Hotel website using Laravel 5 and AngularJS

This tutorial is the first of a serie where we explain how to create an **hotel booking system** from scratch using **Laravel 5.1** and **AngularJS**. In the booking engine we will use AngularJS to create a **single page application**, the front end, where the final user can make **reservations** and the hotel owner can manage **rooms**, **price** and **reservations**. On the backend side we will use **Laravel** to create all the needed **REST API** to serve the application.

Data Model

To store all the needed information about the **hotel rooms, customers, reservations and prices** we need a total of 5 tables.

Let's create the migration file with Laravel artisan:

```
php artisan make:migration create_table_room_type
php artisan make:migration create_table_room_calendar
php artisan make:migration create_table_reservations
php artisan make:migration create_table_reservation_nights
php artisan make:migration create_table_customers
```

Starting from the first table "room_type", this table will contain the information about each room type the hotel can offer to it customers. For each room type we set a **name**, a **short name**, the **max occupancy** and some basic information about the room **availability** and night **rate**.

```
<?php

use Illuminate\Database\Schema\Blueprint;
use Illuminate\Database\Migrations\Migration;

class CreateTableRoomType extends Migration
{
    public function up()
    {
        Schema::create('room_types', function ($table) {
            $table->increments('id');
            $table->string('name', 100);
            $table->string('short_name',10);
            $table->float('base_price');
            $table->integer('base_availability');
            $table->integer('max_occupancy');
            $table->date('created_at');
            $table->date('updated_at');
        });

    }

    public function down()
    {
        Schema::drop('room_types');

    }
}
```

The second table in the list is the **room_calendar**. In this table we will store the**daily** information for each room type. Basically, for every room type, we will create a new row for each days with a reference to **room_type** and info about the rate, the availability and how reservation for a single date.

```
<?php

use Illuminate\Database\Schema\Blueprint;
use Illuminate\Database\Migrations\Migration;

class CreateTableRoomCalendar extends Migration
{
    public function up()
    {

        Schema::create('room_calendars', function ($table) {
            $table->increments('id');
            $table->integer('room_type_id');
            $table->integer('availability');
            $table->integer('reservations');
            $table->float('rate');
            $table->date('day');
            $table->date('created_at');
            $table->date('updated_at');
        });
    }


    public function down()
    {
        Schema::drop('room_calendars');

    }
}
```

The third is the **reservation table**. As the name suggest this table will store the information about a single **reservation** and a reference to the customer who booked the room.

```
<?php

use Illuminate\Database\Schema\Blueprint;
use Illuminate\Database\Migrations\Migration;

class CreateTableReservations extends Migration
{

    public function up()
    {
        Schema::create('reservations', function ($table) {
            $table->increments('id');
            $table->float('total_price');
            $table->integer('occupancy');
            $table->date('checkin');
            $table->date('checkout');
            $table->string('customer_id');
            $table->date('created_at');
            $table->date('updated_at');
        });
    }
```

```
    public function down()
    {
        Schema::drop('reservations');

    }
}
```

The fourth is the **reservation_nights**. Like the **room_calendar** this table store the information about every single night relative to a reservation. The stored information will be the price and the **room type**, this repeated for each reservation day. This table can be very useful if we need to calculate availability for a **room type** in a single day or to create a calendar where the hotel owner can take a look to his reservations.

```
<?php

use Illuminate\Database\Schema\Blueprint;
use Illuminate\Database\Migrations\Migration;

class CreateTableReservationNights extends Migration
{

    public function up()
    {
        Schema::create('reservation_nights', function ($table) {
            $table->increments('id');
            $table->float('rate');
            $table->date('day');
            $table->integer('room_type_id');
            $table->integer('reservation_id');
            $table->date('created_at');
            $table->date('updated_at');
        });
    }


    public function down()
    {
        Schema::drop('reservation_nights');
    }
}
```

The last table is self explanatory. Customer will store all the info about the final user that make a reservation.

```
<?php

use Illuminate\Database\Schema\Blueprint;
use Illuminate\Database\Migrations\Migration;

class CreateTableCustomers extends Migration
{

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

```
            $table->string('last_name');
            $table->string('email');
            $table->date('created_at');
            $table->date('updated_at');
        });
    }

    public function down()
    {
          Schema::drop('customers');

    }
}
```

That's all about the **database model**. All the created table contains a **minimal set of fields**, but can easily extended to store more info. Take the **customer** table as example, we stored just the name, last name and the email but can also be useful to store other info like the phone number, the title and the address, but in order to maintain the code of the tutorial short and readable we intentionally omitted this info.

Once all the file are in place you can execute a **migration** with the usual command

```
php artisan migrate
```

Laravel Model

To use the **ORM** functionality of **Laravel** each database table need to be mapped to a **Model class** in our application where we also can define the **relation between each model**.

