Student name	**Mohan Kumar Ramalingaiah Neerakallu**
Student ID	**19493389**
Course code	**COMP717**
Course Title	**Advanced Web Technologies**
Tutor name	**Sue Beale**
Assignment	**3 – Hotel Booking**
Due Date	**Week 16**
Date Submitted	**14-11-2020**
Case Study	**Alpha Hotel Booking with Angular and API**

This assignment is my own work:
Your name: _____Mohan Kumar_____
Case Study: ____Hotel Booking_____
Signature: _____Mohan Kumar_____

RESTful APIs

REST stands for Representational State Transfer and is an style of architecture for network communication between applications, which relies on a stateless protocol (usually HTTP) for interaction. (Castelo)

HTTP Verbs Represent Actions

In RESTful APIs, we use the HTTP verbs as actions, and therefore the endpoints are the resources acted upon. We'll be using the HTTP verbs for his or her semantic meaning:

GET: retrieve resources

POST: create resources

PUT: update resources

DELETE: delete resources

Laravel

Laravel may be a PHP framework developed with PHP developer productivity in mind. Written and maintained by Taylor Otwell, the framework is extremely opinionated and strives to save lots of developer time by favoring convention over configuration. The framework also aims to evolve with the online and has already incorporated several new features and concepts within the web development world—such as job queues, API authentication out of the box, real-time communication, and much more.

Update Action: PUT vs. POST

RESTful APIs are a matter of much debate and there are many opinions out there on whether is best to update with POST, PATCH, or PUT, or if the create action is best left to the PUT verb. during this Booking we'll be using PUT for the update action, as

consistent with the HTTP RFC, PUT means to create/update a resource at a selected location. Another requirement for the PUT verb is idempotence, which during this case basically means you'll send that request 1, 2 or 1000 times and therefore the result are going to be the same: one updated resource within the database.

Resources

Resources are going to be the targets of the actions, in our case Booking and Users, and that they have their own endpoints:

/Booking

/users

Setting Up a Laravel Web Service Project (Adelekan)

As with all modern PHP frameworks, we'll need Composer to put in and handle our dependencies. After you follow the download instructions (and increase your path environment variable), install Laravel using the command:

```
composer global require laravel/installer
```

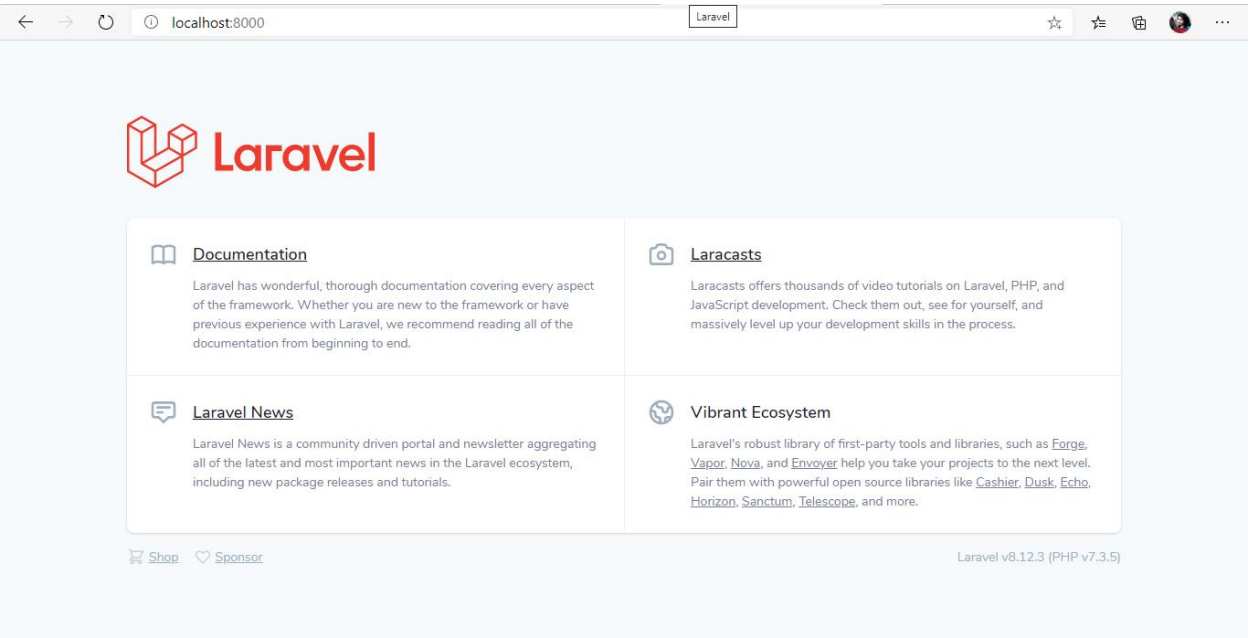
After the installation finishes, you'll scaffold a replacement application like this:

```
composer create-project --prefer-dist laravel/Laravel backend_api
```

With Laravel installed, you ought to be ready to start the server and test if everything is working:

```
$ php artisan serve
```

Laravel development server started:



When you open localhost:8000 on your browser, you ought to see this sample page.

Migrations and Models

Before actually writing your first migration, confirm you've got a database created for this app and add its credentials to the .env file located within the root of the project.

DB_CONNECTION=mysql

DB_HOST=127.0.0.1

DB_PORT=3306

DB_DATABASE=hotel

DB_USERNAME=root

DB_PASSWORD=

Create first model and migration—the Booking. The Booking should have a title and a body field, also as a creation date. Laravel provides several commands through Artisan—Laravel's instruction tool—that help us by generating files and putting them within the correct folders. to make the Booking model, we will run:

```
php artisan make:model Booking -m
```

The -m option is brief for --migration and it tells Artisan to make one for our model. Here's the generated migration:

```
<?php


use Illuminate\Database\Migrations\Migration;

use Illuminate\Database\Schema\Blueprint;

use Illuminate\Support\Facades\Schema;


class booking extends Migration

{

    /**

     * Run the migrations.

     *

     * @return void

     */

    public function up()

    {

          Schema::create('booking', function (Blueprint $table) {

            $table->increments('id');

            $table->timestamps();

        });

    }
```

```
    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::dropIfExists('booking');
    }
}
```

The up() and down() methods will be run when we migrate and rollback respectively;

$table->increments('id') sets up an auto incrementing integer with the name id;

$table->timestamps() will set up the timestamps for us—created_at and updated_at, but don't worry about setting a default, Laravel takes care of updating these fields when needed.

And finally, Schema::dropIfExists() will, of course, drop the table if it exists.

