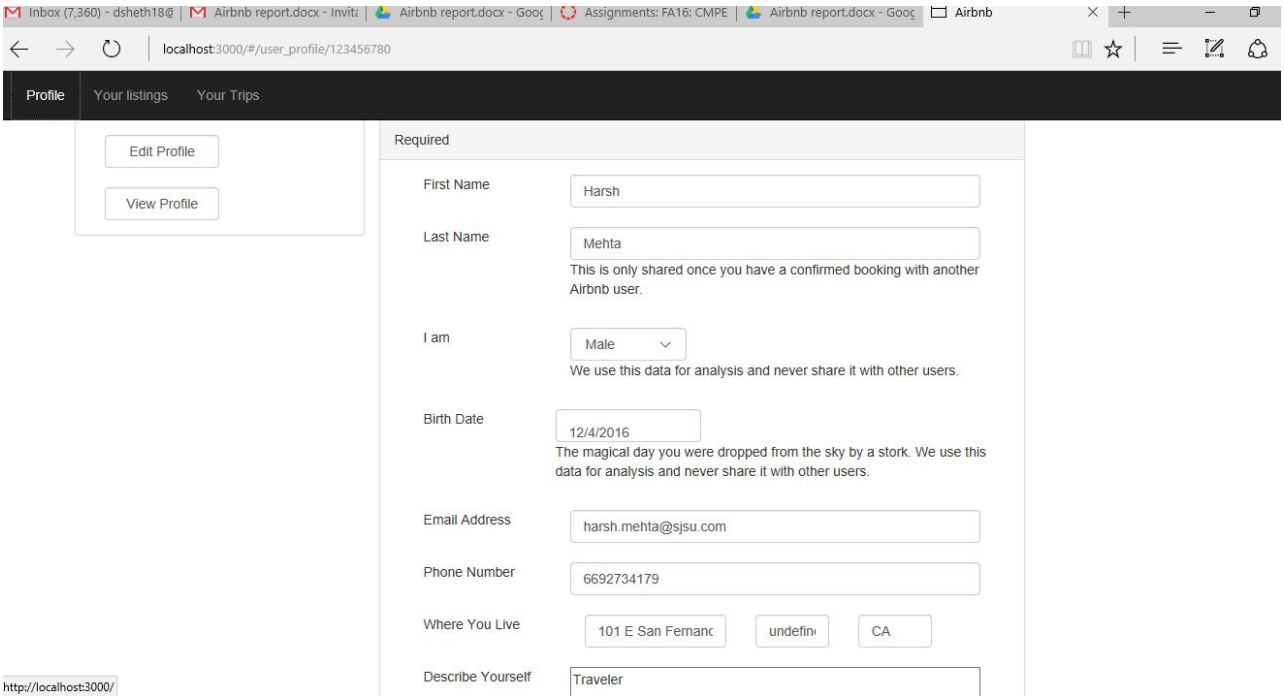
`



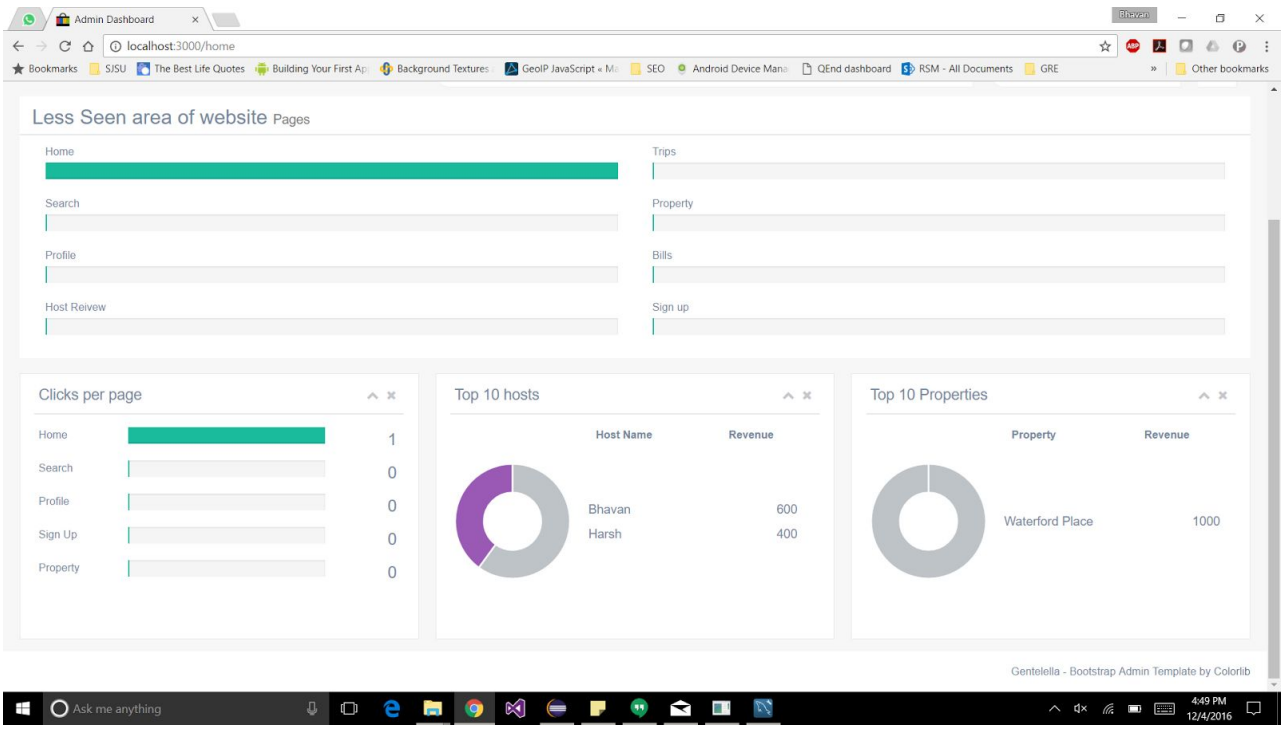
14.Log in validation:

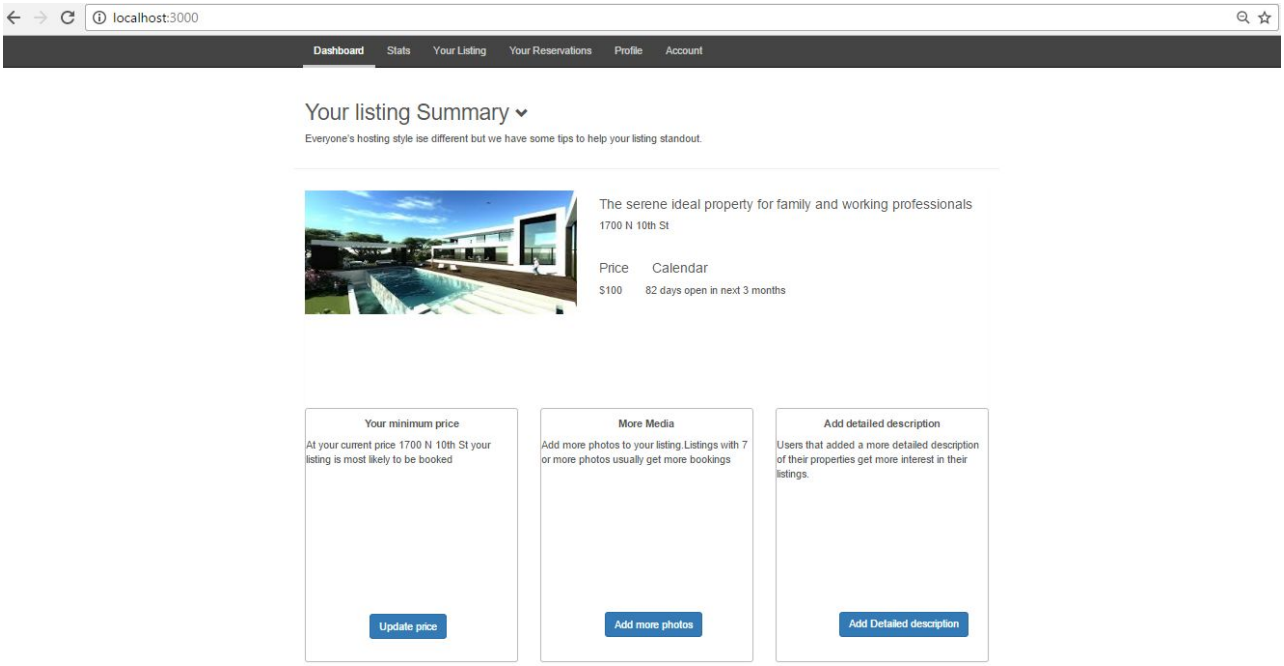


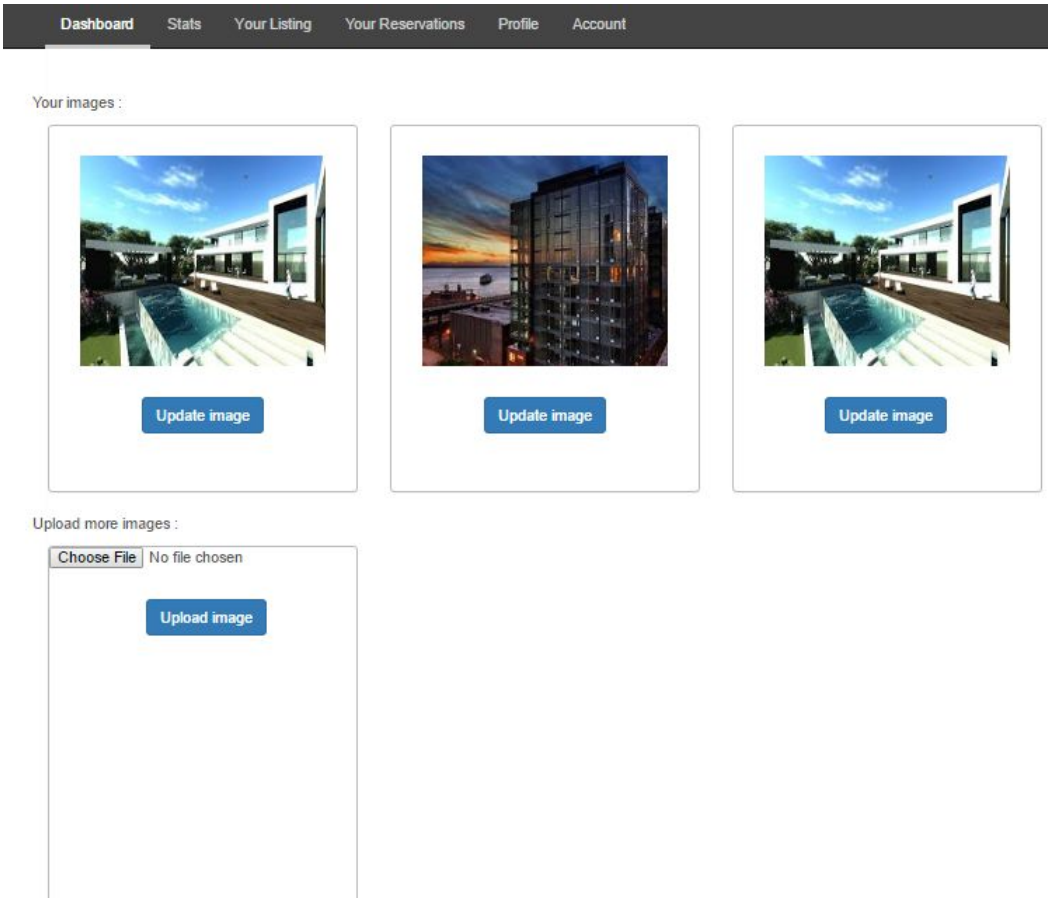
15.User Dashboard:The user dashboard allows the user to update profile, view listings, reservations.