```
php artisan make:model RoomType
php artisan make:model RoomCalendar
php artisan make:model Reservation
php artisan make:model ReservationNight
php artisan make:model Customer
```

We created a model class for each database table in the same order. The code below show the model file for each class where $fillable properties and the relationship between tables are declared.

```
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class RoomType extends Model
{
    protected $fillable = ['name',
'short_name','max_occupancy','base_price','base_availability'];

}



<?php
```

```
namespace App;

use Illuminate\Database\Eloquent\Model;

class RoomCalendar extends Model
{
    protected $fillable = ['room_type_id', 'rate','day'];

    function RoomType(){
        return $this->hasOne('App\RoomType');
    }
}
```

Every RoomCalendar store a reference to the RoomType and it's mandatory so we modeled it with an hasOne relationship.

```
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class Reservation extends Model
{
    protected $fillable = ['total_price',
'occupancy','checkin','checkout','name'];

    public function nights()
    {
        return $this->hasMany('App\ReservationNight');
    }

    function Customer(){
        return $this->belongsTo('App\Customer');
    }

}
```

Each **Reservation** need a relationship with a **Customer**, the one who made the reservation and a list of **ReservationsNights** where the price for each day is stored.

```
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class ReservationNight extends Model
{
    protected $fillable = ['rate', 'date', 'room_type_id'];

    function Reservation()
    {
        return $this->hasOne('App\Reservation');
    }

    function RoomType()
    {
```

```
        return $this->hasOne('App\RoomType');
    }
}
```

As the **Reservation** contain a reference to **multiple nights**, every night contain a reference to the **Reservation** and the **RoomType** for which it was created.

```
<?php

namespace App;

use Illuminate\Database\Eloquent\Model;

class Customer extends Model
{
    protected $fillable = ['first_name', 'last_name','email'];

}
```

The Laravel model is now in place. All the class are defined with it relationship so we can move to implement some first basics API to populate our tables.

The Room Type Controller

In a complex application like this we have to care about how to separate things without lose the purpose of each entity. Each **controller** has all the basics functions needed for a CRUD plus some other needed to implement application specific functionality.

The first controller we are going to create is the **RoomTypeController**. This controller is basically a **CRUD** controller where, for the moment, we implement just the **store** and **index** function.

```
php artisan make:controller RoomTypeController
```

```
<?php

namespace App\Http\Controllers;

use App\RoomType;
use Illuminate\Http\Request;

use App\Http\Requests;
use App\Http\Controllers\Controller;



class RoomTypeController extends Controller
{

    public function index()
    {
        $room_types = RoomType::all();
        return $room_types;
    }
```

```
    public function store(Request $request)
    {
        $room_type=new RoomType($request->all());
        $room_type->save();

        return $room_type;
    }
}
```

The Routing

As we said at begin of the tutorial we will have **two separate** sections of the application. One will be **public** to the **users** who want to **book a room** in the hotel and the other will be accessible by the **hotel owner** to take a look at the reservations or to simple change the price for a room.

To make things simple, for the moment, all the API will be accessible by everyone but can be useful to separate things from begin so when we will implement the authorization system for the hotel owner it will be more easy to modify the files.

So let's define our first route in the **route.php** file.

```
Route::group(['prefix' => 'adminapi'], function()
{
    Route::resource('room_type', 'RoomTypeController');