With that out of the way, let’s add two lines to our up() method:

```
    public function up()
    {
        Schema::create('booking', function (Blueprint $table) {
            $table->increments('id');
            $table->string('name');
```

```
            $table->String('email');
            $table->String('phone');
            $table->String('address');
            $table->String('city');
            $table->String('cat');
            $table->Integer('no');
            $table->timestamps();
        });
    }
```

The string() method creates a VARCHAR equivalent, text() creates a TEXT equivalent and Integer() Create the Number Equivalent . After that migrate:

```
php artisan migrate
```

migrations, Create the Booking table in database with field that are declare in up function.

Now let's go back to our model and add those attributes to the $fillable field so that we can use them in our Booking::create and Booking::update models:

```
class Booking extends Model
{
    protected $fillable = ['title', 'body'];
}
```

Fields inside the $fillable property can be mass assigned using Eloquent's create() and update() methods. You can also use the $guarded property, to allow all but a few properties.

Routes and Controllers

Let's create the basic endpoints for our application: create, retrieve the list, retrieve a single one, update, and delete. On the routes/web.php file, we can simply do this:

```
Use App\Booking;

 Route::get('/', function () {

    return view('welcome');

});
```

The routes inside api.php will be prefixed with /api/ and the API throttling middleware will be automatically applied to these routes (if you want to remove the prefix you can edit the RouteServiceProvider class on /app/Providers/RouteServiceProvider.php).

Now let's move this code to its own Controller:

```
php artisan make:controller BookingController
```

BookingController.php:

```
<?php


namespace App\Models;


use Illuminate\Database\Eloquent\Factories\HasFactory;

use Illuminate\Database\Eloquent\Model;

use DB;
```

```
class booking extends Model
{
    use HasFactory;


}
```

We can fix that by editing our exception handler class, located in app/Exceptions/Handler.php, to return a JSON response:

```
public function render($request, Exception $exception)
{
    // This will replace our 404 response with
    // a JSON response.
    if ($exception instanceof ModelNotFoundException) {
        return response()->json([
            'error' => 'Resource not found'
        ], 404);
    }


    return parent::render($request, $exception);
}
```

Here’s an example of the return:

```
{
    data: "Resource not found"
}
```

If you're using Laravel to serve other pages, you have to edit the code to work with the Accept header, otherwise 404 errors from regular requests will return a JSON as well.

Create a Route GET

GET Method is use to get the data

```
Route::get('/booking', 'App\Http\Controllers\UserControler@getBooking');
```

When we hit a /booking api its show all data available in booking table.

Then create a getBooking function in Usercontroller which is help to show the data.

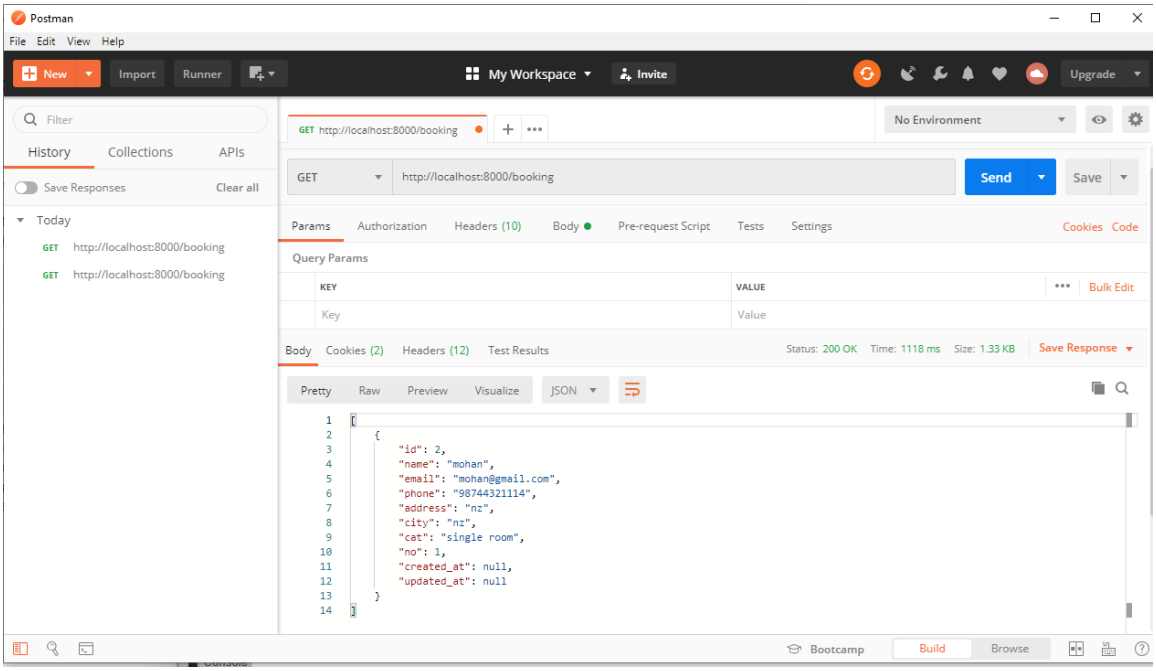
```
function getBooking()
  {
      $em=new Booking();
      $data=$em->getBooking();
      return response()->json($data);
    /*return view('Booking'); */
  }
```

Model:

```
function getBooking()
  {
    $db=DB::table('booking')->get();
    /*dd($db); */
    return $db;
  }
```

Test the route Get in Postman:

Postman is use to test api and check result is given or not.



Create a Route GET for single data

GET Method is use to get the data

```
Route::get('/booking/{id}', 'App\Http\Controllers\UserControler@getoneBooking');
```

When we hit a /booking/{id} api its show the specific id data available in booking table.

Then create a getOneBooking function in Usercontroller which is help to show the data.