16. Host Dashboard: The host dashboard allows the user to maintain stats, view listings, reservations, profile and account.




localhost:3000
Dashboard
Stats
Your Listing
Your Reservations
Profile
Account
Your listing Summary
Everyone's hosting style ise different but we have some tips to help your listing standout.
The serene ideal property for family and working professionals
1700 N 10th St
Price
Calendar
$100
82 days open in next 3 months
Your minimum price
At your current price 1700 N 10th St your listing is most likely to be booked
Update price
More Media
Add more photos to your listing.Listings with 7 or more photos usually get more bookings
Add more photos
Add detailed description
Users that added a more detailed description of their properties get more interest in their listings.
Add Detailed description


Dashboard
Stats
Your Listing
Your Reservations
Profile
Account
Your images :
Update image
Update image
Update image
Upload more images :
Choose File
No file chosen
Upload image

Dashboard | Stats | Your Listing | Your Reservations | Profile | Account

Your videos :

0:00 / 1:13

Update video

0:00 / 1:13

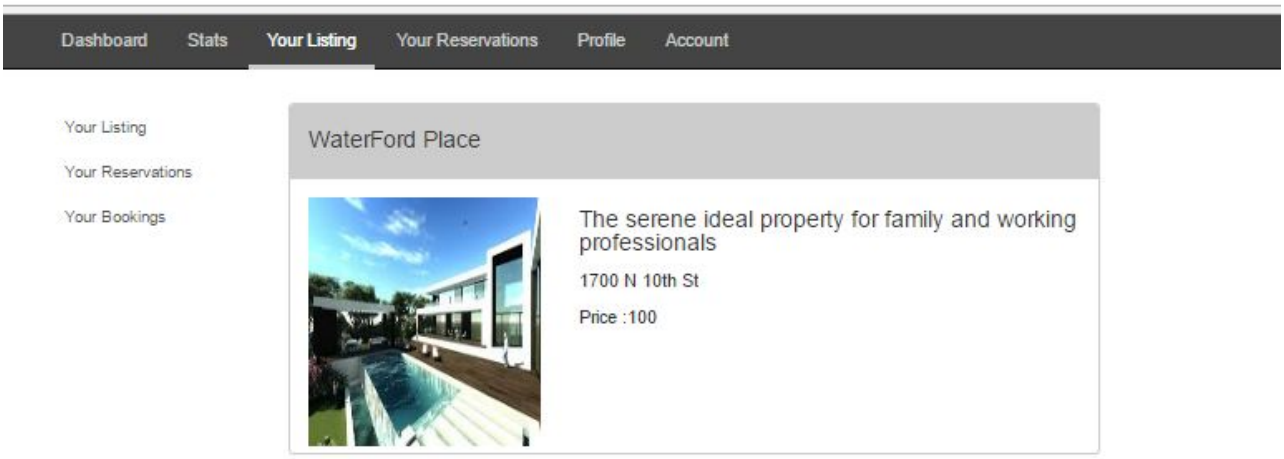
Update video

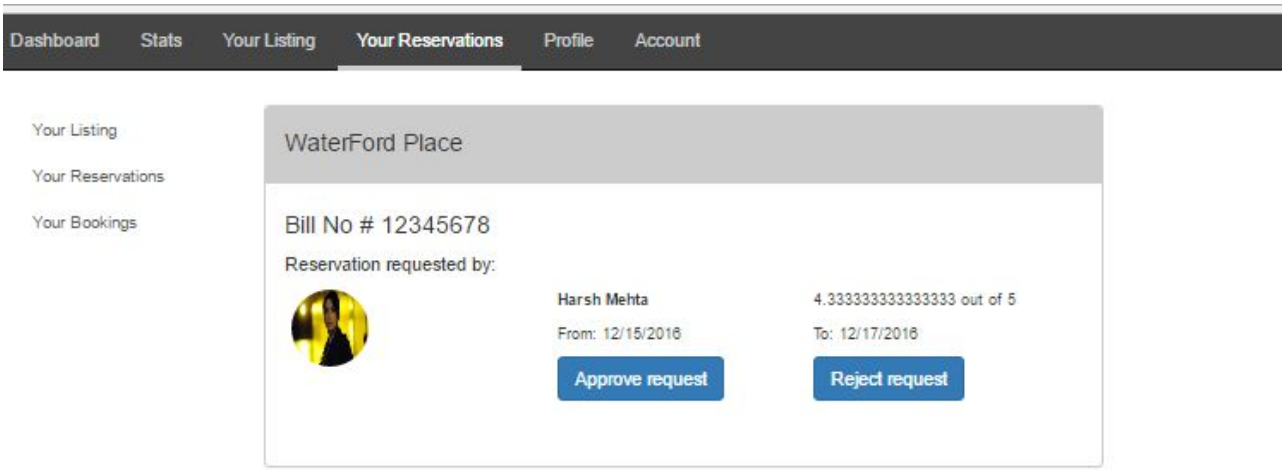
Upload more videos :

Choose File No file chosen

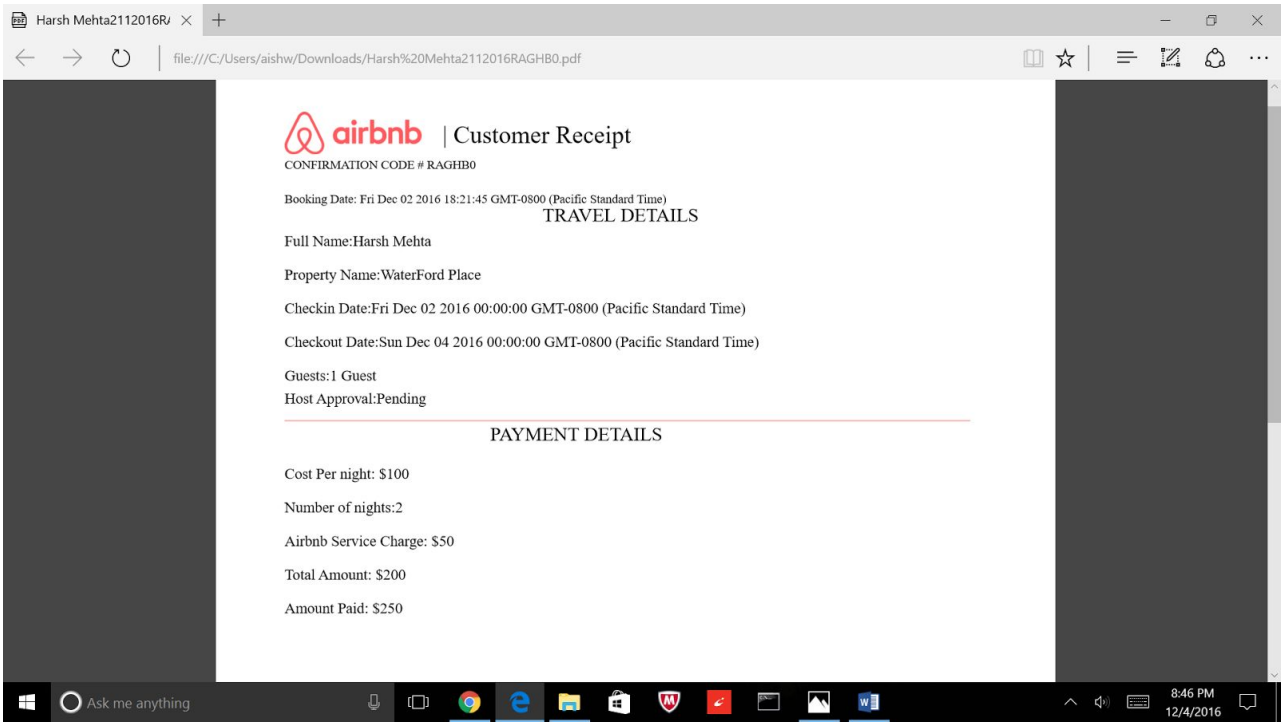
0:00

Upload video

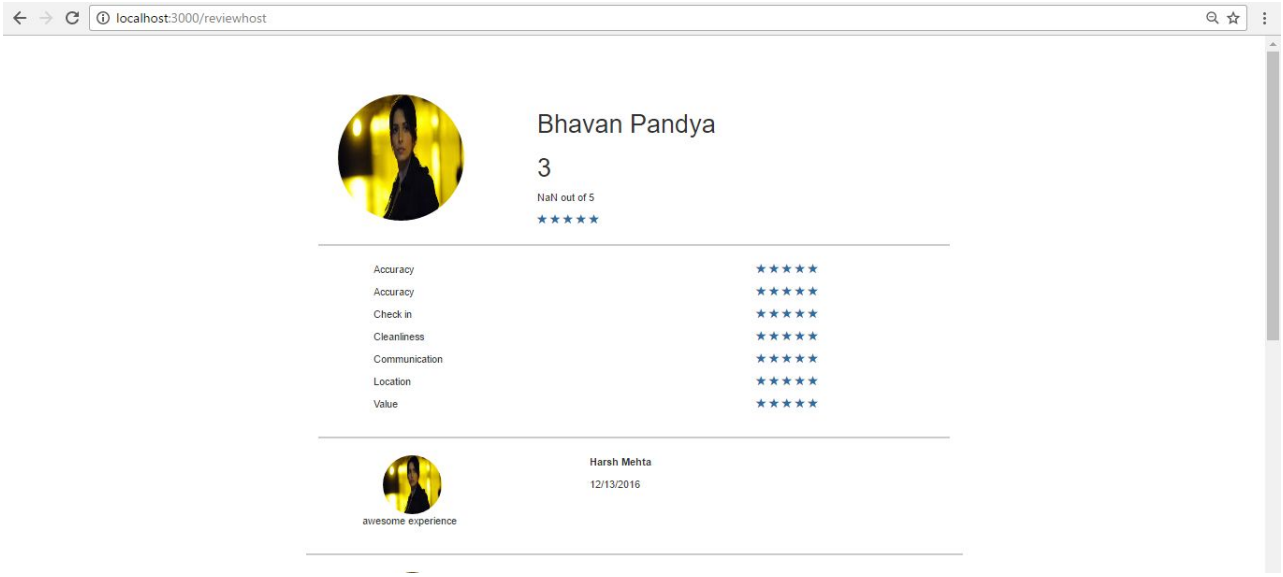




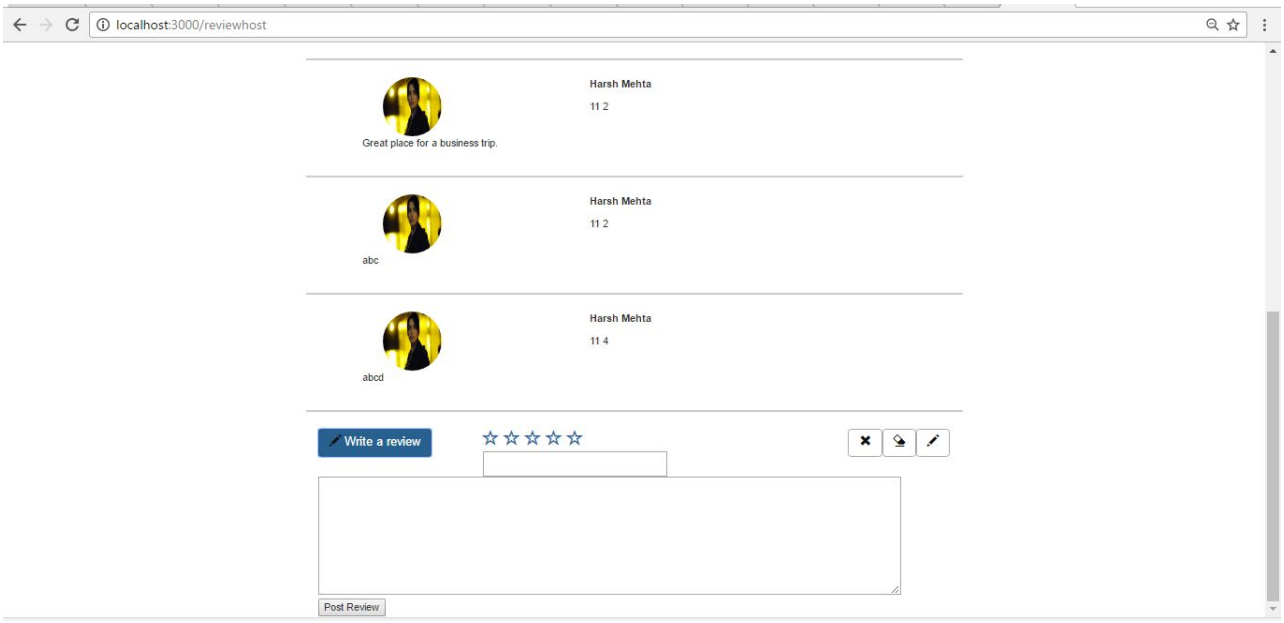
16. Bill Details:



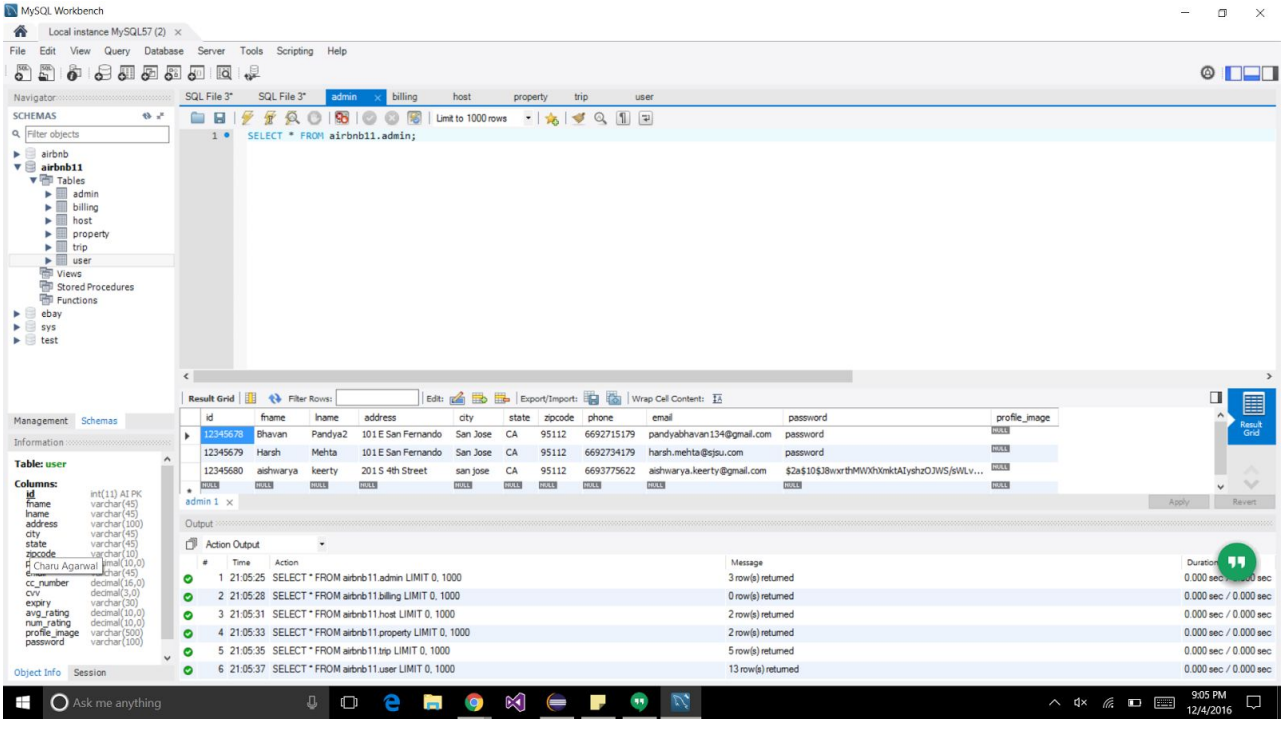
17. Host/User reviews:



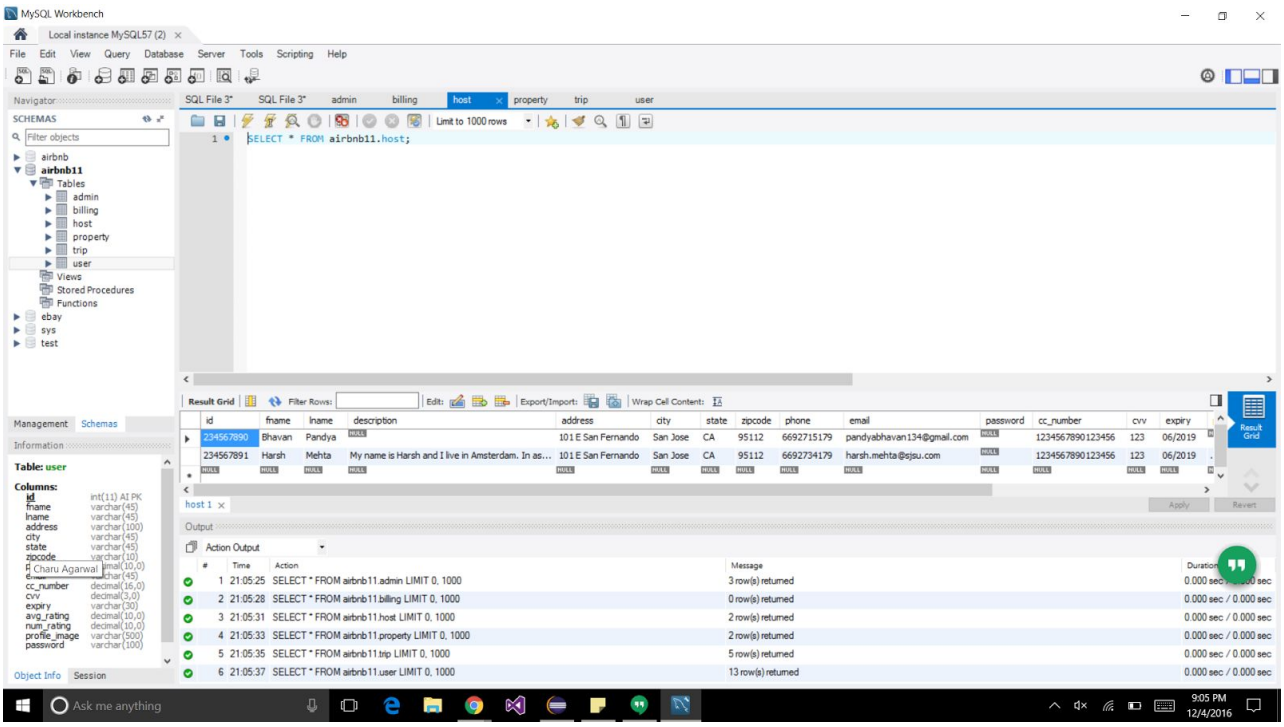
18. Write a review :



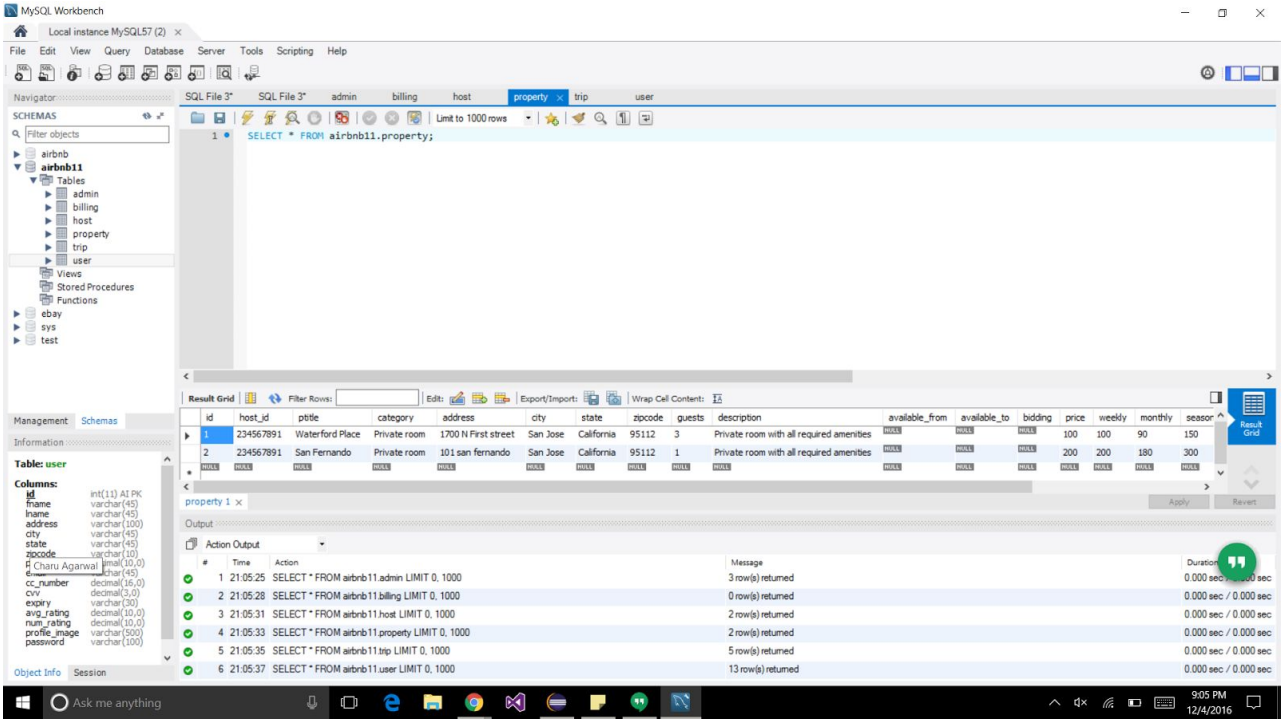
17. MySQL admin table:



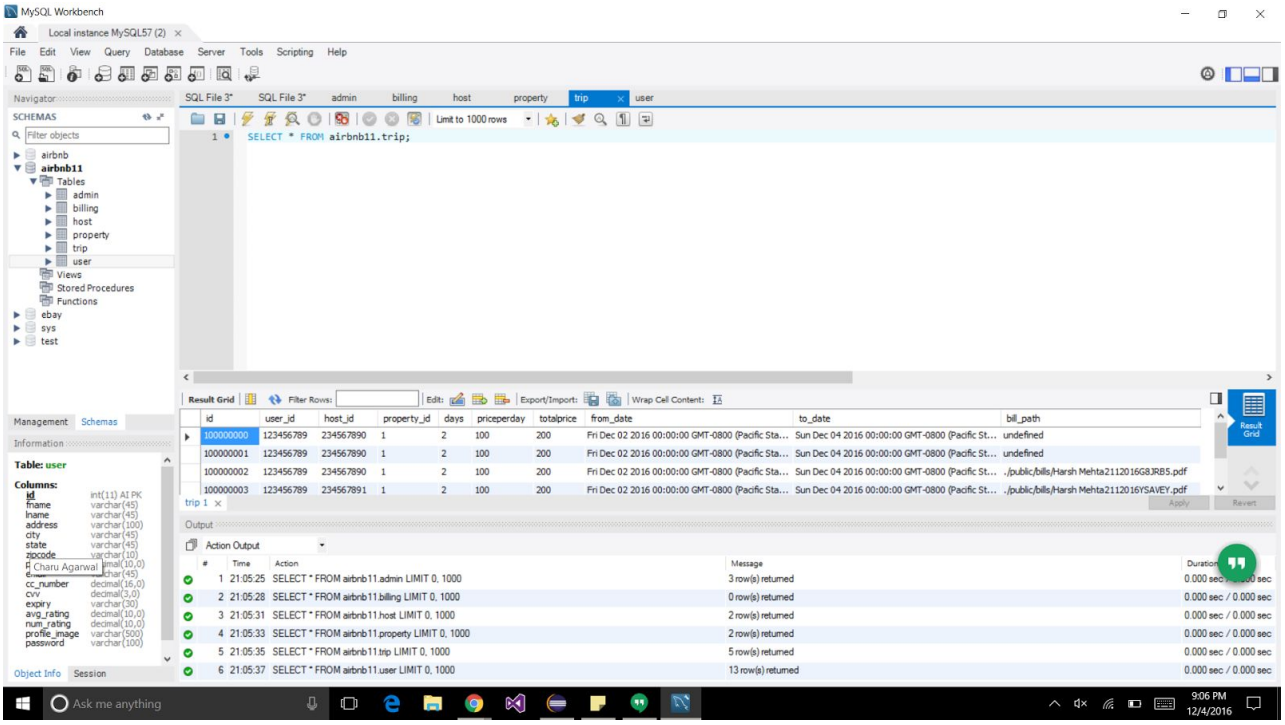
18. MySQL host database:



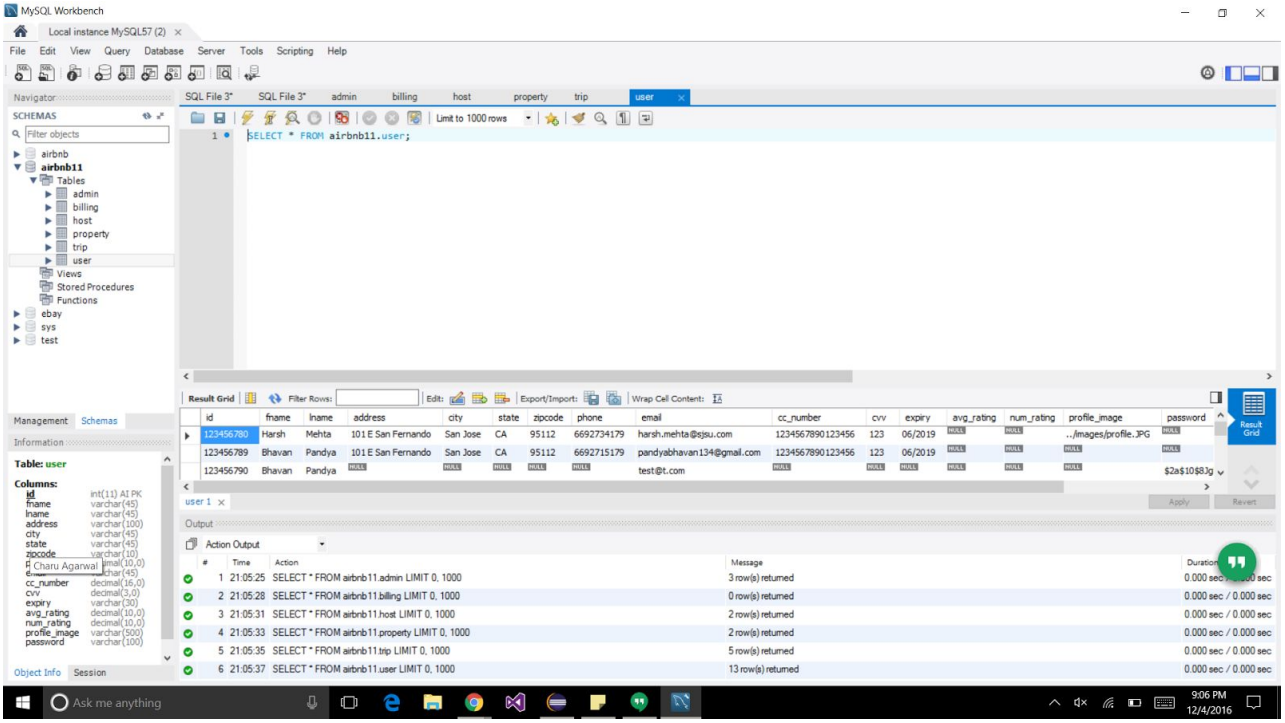
19. MySQL property Database:



20.MySQL trip table:



21.MySQL user table:



22. Add property:

AirBNB

Hi, Aishwarya! Lets get you ready to become a host.

What kind of place do you have?

Entire Place | for 1 guest

Address:

201 S 4th Street

City:

San jose

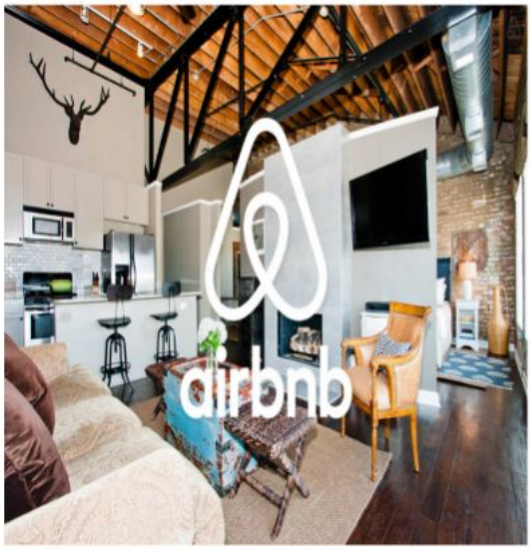
State:

CA

Zip code:

95112

Continue



Ask me anything

9:59 PM
12/4/2016

AirBNB

Give your place a title

The Colonnade Apartments

Examples from similar listings near you

The Cyprus' Nimpkish Forest B and B

Image here

How travelers use titles

Travelers skim over the title when searching for places on Airbnb. Make titles catchy and short.

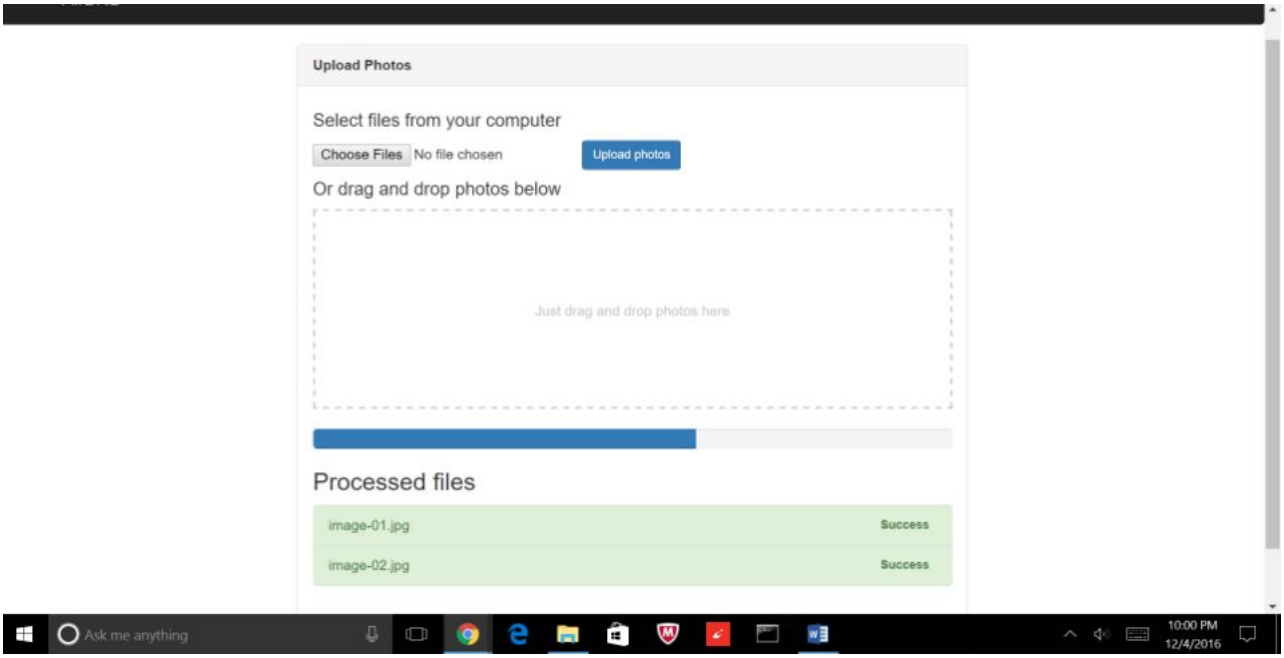
The best titles mention what's unique about your place—where it's close to, if it's big or cosy, if you have a special amenity—and what style it is—like "bright", "hipster", "family-style".