});
```

As you can see we defined a group of route with a **prefix** 'adminapi'. In this group we will put all the api accessible and needed by the **hotel owner**.

Starting Angular

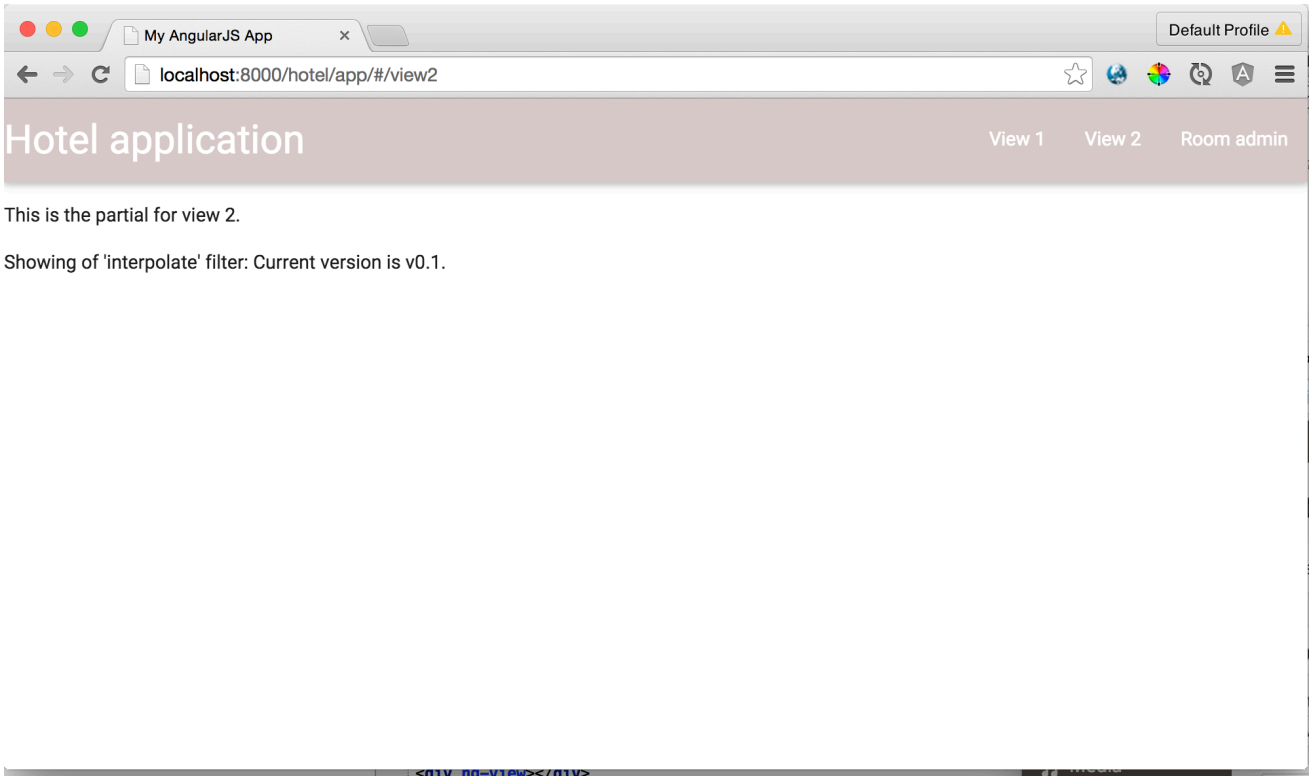
The frontend of the **booking engine** will be an **AngulaJS** single page application divided in **two section**, one for the final user and another for the **hotel administration.**

We will create the **Angular** application directly inside the **public folder** of the **Laravel** project. This is for convenience and also to maintain all the code in the same place under a single web server.

Once the you copy the package code into public folder, let's do some modification to the index.html file into the hotel directory. We need to load some more **javascript** and **CSS**. The final code of the **index.html** file will be something like this :

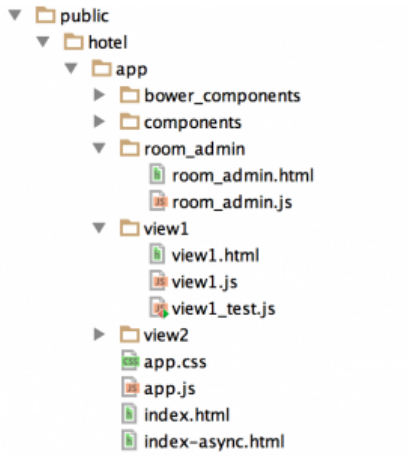
```
</html>
<!DOCTYPE html>
<!--[if lt IE 7]>      <html lang="en" ng-app="myApp" class="no-js lt-ie9 lt-ie8
lt-ie7"> <![endif]-->
<!--[if IE 7]>         <html lang="en" ng-app="myApp" class="no-js lt-ie9 lt-
ie8"> <![endif]-->
<!--[if IE 8]>         <html lang="en" ng-app="myApp" class="no-js lt-ie9">
<![endif]-->
<!--[if gt IE 8]><!--> <html lang="en" ng-app="myApp" class="no-js"> <!--
<![endif]-->
<head>
  <meta charset="utf-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <title>My AngularJS App</title>
  <meta name="description" content="">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="bower_components/html5-
boilerplate/dist/css/normalize.css">
  <link rel="stylesheet" href="bower_components/html5-
boilerplate/dist/css/main.css">
  <link rel="stylesheet"
href="bower_components/materialize/dist/css/materialize.css">