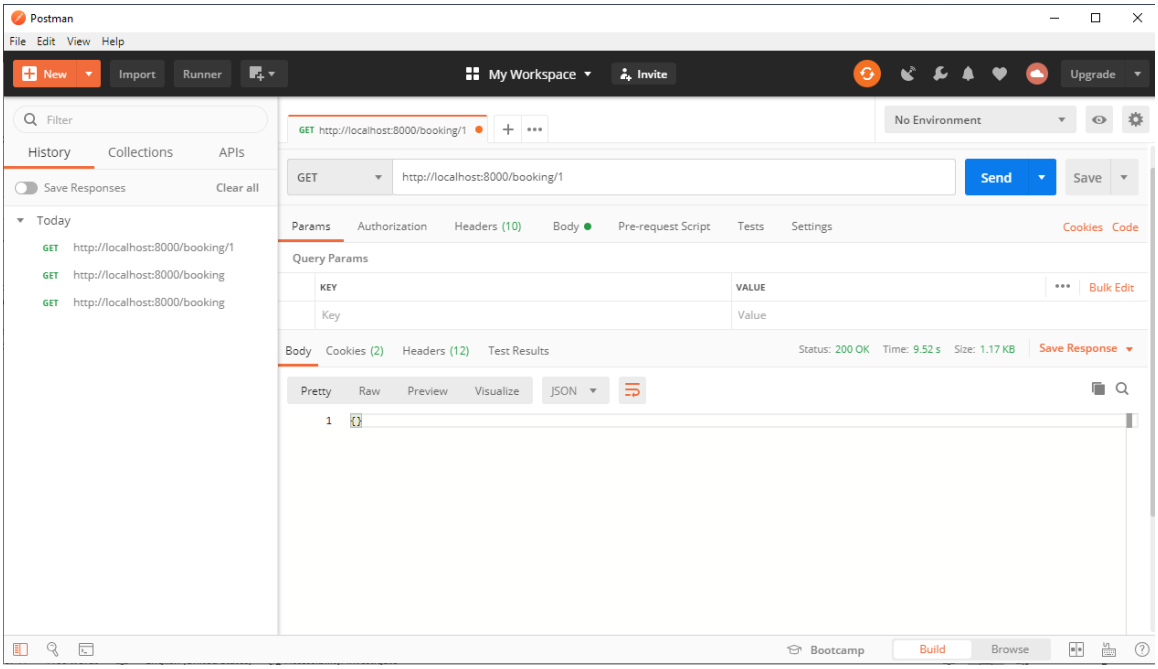
```
function getoneBooking(Request $request)
   {
      $id=$request->id;
      $em=new Booking();
      $data=$em->getoneBooking($id);
```

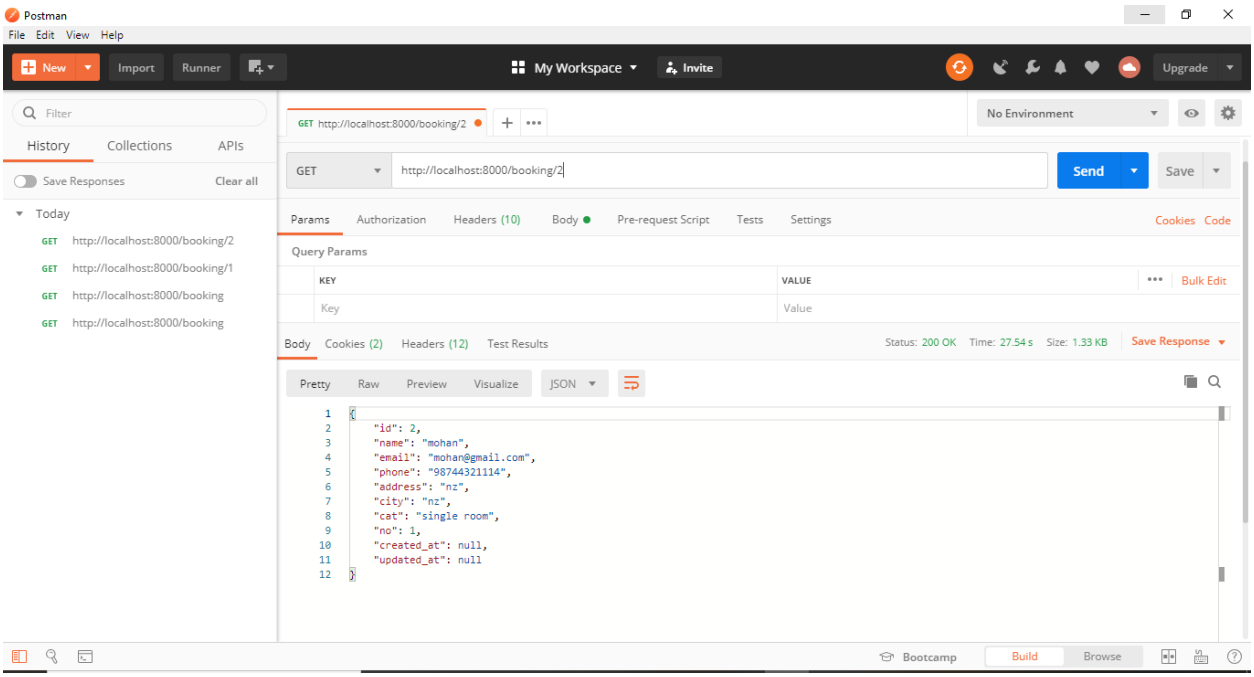
```
        return response()->json($data);
        /*return view('Booking'); */
    }
```

Model:

```
    function getoneBooking($id)
    {
        $db=DB::table('booking')->where('id',$id)->get()->first();
        /*dd($db); */
        return $db;
    }
```

Test the route Get in Postman:





Create a Route POST

POST Method is use to put the data into the database

```
Route::post('/addbooking', 'App\Http\Controllers\UserControler@addBooking');
```

When we hit a /addbooking api its put all data into available booking table.

Then create a addBooking function in Usercontroller which is help to put the data.