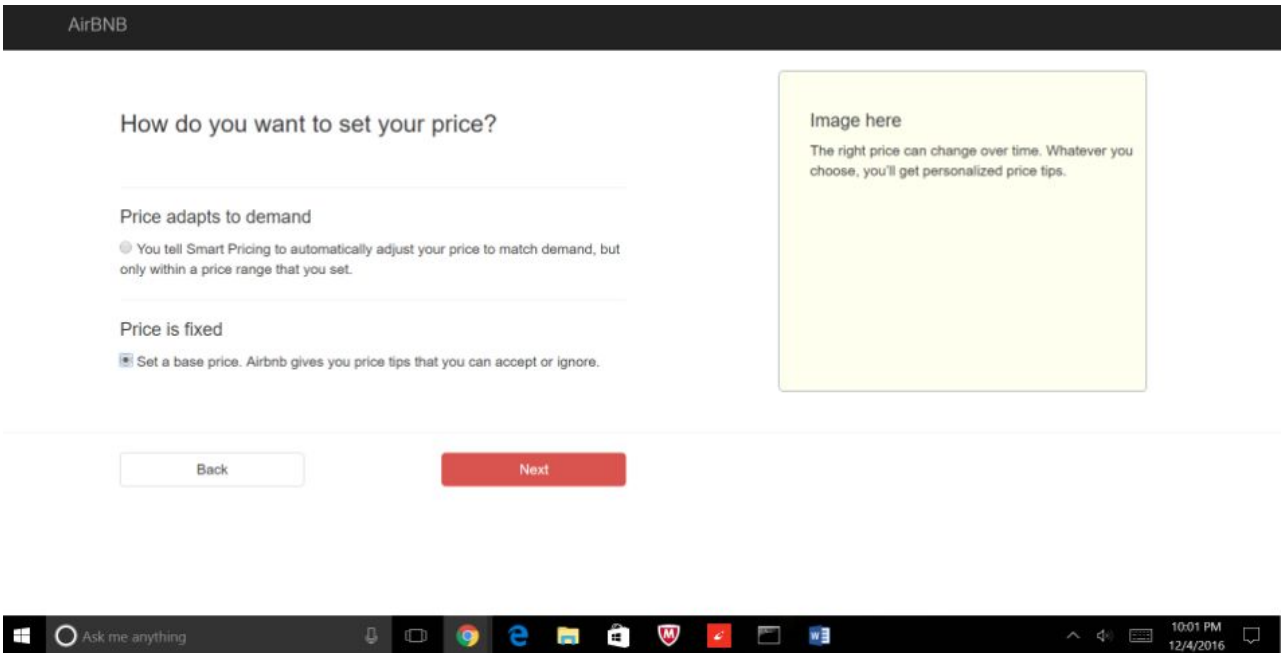
Back

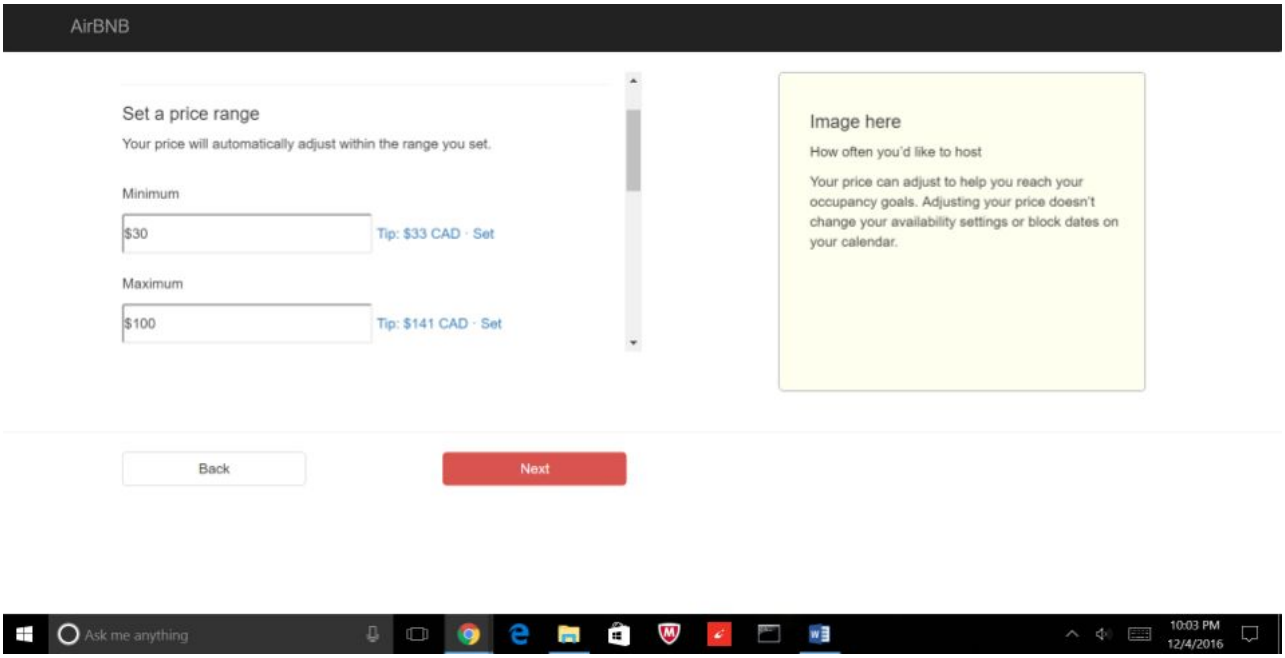
Next

Ask me anything

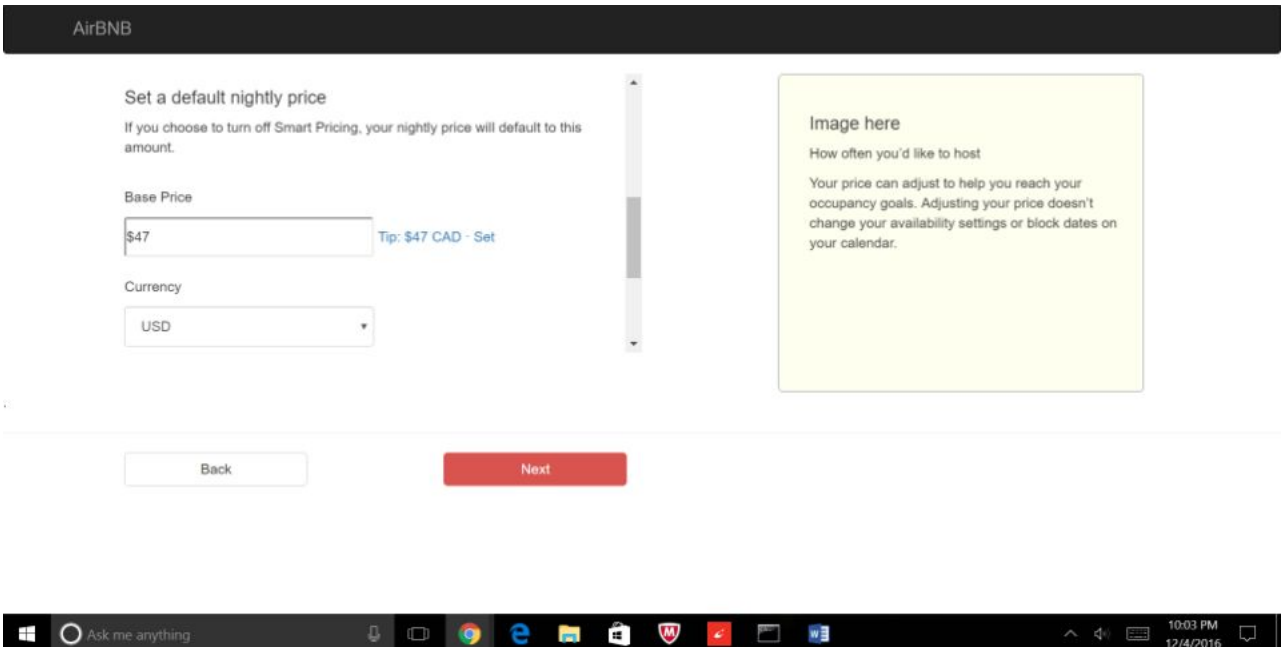
9:59 PM
12/4/2016


Upload Photos
Select files from your computer
Choose Files
No file chosen
Upload photos
Or drag and drop photos below
Just drag and drop photos here
Processed files
image-01.jpg
Success
image-02.jpg
Success
Ask me anything
10:00 PM
12/4/2016

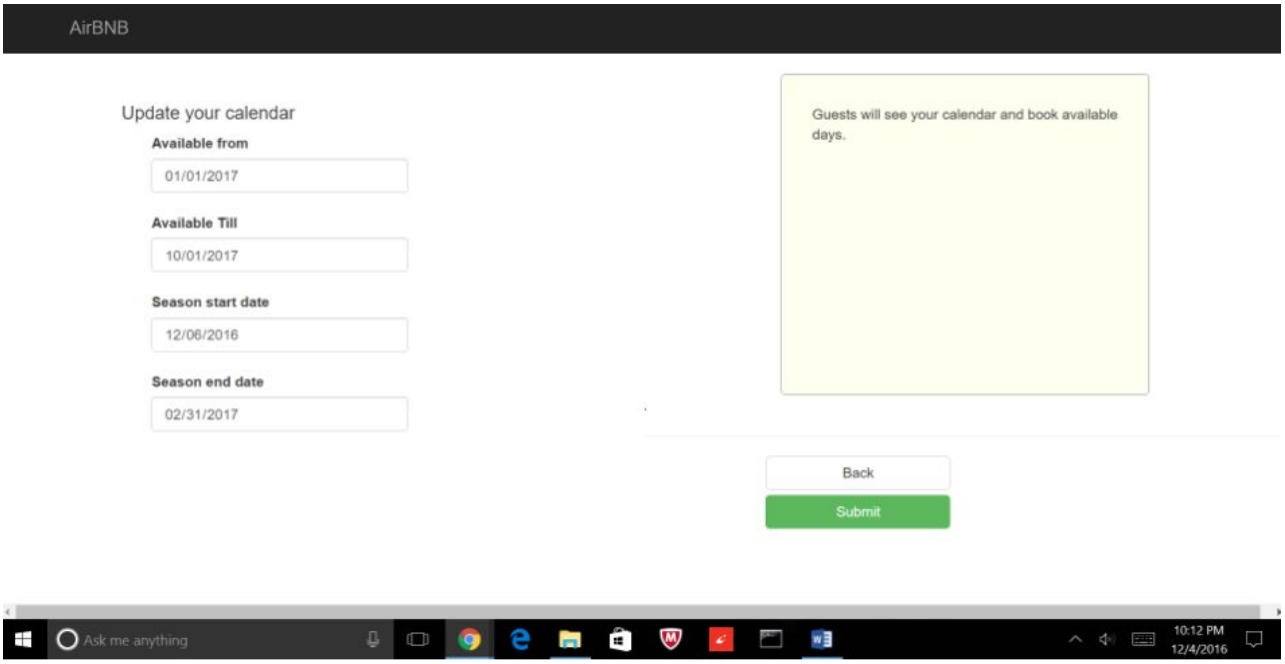

AirBNB
How do you want to set your price?
Price adapts to demand
You tell Smart Pricing to automatically adjust your price to match demand, but only within a price range that you set.
Price is fixed
Set a base price. Airbnb gives you price tips that you can accept or ignore.
Image here
The right price can change over time. Whatever you choose, you'll get personalized price tips.
Back
Next
Ask me anything
10:01 PM
12/4/2016


AirBNB
Set a price range
Your price will automatically adjust within the range you set.
Minimum
$30
Tip: $33 CAD · Set
Maximum
$100
Tip: $141 CAD · Set
Image here
How often you'd like to host
Your price can adjust to help you reach your occupancy goals. Adjusting your price doesn't change your availability settings or block dates on your calendar.
Back
Next

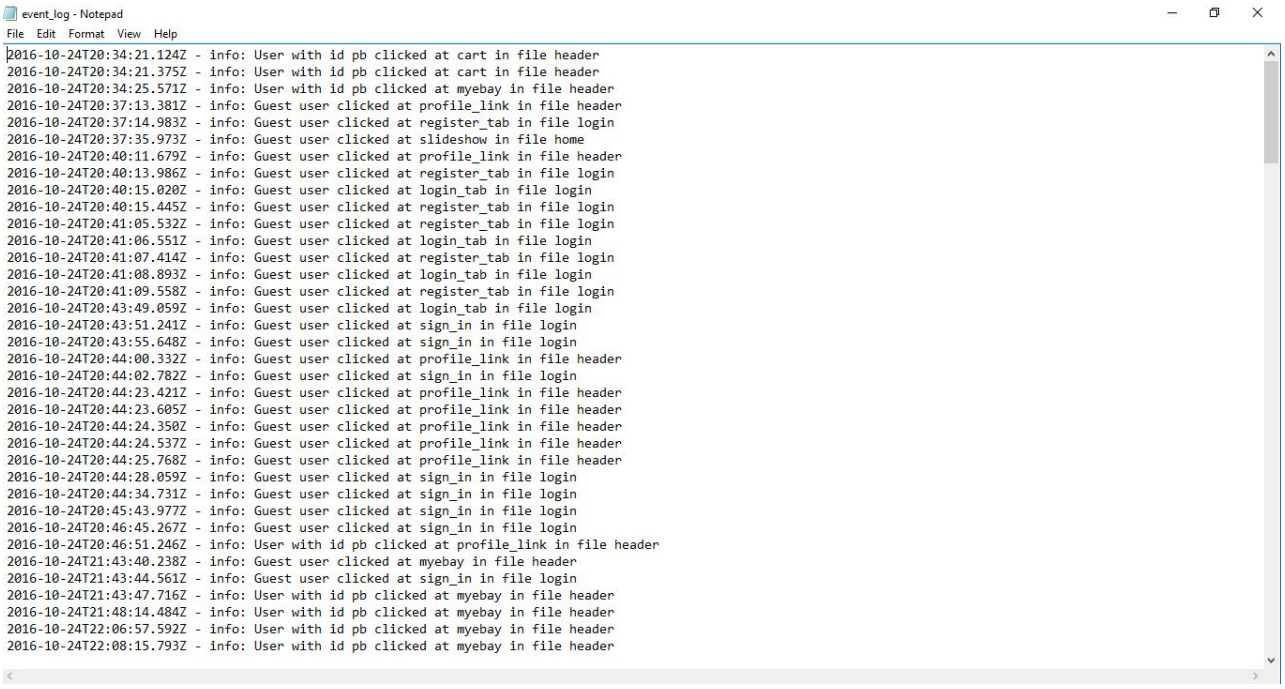
Ask me anything
10:03 PM
12/4/2016


AirBNB
Set a default nightly price
If you choose to turn off Smart Pricing, your nightly price will default to this amount.
Base Price
$47
Tip: $47 CAD · Set
Currency
USD
Image here
How often you'd like to host
Your price can adjust to help you reach your occupancy goals. Adjusting your price doesn't change your availability settings or block dates on your calendar.
Back
Next