  <link rel="stylesheet" href="app.css">
  <script src="bower_components/html5-boilerplate/dist/js/vendor/modernizr-
2.8.3.min.js"></script>
</head>
<body>



<nav class="brown lighten-4">
  <div class="nav-wrapper brown lighten-4 ">
    <a href="#" class="brand-logo"> Hotel application</a>
    <ul id="nav-mobile" class="right hide-on-med-and-down">
      <li><a href="#/view1">View 1</a></li>
      <li><a href="#/view2">View 2</a></li>
      <li><a href="#/room_admin">Room admin</a></li>
    </ul>
  </div>
</nav>

<div ng-view></div>

<script src="bower_components/jquery/dist/jquery.js"></script>
<script src="bower_components/materialize/dist/js/materialize.js"></script>
<script src="bower_components/angular/angular.js"></script>
<script src="bower_components/angular-route/angular-route.js"></script>
<script src="bower_components/angular-materialize/src/angular-
materialize.js"></script>
<script src="app.js"></script>



<script src="view1/view1.js"></script>
<script src="view2/view2.js"></script>
<script src="components/version/version.js"></script>
<script src="components/version/version-directive.js"></script>
<script src="components/version/interpolate-filter.js"></script>

</body>
</html>
```

If all is ok if you navigate now to the same URL you will see something like this in your browser



AngularJS Room Controller

All is set and we can start by writing some **Javascript**. As the first thing we are going to create the room administration page. We alredy defined in the Laravel controller the**REST API** to add a new room type. We miss the **AngularJS frontend.**



The seed we used propose a good folder and application structure so we continue using it. Create a new folder under the app called **room_admin**and put inside it the template and the javascript file with the same name as the folder.

We need to import the created js file in the **index.html**. Add after the import of **app.js** in the **index.html** file the row below

```
<script src="room_admin/room_admin.js"></script>
```

In the template html file of the **room_admin** we will need a form to create a new room_type and another to set the price for a dates range. The **backend Laravel** part for the price set will be seen later.

The complete code of the **room_admin.html**

```
<div class="row">
    <h3 class="teal-text center">Room admin</h3>
</div>
<div class="row">
    <div class="col m6 s12">
        <div class="card-panel">
            <span class="card-title"><h5>Create new room Type</h5></span>
            <div class="row">
                <div class="input-field col s6">
                    <input ng-model="new_type.name" id="name" type="text" >
                    <label for="name">Room Name</label>
                </div>
                <div class="input-field col s6">
                    <input ng-model="new_type.short_name" id="short_name"
type="text" >
                    <label for="short_name">Short Name</label>
                </div>
            </div>
            <div class="row">
                <div class="input-field col s6">
                    <input ng-model="new_type.base_price" id="base_price"
type="text" >
                    <label for="name">Base price</label>
                </div>
                <div class="input-field col s6">
                    <input ng-model="new_type.base_availability"
id="base_availability" type="text" >
                    <label for="base_availability">Base availability</label>
                </div>
            </div>
            <div class="row">
                <div class="input-field col s6">
                    <input ng-model="new_type.max_occupancy" id="max_occupancy"
type="text" >
                    <label for="max_occupancy">Max occupancy</label>
                </div>
            </div>
            <button class="btn" ng-click="create()">Create</button>
        </div>
    </div>

    <div class="col m6 s12">
        <div class="card-panel">
            <span class="card-title"><h5>Set price for room Types</h5></span>
            <div class="collection">
                <a href="" ng-repeat="room_type in room_types" ng-class="{active:
$index==selected_idx}" class="collection-item" ng-
click="select_type($index)">{{room_type.name}}</a>
            </div>
            <div class="row" ng-if="selected_idx>=0">
                <span class="card-title">Set price for {{
selected_type.name}}</span>
                <div class="input-field">
                    <label for="start_dt">Start Date</label>
                    <input input-date type="text" id="start_dt" ng-
model="update_data.start_dt"  on-set="onStartSet()"  />
                </div>
```