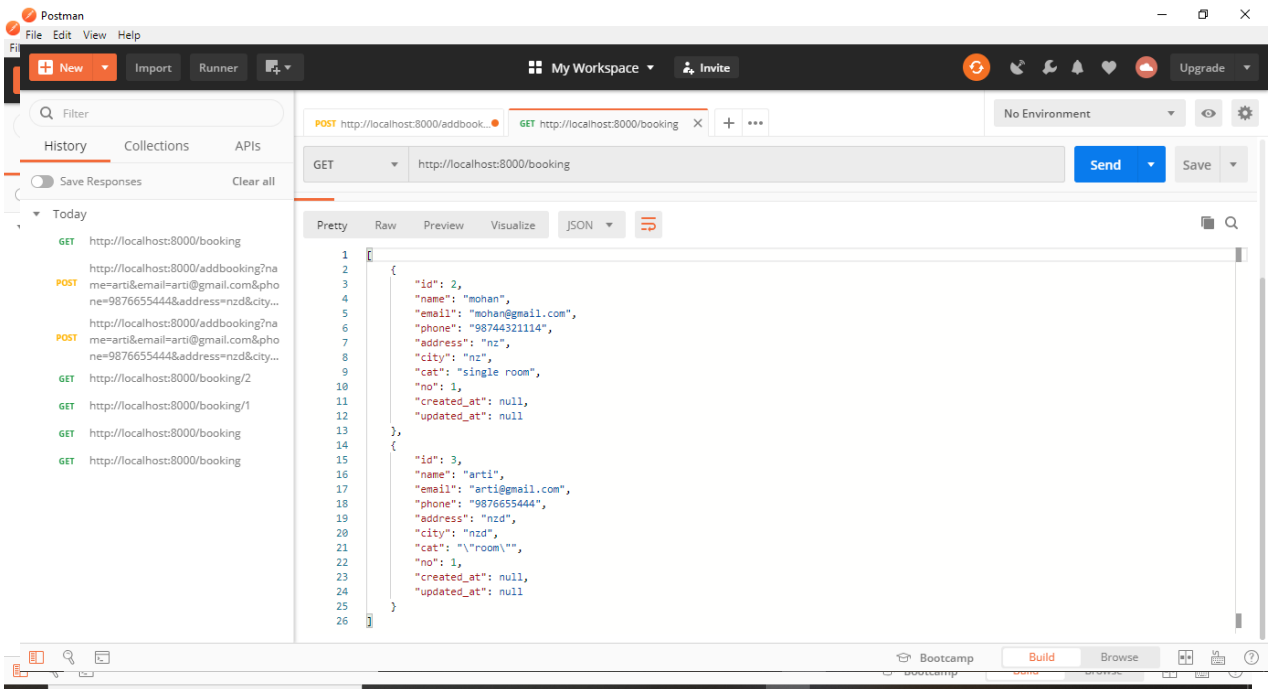
```
function addBooking(Request $request)
  {
      $em=new Booking();
      $data=$em->addBooking($request->all());
  }
```

Model:

```
function addBooking($data)
  {
     DB::table('booking')->insert($data);
     /*dd($db); */
  }
```

Test POST API In postman

Hit the POST Api /addbooking with all param.



Create a Route Delete

DELETE Method is use to Delete the specific data into the database

```
Route::delete('/deletebooking/{id}','App\Http\Controllers\UserControler@deleteBooking');
```

When we hit a /deletebooking/1 api its delete the specific data into available booking table.

Then create a deleteBooking function in Usercontroller which is help to put the data.

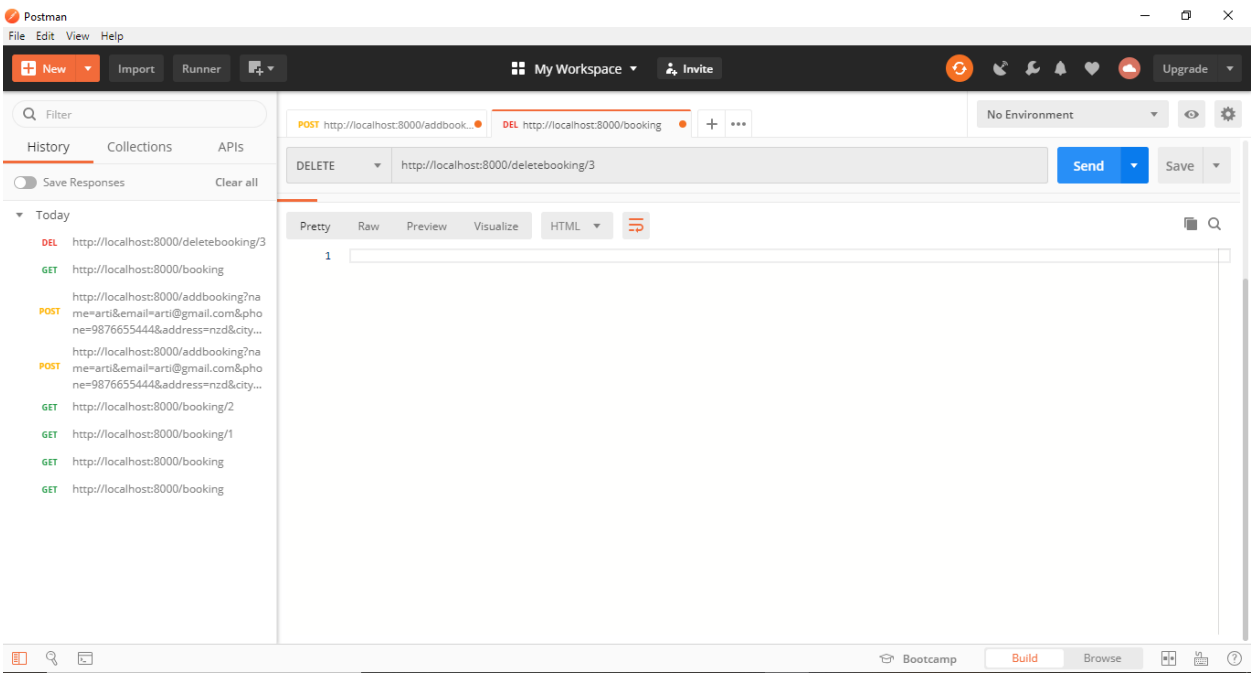
```
function deleteBooking(Request $request)
  {
                              $id=$request->id;
                              $em=new Booking();
                              $em->deleteBooking($id);
  }
```

Model:

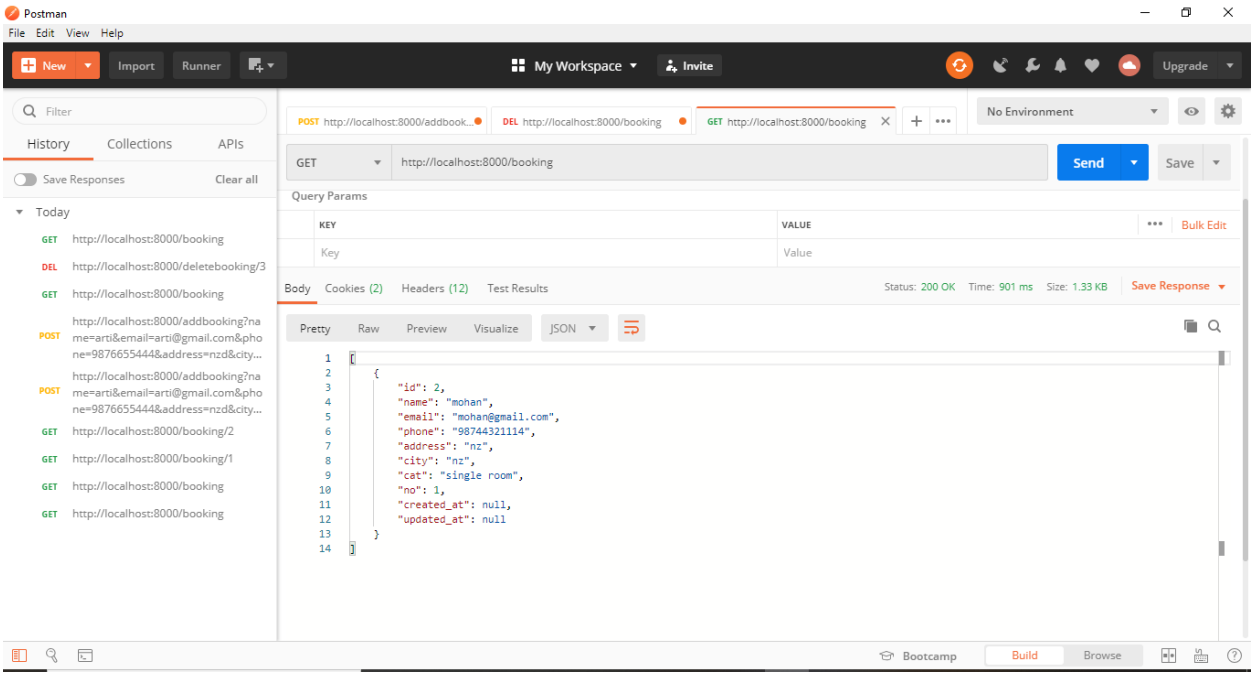
```
  function deleteBooking($id)
  {
     DB::table('booking')->where('id',$id)->delete();
     /*dd($db); */


  }
```