Ask me anything
10:03 PM
12/4/2016



22. Log file:



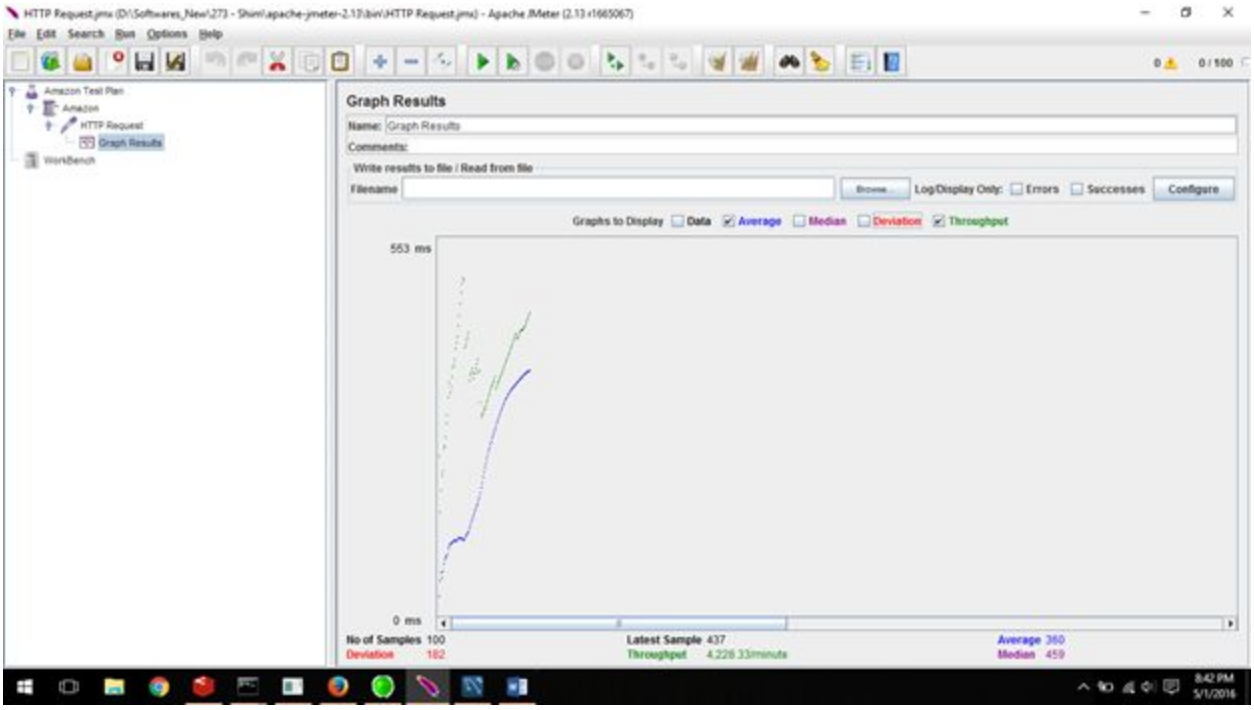
23. Testing: mocha.js

```
var request = require('request')
, express = require('express')
,assert = require("assert")
,http = require("http");
```

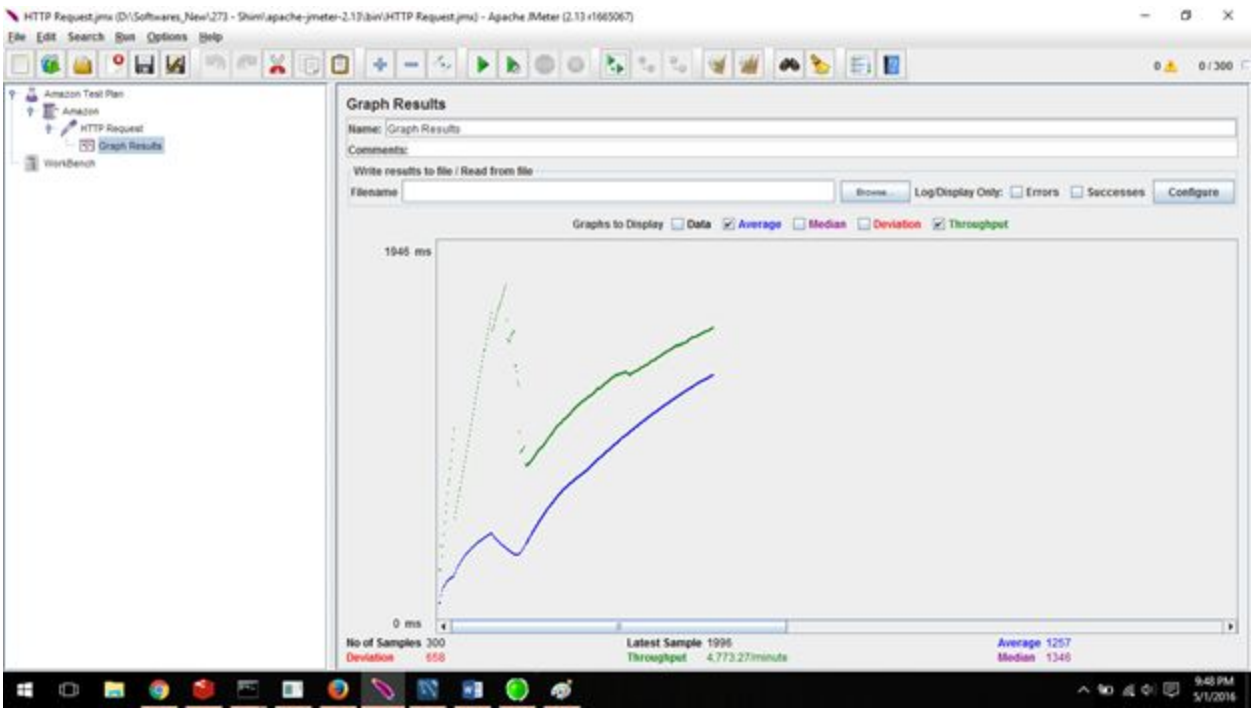
```
describe('http tests', function(){
it('Test case for Login', function(done) {
    request.post( 'http://localhost:3000/login,{ form: {
'username':'pandyabhavan134@gmail.com','password':'password' } },
 function (error, response, body) {
 assert.equal(200, response.statusCode);
 done();
  }
   );
   });
   it('Test case Get Header values for admin dashboard', function(done) {
   request.get('http://localhost:3000/getHeaderValues',function (error, response, body) {
assert.equal(200, response.statusCode);
done();
}
);
});
```

JMETER TESTING:

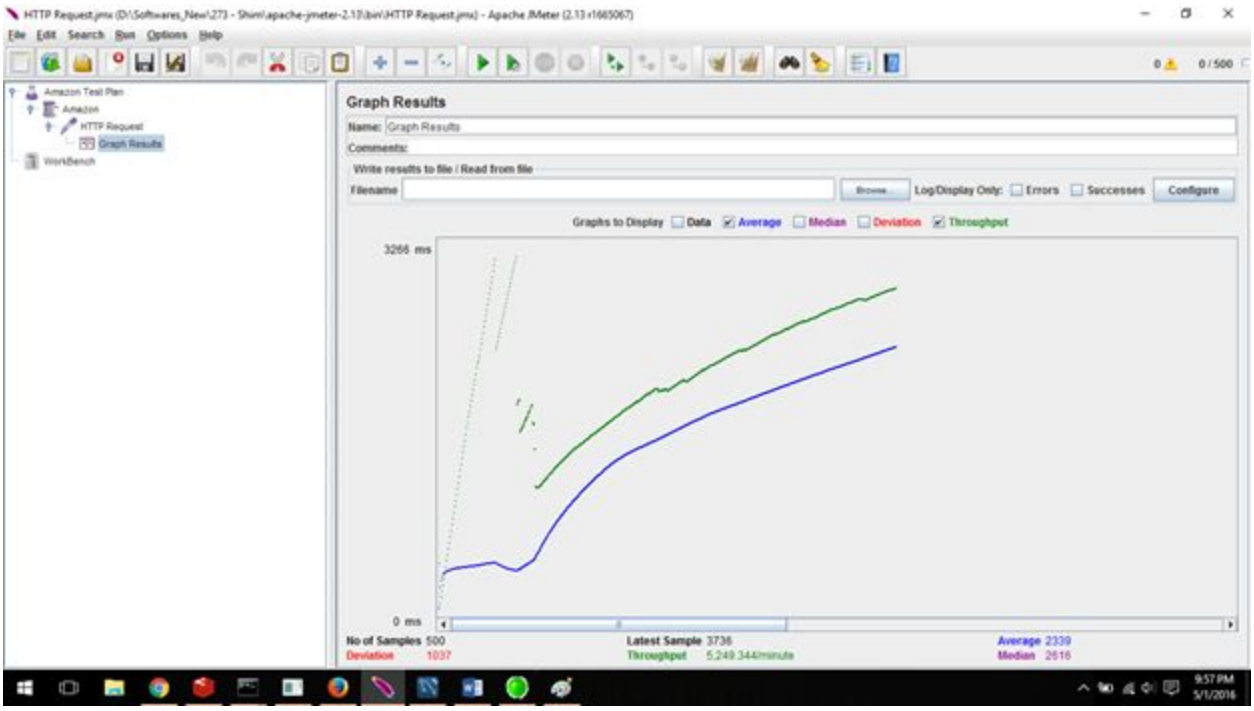
- 100 concurrent users:



- 300 concurrent users:



- 500 concurrent users:



CODE LISTING OF CLIENT APPLICATION:

- ejs file:

```
<!DOCTYPE html>
```

```
<html>
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1">
<title>Airbnb</title>
<link rel='stylesheet' href='/stylesheets/style.css' />
<link rel="stylesheet" href="../stylesheets/bootstrap/css/bootstrap.css">
<link rel="stylesheet" href="https://unpkg.com/purecss@0.6.0/build/pure-min.css">
<link rel="stylesheet"
href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.6.3/css/font-awesome.min.css">
    <script type="text/javascript"
src="http://maps.googleapis.com/maps/api/js?libraries=places&sensor=false"></script>
   <script src="../javascripts/angular.js"></script>
   <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.1.1/jquery.min.js"></script>
   <script src="../stylesheets/bootstrap/js/bootstrap.js"></script>
   <script src="../angularjs/clientScript.js"></script>
   <script src="../angularjs/ngAutocomplete.js"></script>

   <!--//UI Router script-->
   <script
src="https://cdnjs.cloudflare.com/ajax/libs/angular-ui-router/0.3.2/angular-ui-router.js"></script>
  </head>
  <body ng-app="clientModule">
   <div ui-view="index"></div>
  </body>
</html>
```

- html form snippet:

```
<div class="form-group">
   <div class="cols-sm-10">
      <diiv class="input-group">
         <input type="email" class="form-control" name="email" id="regemail" placeholder="Email
address" ng-model="user.email" required />
 <span class="input-group-addon"><i class="fa fa-envelope fa" aria-hidden="true"></i></span>
</div>
 </div>
</div>
<div class="form-group">
<div class="cols-sm-10">
<div class="input-group">
<input type="password" class="form-control" name="password" id="password" placeholder="New
Password" ng-model="user.password" required />
 <span class="input-group-addon"><i class="fa fa-lock fa-lg" aria-hidden="true"></i></span>
</div>
</div>
</div>
```

CONFIGURATION FILES:

- app.js

```
var express = require('express')
  , routes = require('./routes')
  , user = require('./routes/user')
  , http = require('http')
  , path = require('path')
    , host = require('./routes/host')
    , property = require('./routes/property');

var app = express();
var passport = require('passport');
require('./routes/passport')(passport);

var mysql=require('mysql');
var bcrypt = require('bcryptjs');

var expressSession = require("express-session");
var mongoStore = require("connect-mongo/es5")(expressSession);

var mongoURL="mongodb://localhost:27017/airbnbv10";
var login=require('./routes/login');
var room=require('./routes/room');
var profile=require('./routes/profile');
//Session configuration
app.use(expressSession({  secret: 'Airbnb',
  resave: false,
  //don't save session if unmodified   // don't create session until something stored
  saveUninitialized: false,
  duration: 30 * 60 * 1000,
  activeDuration: 5 * 60 * 1000,
  store: new mongoStore({   url: mongoURL  })
}));
//uncomment below middleware once passport startegy is defined
app.use(passport.initialize());
app.set('port', process.env.PORT || 3000);
app.set('views', __dirname + '/views');
app.set('view engine', 'ejs');
app.use(express.favicon());
app.use(express.logger('dev'));
app.use(express.bodyParser());
app.use(express.methodOverride());
app.use(app.router);
app.use(express.static(path.join(__dirname, 'public')));

// development only
```

```
if ('development' == app.get('env')) {
  app.use(express.errorHandler());
}

app.post('/loginUser', function(req, res, next) {
    passport.authenticate('login', function(err, user, info) {
        if(err) {
            response = {"statusCode":403,"data":null};
            res.send(JSON.stringify(response));
        }

        if(!user) {
            response = {"statusCode":401,"data":null};
            res.send(JSON.stringify(response));
        }

        req.logIn(user, {session:false}, function(err) {
            if(err) {
                response = {"statusCode":401,"data":null};
                res.send(JSON.stringify(response));
            }

            req.session.login = user;
            console.log("inside passport"+req.session.login);
            response = {"statusCode":200,"data":user};
            res.send(JSON.stringify(response));
        })
    })(req, res, next);
});
app.post('/getLoginSession',login.getLoginSession);

app.get('/loginUser', isAuthenticated, function(req, res) {
    res.render('successLogin', {user:{username: req.session.user}});
});

function isAuthenticated(req, res, next) {
    console.log('inside isauth');
    if(req.session.user) {
        console.log('in inside isauth');
        console.log(req.session.user);
        return next();
    }

    res.redirect('/');
};
```

```
// development only
app.post('/registerUser',login.register);


app.get('/',routes.index);
app.get('/users', user.list);
//calls to routes for airbnb

//app.post('/register',login.register);

//api to get all the room details
app.post('/getRoom',room.getDetails);
app.post('/getRoomd',room.getRoomDetails);

app.post('/gethostprofile',profile.getHostProfile);
app.post('/checkout',room.checkout);

app.post('/bookproperty',room.bookRoom);

app.post('/logout', function(req,res) {
   req.session.destroy();
   res.redirect('/');
});

var fs = require('fs');
app.post('/getPropertyListing',property.getPropertyListing);
app.post('/getBilling',property.getBilling);
app.post('/uploadimage', uploadimage);
app.post('/uploadvideo', uploadvideo);
//app.get('/rate_host', host.rate_host);
app.post('/review_host', host.review_host);
app.get('/reviewhost', host.reviewhost);
app.post('/getcurrentrating', host.getcurrentrating);
function uploadimage(req,res) {
   sleep(5000);
   var inStr =  fs.createReadStream("/Users/mbars/Downloads/"+req.param("image"));
   var outStr =
fs.createWriteStream("/airbnb/RabbitMQApplication/RabbitMQApplication/RabbitMQ-Client/public/i
mages/"+req.param("image"));

   inStr.pipe(outStr);
   property.addImage(req,res);

}
function uploadvideo(req,res) {
   sleep(5000);
   var inStr =  fs.createReadStream("/Users/mbars/Downloads/"+req.param("image"));
   var outStr =
```

```
fs.createWriteStream("/airbnb/RabbitMQApplication/RabbitMQApplication/RabbitMQ-Client/public/images/"+req.param("image"));

    inStr.pipe(outStr);
    property.addVideo(req,res);

}

http.createServer(app).listen(app.get('port'), function(){
  console.log('Express server listening on port ' + app.get('port'));
});
```

- Server.js

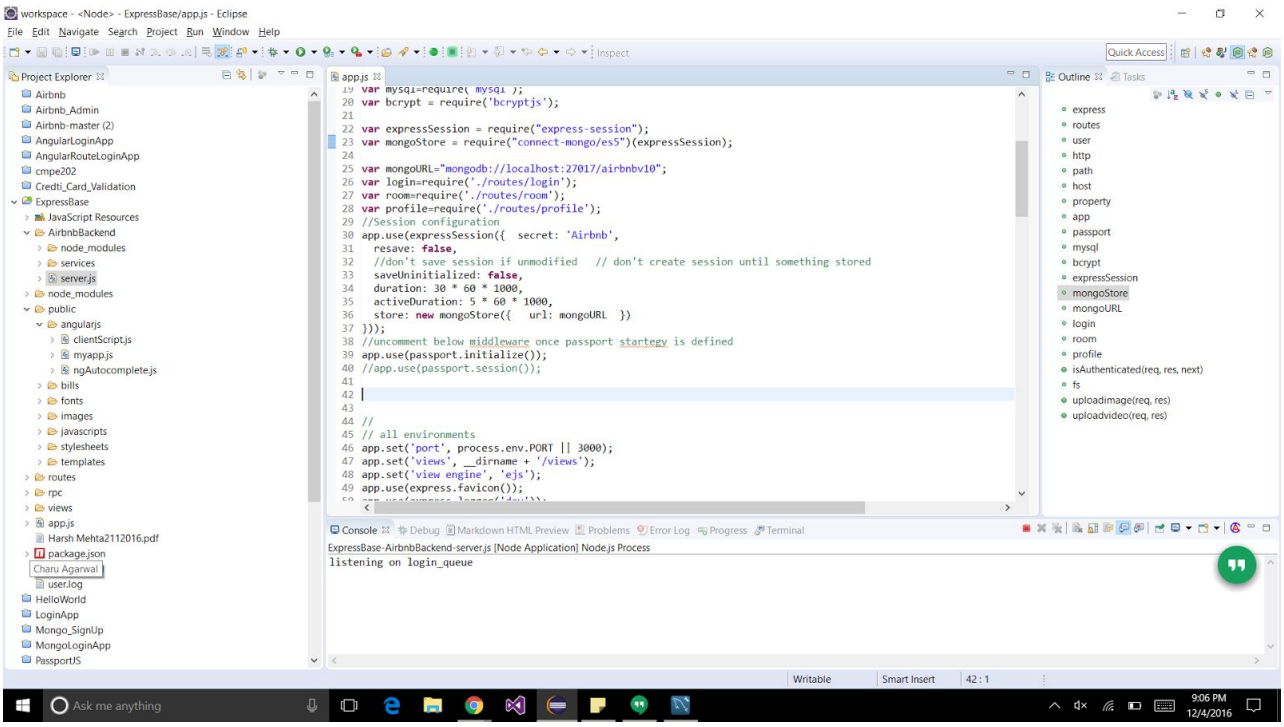
```
//super simple rpc server example
var amqp = require('amqp')
    , util = require('util');

var login = require('./services/login');

var property = require('./services/property');

var host = require('./services/host');
var cnn = amqp.createConnection({host:'127.0.0.1'});

cnn.on('ready', function(){
    console.log("listening on login_queue");

    cnn.queue('login_queue', function(q){

        q.subscribe(function(message, headers, deliveryInfo, m){
            util.log(util.format( deliveryInfo.routingKey, message));
            util.log("Message: "+JSON.stringify(message));
            util.log("DeliveryInfo: "+JSON.stringify(deliveryInfo));
            login.handle_request(message, function(err,res){

                //return index sent
                cnn.publish(m.replyTo, res, {
                    contentType:'application/json',
                    contentEncoding:'utf-8',
                    correlationId:m.correlationId
                });
            });
        });
    });


    cnn.queue('propertyListing_queue', function(q){
        q.subscribe(function(message, headers, deliveryInfo, m){
```

```
        util.log(util.format( deliveryInfo.routingKey, message));
        util.log("Message: "+JSON.stringify(message));
        util.log("DeliveryInfo: "+JSON.stringify(deliveryInfo));
        property.handle_request(message, function(err,res){

            //return index sent
            cnn.publish(m.replyTo, res, {
                contentType:'application/json',
                contentEncoding:'utf-8',
                correlationId:m.correlationId
            });
        });
    });
});
cnn.queue('billing_queue', function(q){
    q.subscribe(function(message, headers, deliveryInfo, m){
        util.log(util.format( deliveryInfo.routingKey, message));
        util.log("Message: "+JSON.stringify(message));
        util.log("DeliveryInfo: "+JSON.stringify(deliveryInfo));
        property.getBilling(message, function(err,res){

            //return index sent
            cnn.publish(m.replyTo, res, {
                contentType:'application/json',
                contentEncoding:'utf-8',
                correlationId:m.correlationId
            });
        });
    });
});
cnn.queue('approvebill_queue', function(q){
    q.subscribe(function(message, headers, deliveryInfo, m){
        util.log(util.format( deliveryInfo.routingKey, message));
        util.log("Message: "+JSON.stringify(message));
        util.log("DeliveryInfo: "+JSON.stringify(deliveryInfo));
        property.approveBill(message, function(err,res){

            //return index sent
            cnn.publish(m.replyTo, res, {
                contentType:'application/json',
                contentEncoding:'utf-8',
                correlationId:m.correlationId
            });
        });
    });
});
cnn.queue('addImage_queue', function(q){
    q.subscribe(function(message, headers, deliveryInfo, m){
```