```
                <div class="input-field">
                    <label for="end_dt">End Date</label>
                    <input input-date type="text"  id="end_dt" ng-
model="update_data.end_dt" on-set="onEndSet()" />
                </div>
                <div class="input-field">
                    <label for="rate">Price</label>
                    <input type="text"  id="rate" ng-
model="update_data.price"  />
                </div>
                <button class="btn" ng-click="setprice()">Set</button>
            </div>
            <div class="row" ng-if="message!=null">
                <div class="card-panel">
                    <span class="blue-text text-darken-2">{{message}}</span>
                </div>
            </div>
        </div>
    </div>
</div>
```

In the view we have a first form to create the **new room types** then on the right side we have a l**ist of already existent room** type. We want to open the price set form only when a room type is selected so for each item in the list we call the function **select_type($index)**. To hide or show the price set form we used a **ng-if** directive drive by the **selected_idx** variable.

The controller for the room administrator is all about calling **REST api** to the backend.

```
'use strict';

angular.module('myApp.roomAdmin', ['ngRoute','ui.materialize'])

    .config(['$routeProvider', function($routeProvider) {
      $routeProvider.when('/room_admin', {
        templateUrl: 'room_admin/room_admin.html',
        controller: 'RoomAdminCtrl'
      });
    }])

    .controller('RoomAdminCtrl', function($scope,$http,$location) {

      $scope.room_types = [];
      $scope.new_type = {};
      $scope.selected_idx;
      $scope.selected_type= {};
      $scope.message=null;
      $scope.update_data= {};

      $scope.init = function(){
        $http.get('/adminapi/room_type').success(function(data){
          $scope.room_types=data;
        });
      }

      $scope.init();

      $scope.create = function(){
        $http.post('/adminapi/room_type',$scope.new_type).success(function(data){

          $scope.room_types.push(data);
```

```
          $scope.new_type.name="";
          $scope.new_type.short_name="";
          $scope.new_type.base_price="";
          $scope.new_type.base_availability="";
          $scope.new_type.max_occupancy="";

        });
      };

      $scope.select_type = function(index){
        $scope.selected_idx=index;
        $scope.selected_type =$scope.room_types[index];
        $scope.update_data.room_type= $scope.selected_type.id;
        $scope.message = null;

      };

      $scope.setprice = function(){

        $http.post('/adminapi/setpriceinrange',$scope.update_data).success(functi
on(data){
          $scope.selected_idx=-1;
          $scope.selected_type=null;
          $scope.update_data={};

          $scope.message = data;
        });