Test Delete API In postman

Hit the DELETE Api /deletebooking/1



After Deleteing:



Coding:

User.php

```
<?php


namespace App\Models;


use Illuminate\Contracts\Auth\MustVerifyEmail;
use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Foundation\Auth\User as Authenticatable;
use Illuminate\Notifications\Notifiable;
```

```
class User extends Authenticatable
{
    use HasFactory, Notifiable;

    /**
     * The attributes that are mass assignable.
     *
     * @var array
     */
    protected $fillable = [
        'name',
        'email',
        'password',
    ];

    /**
     * The attributes that should be hidden for arrays.
     *
     * @var array
     */
    protected $hidden = [
        'password',
        'remember_token',
    ];

    /**
     * The attributes that should be cast to native types.
```

```
     *
     * @var array
     */
    protected $casts = [
        'email_verified_at' => 'datetime',
    ];

public function catergories()
{
     return $this->hasMany(catergories::class, 'user_id');
}
}
```

Booking.php:

```
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;
use DB;
class booking extends Model
{
    use HasFactory;



    function addBooking($data)
```

```
{
    DB::table('booking')->insert($data);
    /*dd($db); */
}
function getBooking()
{
    $db=DB::table('booking')->get();
    /*dd($db); */
    return $db;
}
function updateBooking($id,$data)
{
    DB::table('booking')->where('id',$id)->update($data);
    /*dd($db); */

}
function getoneBooking($id)
{
    $db=DB::table('booking')->where('id',$id)->get()->first();
    /*dd($db); */
    return $db;
}
 function deleteBooking($id)
{
    DB::table('booking')->where('id',$id)->delete();
    /*dd($db); */

}
```

```
}
```

Web.php:

```
<?php

use Illuminate\Support\Facades\Route;

/*
|--------------------------------------------------------------------------
| Web Routes
|--------------------------------------------------------------------------
|
| Here is where you can register web routes for your application. These
| routes are loaded by the RouteServiceProvider within a group which
| contains the "web" middleware group. Now create something great!
|
*/

Route::get('/', function () {
    return view('welcome');
});




Route::post('/addbooking', 'App\Http\Controllers\UserControler@addBooking');
Route::get('/booking', 'App\Http\Controllers\UserControler@getBooking');
Route::delete('/deletebooking/{id}',
'App\Http\Controllers\UserControler@deleteBooking');
```

```
Route::patch('/updatebooking/{id}',
'App\Http\Controllers\UserControler@updateBooking');
Route::get('/booking/{id}', 'App\Http\Controllers\UserControler@getoneBooking');
```

BookingController:

```
<?php

namespace App\Http\Controllers;
use App\Http\Controllers\Controller;
use Illuminate\Http\Request;
use App\Models\Booking;
class UserControler extends Controller
{
   function addBooking(Request $request)
   {
                                                $em=new Booking();
                                                $data=$em->addBooking($request-
>all());
   }
   function getBooking()
   {
                                                $em=new Booking();
                                                $data=$em->getBooking();
                                                return response()->json($data);
      /*return view('Booking'); */
   }
```

```
    function getoneBooking(Request $request)
    {
        $id=$request->id;
        $em=new Booking();
        $data=$em->getoneBooking($id);
        return response()->json($data);
        /*return view('Booking'); */
    }
    function updateBooking(Request $request)
    {
        $id=$request->id;
        $em=new Booking();
        $data=$em->updateBooking($id,$request->all());
    }
    function deleteBooking(Request $request)
    {
                                                    $id=$request->id;
                                                    $em=new Booking();
                                                    $em->deleteBooking($id);
    }

}
```

Frontend Coding:

Route page:

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import { AboutComponent } from './components/about/about.component';
import { CategoriesComponent } from './components/categories/categories.component
';
import { ContactComponent } from './components/contact/contact.component';
import { RoomsComponent } from './components/rooms/rooms.component';
import { ServicesComponent } from './components/services/services.component';
import { UpdateBookingComponent } from './components/services/update-
booking/update-booking.component';
import { TaskManagerComponent } from './components/task-manager/task-
manager.component';

const routes: Routes = [
  { path: '', component: TaskManagerComponent },
  { path: 'rooms', component: RoomsComponent },
  { path: 'service', component: ServicesComponent },
  { path: 'contact', component: ContactComponent },
  { path: 'categories', component: CategoriesComponent },
  { path: 'about', component: AboutComponent },
  { path: 'update_booking/:id', component: UpdateBookingComponent },

];
@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Booking } from '../components/services/services.model';
@Injectable({
  providedIn: 'root'
})
export class TaskService {

  data: any;
  constructor(private httpClient:HttpClient) {
```

```
  }

  bookingData(data: any) {
    return this.httpClient.post('http://localhost:8000/addbooking',data);
  }
   getBookingData()
  {
    return this.httpClient.get('http://localhost:8000/booking');
  }
  getOneBooking(id: any)
  {
    return this.httpClient.get('http://localhost:8000/booking/'+id);
  }
  deleteBooking(id: any)
  {
    return this.httpClient.delete('http://localhost:8000/deletebooking/'+id);
  }
   updateBooking(id:any,data:any)
  {
    return this.httpClient.patch('http://localhost:8000/updatebooking/'+id,data);
   }