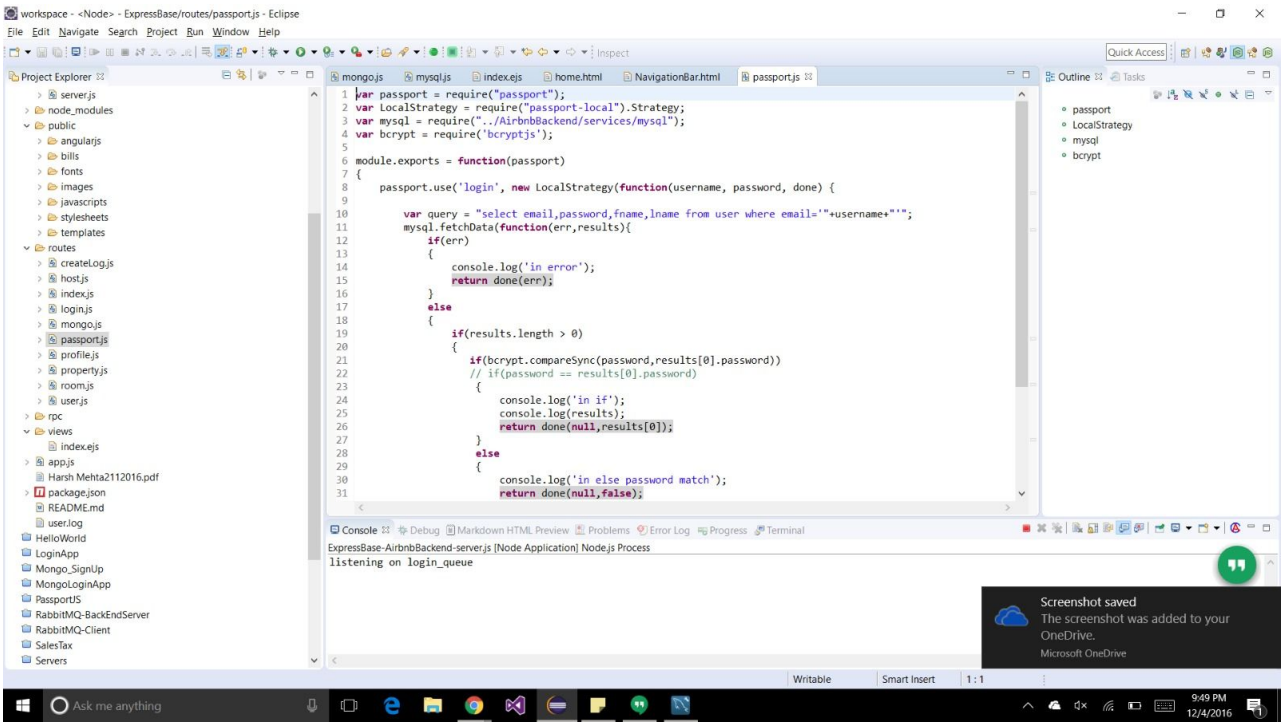
```
        util.log(util.format( deliveryInfo.routingKey, message));
        util.log("Message: "+JSON.stringify(message));
        util.log("DeliveryInfo: "+JSON.stringify(deliveryInfo));
        property.addImage(message, function(err,res){

            //return index sent
            cnn.publish(m.replyTo, res, {
                contentType:'application/json',
                contentEncoding:'utf-8',
                correlationId:m.correlationId
            });
        });
    });
});

cnn.queue('addVideo_queue', function(q){
    q.subscribe(function(message, headers, deliveryInfo, m){
        util.log(util.format( deliveryInfo.routingKey, message));
        util.log("Message: "+JSON.stringify(message));
        util.log("DeliveryInfo: "+JSON.stringify(deliveryInfo));
        property.addVideo(message, function(err,res){

            //return index sent
            cnn.publish(m.replyTo, res, {
                contentType:'application/json',
                contentEncoding:'utf-8',
                correlationId:m.correlationId
            });
        });
    });
});

cnn.queue('gethostreviews_queue', function(q){
    q.subscribe(function(message, headers, deliveryInfo, m){
        util.log(util.format( deliveryInfo.routingKey, message));
        util.log("Message: "+JSON.stringify(message));
        util.log("DeliveryInfo: "+JSON.stringify(deliveryInfo));
        host.getcurrentreviews(message, function(err,res){

            //return index sent
            cnn.publish(m.replyTo, res, {
                contentType:'application/json',
                contentEncoding:'utf-8',
                correlationId:m.correlationId
            });
        });
    });
});
```

```
    cnn.queue('review_host_queue', function(q){
        q.subscribe(function(message, headers, deliveryInfo, m){
            util.log(util.format( deliveryInfo.routingKey, message));
            util.log("Message: "+JSON.stringify(message));
            util.log("DeliveryInfo: "+JSON.stringify(deliveryInfo));
            host.review_host(message, function(err,res){

                //return index sent
                cnn.publish(m.replyTo, res, {
                    contentType:'application/json',
                    contentEncoding:'utf-8',
                    correlationId:m.correlationId
                });
            });
        });
    });


});
```

EXPRESS SESSION:



PASSPORT JS:



Optimisation Techniques for Web Application

1) **Redis Caching**:

Redis caching is implemented on the server side to reduce the overhead of opening and closing DB connections of the frequently visited property.

app.js

```
app.post('/getRoomd', function (req, res) {
  if (!req.param('data'))
  {res.status(400).send("Please send a proper title");
    console.log("in if of app redis details"+req);
  }
  else {
    room.getRoomDetailsCached(redis, req.param('data'), function (room) {
      if (!room) res.status(500).send("Server error");
      else res.status(200).send(room);
    });
  }
});
```

room.js

```
var getRoomDetails=function (req,callback)          //changes in exports after redis
{
   //console.log("In room details"+req.title);
   console.log("in room details"+req);
   var msg_payload={
      proname: req, /////changes in exports after redis
      queue:"room_details"
   }

   mq_client.make_request('login_queue',msg_payload, function(err,results){

      console.log(results);
      if(err){
         throw err;
      }
      else
      {
         if(results.statusCode === 200){
            console.log("valid Login");
            //req.session.fname=results.fname;
            res={"room":results.room, "statusCode":200, "image":results.image};
            //res.send({"room":results.room, "statusCode":200, "image":results.image});
            callback(null,res);

            console.log("success");
         }
         else {
            console.log("Invalid Login");
            //res.send({"login":"Fail"});
            res={"login":"Fail"};
            callback(null,res);
         }
      }
   });
};
exports.getRoomDetailsCached=function (redis, title, callback) {
   console.log("In redis cachd"+title);
   redis.get(title, function (err, reply) {
      if (err) callback(null);
      else if (reply) //room exists in cache
         callback(JSON.parse(reply));
      else {
         console.log("In redis cachd"+title);
         getRoomDetails(title,function (err,room) {
            if (err || !room)
            {
               callback(null);
            }
```

```
            else {
                    redis.set(title, JSON.stringify(room), function () {
                      console.log("in set redis"+redis.get(title));
                      callback(room);
                    });
                }
        });
    }
  });};
```

2) Connection Pooling:

Connection pooling is implemented for both the databases, MySQL and MongoDB.

MongoDB Pooling:

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

/**
 * Connects to the MongoDB Database with the provided URL
 */
exports.connect = function(url, callback){
            var _db= p.get(url);

    db = _db;
    connected = true;
    console.log(connected +" is connected?");
    callback(db);
    p.release(db)

};

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

```
function getConnection(pool){

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

function Pool()
{

  for(var i=0; i < 50; ++i)
  {
    getConnection(pool);
    free.push(i);
  }

}

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

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

var p=new Pool();

MySQL:

- **MySQL connection pool:**

```
var MongoClient = require('mongodb').MongoClient;
var dbConnection;
var connected = false;
var optionvalues = {
        db : {
         numberOfRetries : 5
        },
        server : {
          auto_reconnect : true,
           poolSize : 400,
           socketOptions : {
                connectTimeoutMS : 500
                    }
        },
        replSet : {},
        mongos : {}
};

var dbConnection;
 function initiatePool(url, callback) {
      MongoClient.connect(url, optionvalues, function(err, db) {
        if (err)
                throw err;

                dbConnection = db;
                connected = true;
                callback(dbConnection);
        });
}

exports.connect = function(url, callback) {
         if(!dbConnection){
                        initiatePool(url, callback)
                 }
                 else{
                        callback(dbConnection);
         }
};

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

```
	return dbConnection.collection(name);
};
```

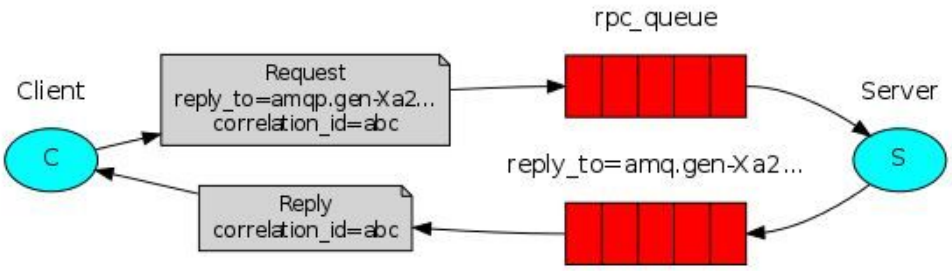
- MySQL Connection Pooling

```
var mysql = require('mysql');
var listOfConnections = require("collections/list")
var connectionPool;

function getConnection(){
	var connection = mysql.createConnection({
	host : 'localhost',
	user : 'root',
	password : 'password',
	database : 'airbnb',
	port : 3306
});
	return connection;
}

exports.createConnectionPool = function(noOfConnections){
	connectionPool = new listOfConnections();
	for(var i=0;i<noOfConnections;i++){
		connectionPool.push(getConnection());
	}
};

var getConnectionFromConnectionPool = function (){
	if(connectionPool.length == 0){
		return getConnection();
	}else{
		return connectionPool.pop();
	}
};

releaseConnection = function(connection){
	connectionPool.push(connection);
};

exports.fetchData = function(callback,sqlQuery){
	var connection=getConnectionFromConnectionPool();
	connection.query(sqlQuery, function(err, rows, fields) {
		if(err){
			console.log("ERROR: " + err.message);
		}
		callback(err, rows);
	});
	releaseConnection(connection);
};
```

3) RabbitMQ:

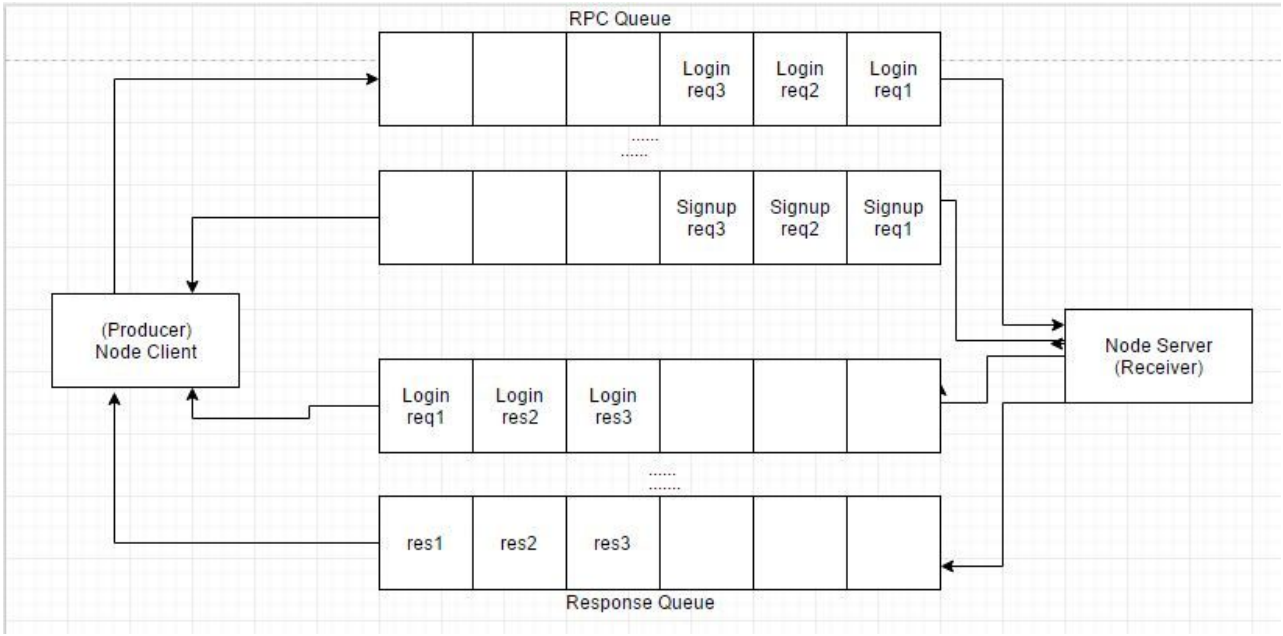
Decoupling of node backend using RPC Message Queues.

Summary



Our RPC will work like this:

- When the Client starts up, it creates an anonymous exclusive callback queue.
- For an RPC request, the Client sends a message with two properties: `reply_to`, which is set to the callback queue and `correlation_id`, which is set to a unique value for every request.
- The request is sent to an `rpc_queue` queue.
- The RPC worker (aka: server) is waiting for requests on that queue. When a request appears, it does the job and sends a message with the result back to the Client, using the queue from the `reply_to` field.
- The client waits for data on the callback queue. When a message appears, it checks the `correlation_id` property. If it matches the value from the request it returns the response to the application.



4) Prepared Statement:

This technique is used to prevent the web application from sql injection.

```
*/var clean_user = sanitizer.sanitize(username);

 var post = {Username: clean_user, Password: hash};

var query = connection.query('INSERT INTO users SET ?', post, function(err, results)

{ // Can a Sql injection happen here? });*/
```