}
```

Booking.html

```
<div class="container">
    <div class="row">
        <div class="col-sm-6 mx-auto">
            <h1>Booking</h1>
            <form (ngSubmit)="bookingData()">
                <div class="form-group">
                    <label for="name">Enter Name:</label>
                    <input type="text" name="name" id="name"
                        class="form-control" [(ngModel)]="booking.name">
                </div>
                <div class="form-group">
                    <label for="name">Enter Email:</label>
                    <input type="email" name="email" id="Email"
                        class="form-control" [(ngModel)]="booking.email">
```

```
                </div>
                <div class="form-group">
                    <label for="name">Enter Phone:</label>
                    <input type="email" name="phone" id="Phone"
                        class="form-control" [(ngModel)]="booking.phone">
                </div>
                <div class="form-group">
                    <label for="name">Enter Address:</label>
                    <textarea name="address" id="address"
                        class="form-
control" [(ngModel)]="booking.address"></textarea>
                </div>
                <div class="form-group">
                    <label for="name">Enter City:</label>
                    <input type="city" name="city" id="city"
                        class="form-control" [(ngModel)]="booking.city">
                </div>
                <div class="form-group">
                    <label for="name">select category:</label>
                    <select class="form-control" name="cat"
                        [(ngModel)]="booking.cat">
                        <option disabled>Select category</option>
                        <option>Delux room</option>
                        <option>Single Room</option>
                        <option>Couple room</option>
                        <option>Family Room</option>
                    </select>
                </div>
                <div class="form-group">
                    <label for="name">Enter No of Room</label>
                    <input type="number" name="num" id="num"
                        class="form-control" [(ngModel)]="booking.no">
                </div>

                <button class="form-control bg-danger text-light mt-4 mb-4">Book
                    a Room</button>
            </form>
            <span id="response" class="">{{data3}}</span>
        </div>
    </div>
</div>
<div class="container p-5">
    <div class="row">

        <div class="col-sm-12 mx-auto">
```

```
            <h1 class="text-center">Booking Table</h1>
            <table width="100%" class="table bg-light">
                <tr>
                    <td>Sno</td>
                    <td>Name</td>
                    <td>Email</td>
                    <td>Phone</td>
                    <td>Address</td>
                    <td>City</td>
                    <td>Category</td>
                    <td>No of Room</td>
                    <td>update</td>
                    <td>delete</td>
                </tr>
                <tr *ngFor="let obj of dataArr">
                    <td scope="row">{{obj.id}}</td>
                    <td>
                        {{obj.name}}
                    </td>
                    <td>{{obj.email}}

                    </td>
                    <td>{{obj.phone}}

                    </td>
                    <td>{{obj.address}}

                    </td>
                    <td>{{obj.city}}

                    </td>
                    <td>{{obj.cat}}

                    </td>
                    <td>{{obj.no}}

                    </td>
                    <td><button class="btn
                            btn-primary"
                            routerLink="/update_booking/{{obj.id}}">Edit</button>
</td>
                    <td><button class="btn
                            btn-
danger" (click)="deleteBooking(obj.id)">Delete</button></td>
                </tr>
```

```
            </table>
        </div>
    </div>
</div>
```

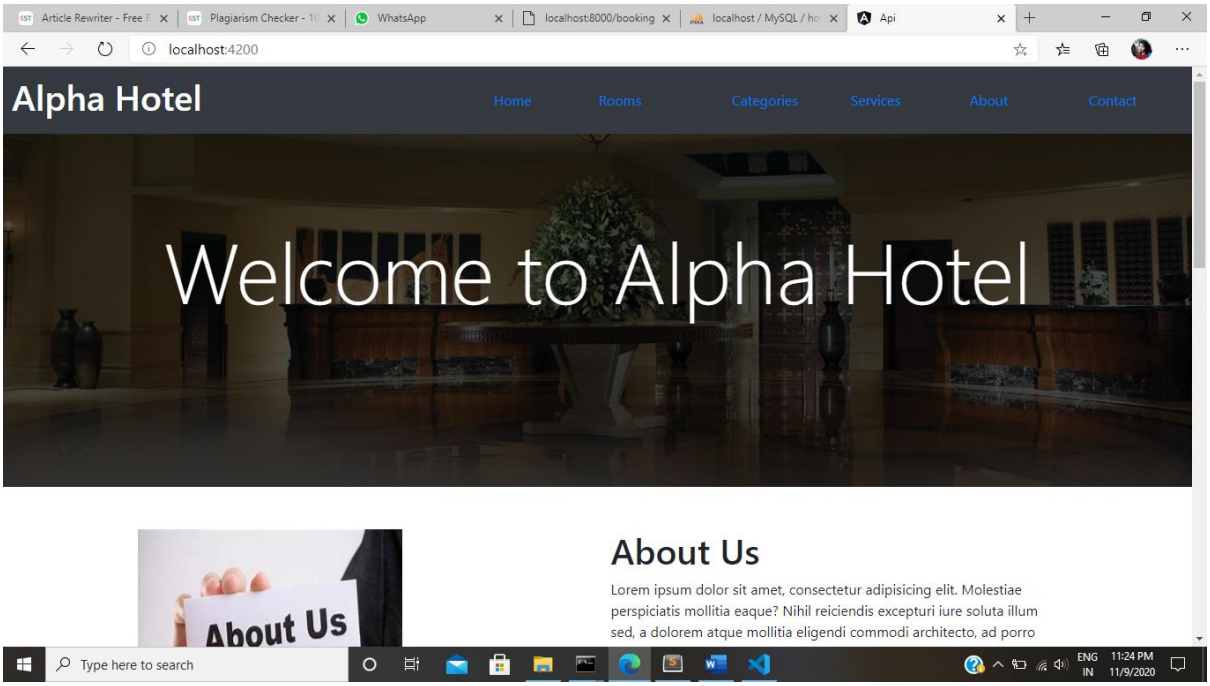
Update Booking:

```
<div class="container">
    <div class="row">
        <div class="col-sm-6 mx-auto">
            <h1 class="text-center">Update Category</h1>
            <form (ngSubmit)="updateBooking()">
                <div class="form-group">
                    <label for="name">Enter Name:</label>
                    <input type="text" name="name" id="name"
                        class="form-control" value="{{booking.name}}"
                        [(ngModel)]="booking.name">
                </div>
                <div class="form-group">
                    <label for="name">Enter email:</label>
                    <input type="text" name="email" id="email"
                        class="form-control" value="{{booking.email}}"
                        [(ngModel)]="booking.email">
                </div>
                <div class="form-group">
                    <label for="name">Enter Phone:</label>
                    <input type="email" name="phone" id="Phone"
                        class="form-control" value="{{booking.phone}}"
                        [(ngModel)]="booking.phone">
                </div>
                <div class="form-group">
                    <label for="name">Enter Address:</label>
                    <textarea name="address" id="address"
                        class="form-control" value="{{booking.address}}"
                        [(ngModel)]="booking.address"></textarea>
                </div>
                <div class="form-group">
                    <label for="name">Enter City:</label>
                    <input type="city" name="city" id="city"
                        class="form-control" value="{{booking.city}}"
                        [(ngModel)]="booking.city">
                </div>
                <div class="form-group">
                    <label for="name">Enter category:</label>
                    <select class="form-control" name="cat"
```

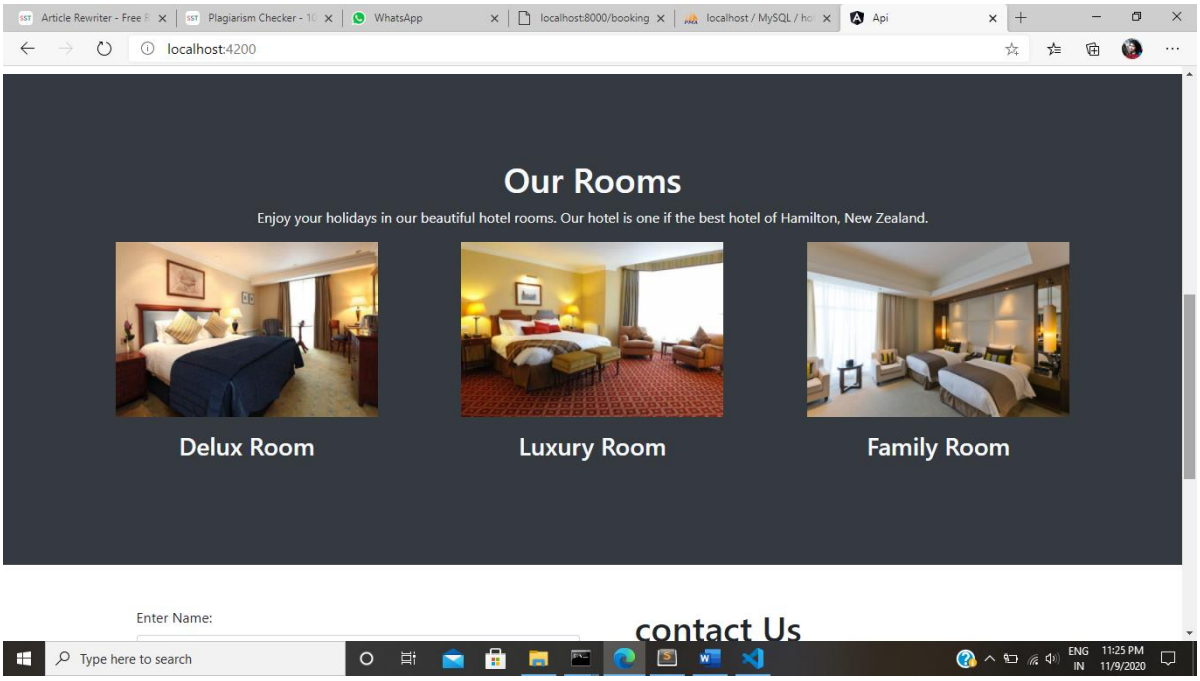
```
                        [(ngModel)]="booking.cat">
                        <option disabled>Select category</option>
                        <option>Couple Room</option>
                        <option>Single Room</option>
                        <option>Delax Room</option>
                    </select>
                </div>
                <div class="form-group">
                    <label for="name">Enter no of rooms:</label>
                    <input type="text" name="no" id="no"
                        class="form-control" value="{{booking.no}}"
                        [(ngModel)]="booking.no">
                </div>
                <button class="form-control bg-danger text-light mt-4 mb-
4">update
                    booking</button>
                <span>{{data3}}</span>
            </form>
        </div>
    </div>
</div>
```

Screenshots:

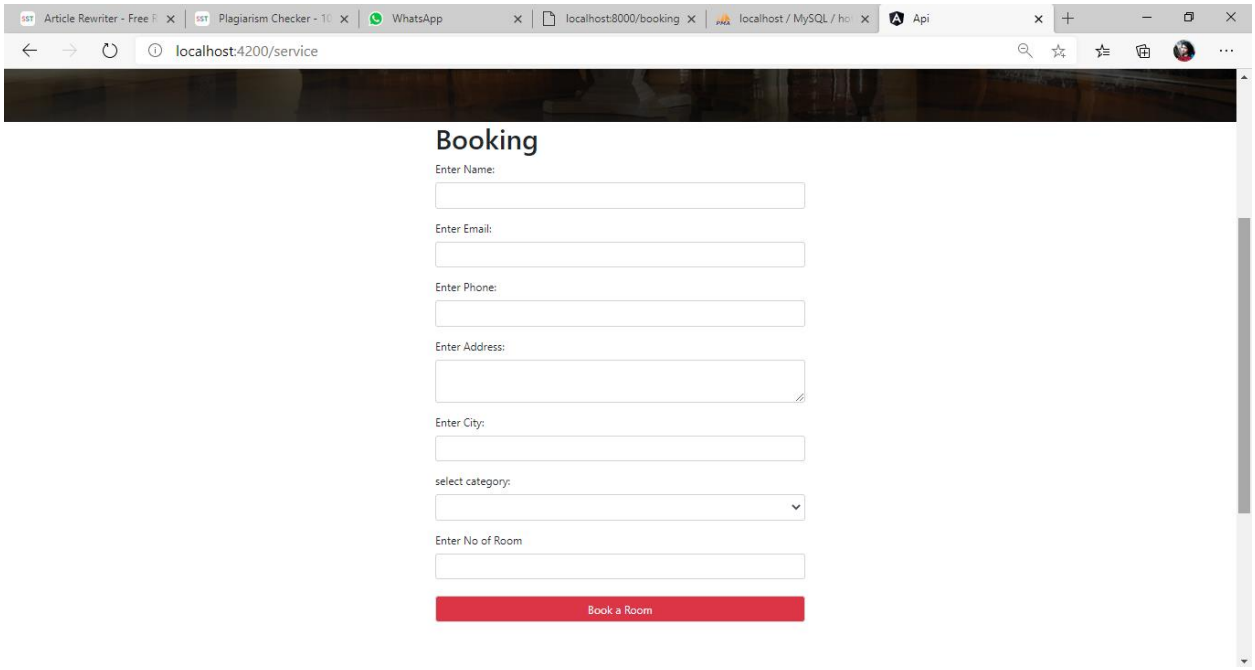
Index page:



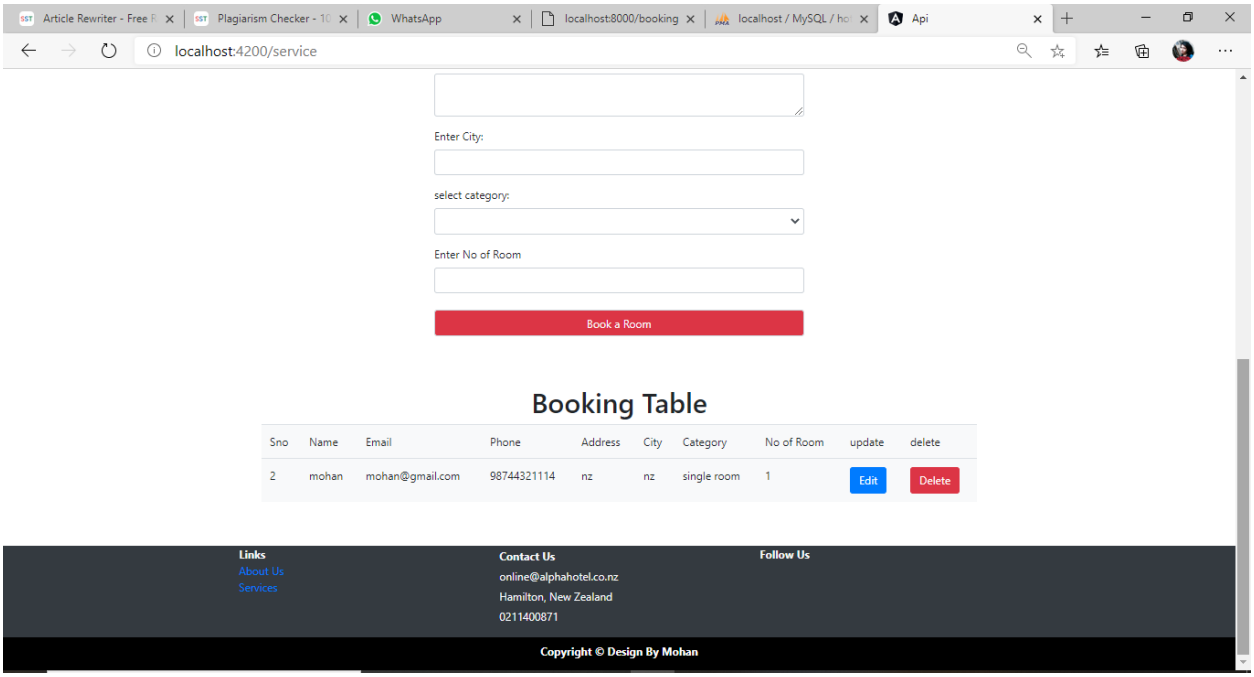
Room page:



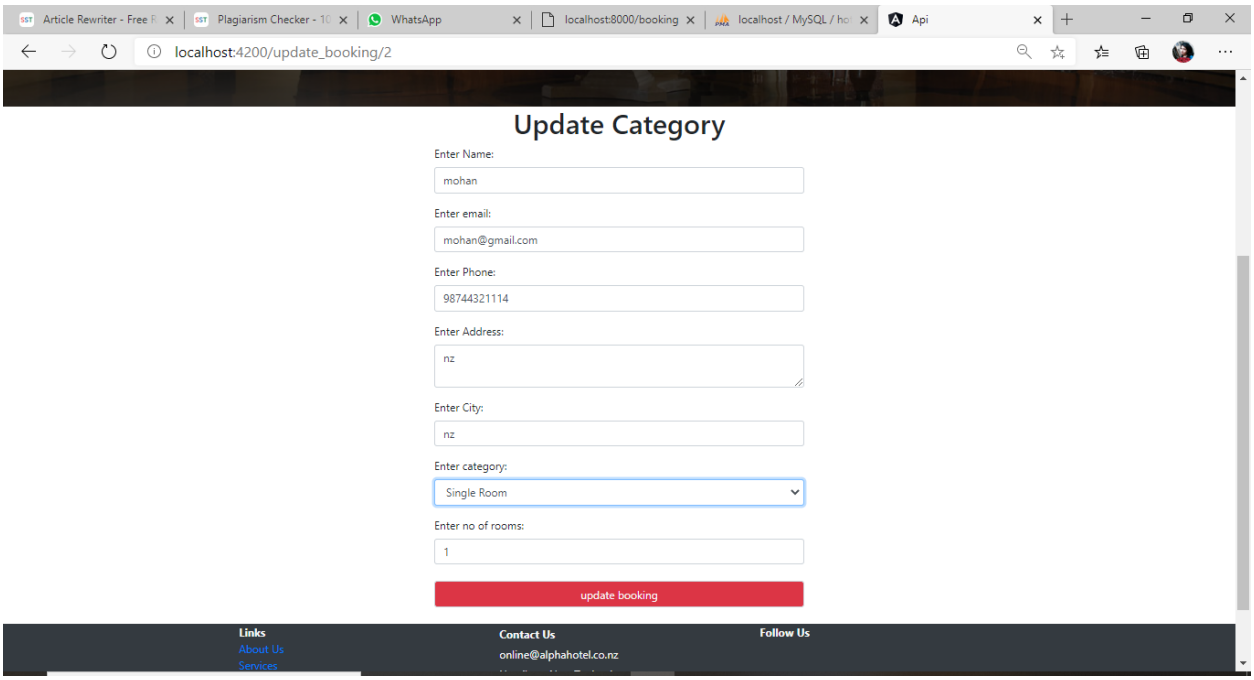
Add Booking:



Show table:



Update page:


localhost:4200/update_booking/2
Update Category
Enter Name:
mohan
Enter email:
mohan@gmail.com
Enter Phone:
98744321114
Enter Address:
nz
Enter City:
nz
Enter category:
Single Room
Enter no of rooms:
1
update booking
Links
About Us
Services
Contact Us
online@alphahotel.co.nz
Follow Us

References

Adelekan, David. [Online] https://medium.com/@adelekandavid2013/laravel-restful-api-development-a-step-by-step-approach-part-1-fdf9341662e6.

Castelo, André. [Online] https://www.toptal.com/laravel/restful-laravel-api-tutorial.