      }
    }
);
```

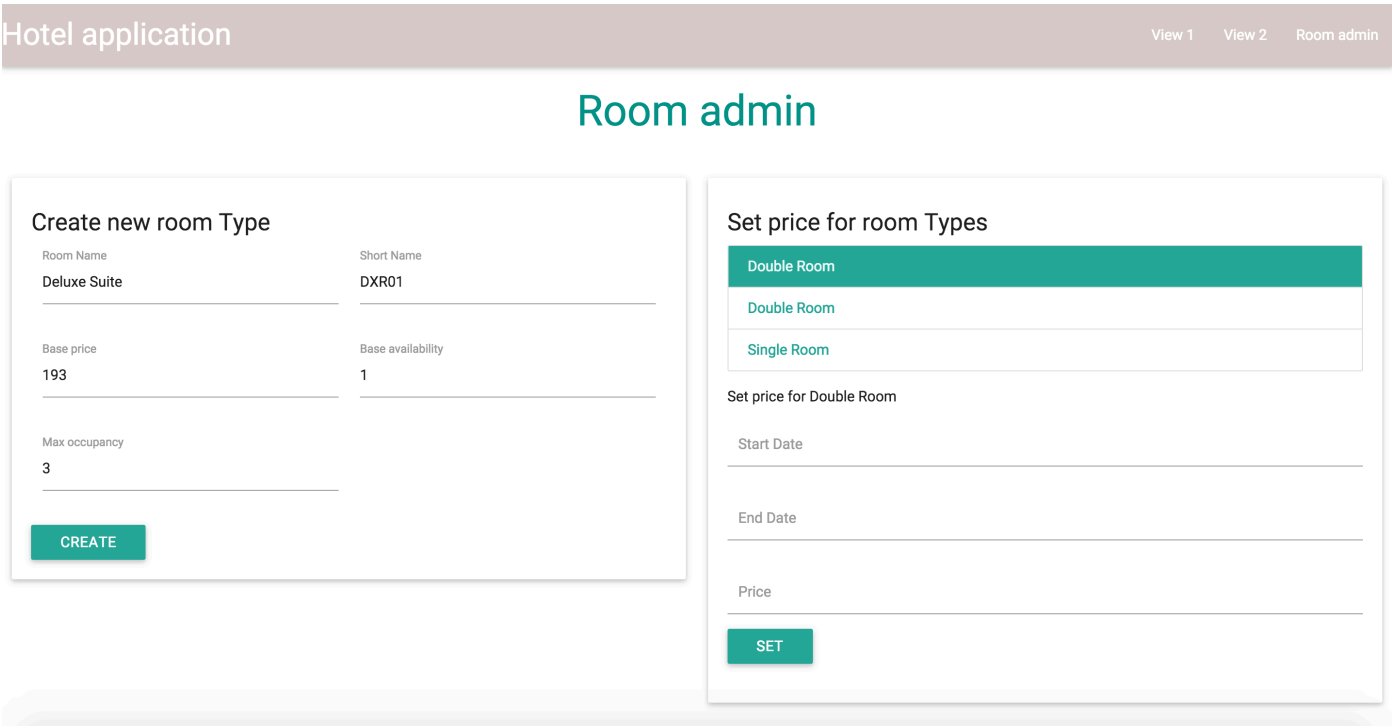
roller and the used **template**. In the controller section we have **3 functions** and some variables. The **init()** function call the Laravel backend to retrieve all the already created room type. The create function make a post call to the Laravel backend. As we defined this function to return the just created type we push the callback data to the **room_type** array.

select_type() function and **setprice()** serve the right form of the page. The first one will set the variable to show the form and to highlight via the ng_class directive the selected room type. The last function make a **POST REST call** to set the price for the room in a date interval populating the message variable with the result of the operation.

We need a last thing in order to make the **Angular application** working. We need to import the created module in the main application. Will be enough to add a line in the**app.js** (i've also modified the standard page redirect to speed up testing)

```
'use strict';

// Declare app level module which depends on views, and components
angular.module('myApp', [
  'ngRoute',
  'myApp.view1',
  'myApp.view2',
  'myApp.version',
  'myApp.roomAdmin'
]).
config(['$routeProvider', function($routeProvider) {
  $routeProvider.otherwise({redirectTo: '/room_type'});
}]);
```

If you browse to http://localhost:8888/hotel/app/index.html#/room_admin now you will get something like shown in the screenshot below. You can also try to use the left form to insert some room types that will be added without reload to the right list. For the right form all is working but we miss the REST API on the Laravel backend side. Let's implement it.



Laravel backend

In the last tutorial we created only the room controller. Now it's time to create another**controller** to code the api about the price change for a room type. From the command line

```
php artisan make:controller RoomCalendarController
```

This **controller** will contain all the function relative to the room type by days. The first function we need is to set the price for a specific room type in a date range. This will be a post call with a **Json payload** where the start date, the end date and the price will be specified.

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

use App\Http\Requests;
use App\Http\Controllers\Controller;

class RoomCalendarController extends Controller
{
    public function setPriceInRangeForRoomType(Request $request)
    {
        $room_type = $request['room_type'];
        $price =  $request['price'];
        $start_dt =  $request['start_dt'];
        $end_dt =  $request['end_dt'];
        $date = date ("Y-m-d",strtotime($start_dt));
```

```
        $base_room = RoomType::find($room_type);

        $i=0;

        while (strtotime($date) <= strtotime($end_dt)) {
            $room_day =  RoomCalendar::firstOrNew(array('room_type_id' =>
$room_type, 'day'=>$date));

            if(!$room_day->id){
                $room_day->availability = $base_room->base_availability;
            }

            $room_day->rate = $price;
            $room_day->save();
            $date = date ("Y-m-d", strtotime("+1 day", strtotime($date)));
            $i++;
        }

        return response("Success updated ".$i." dates",200);
    }

}
```

The function in the code above is easy to understand. First it will look for the **RoomType** and the loop trough the dates to set the price for a specific day. The response of the function is a **HTTP 200** with a message indicating how many dates where updated. We also set the availability of the room for the day where the price was never set. This is important because we don't want to **overwrite** availability for the days already in the database, probably where the **availability and reservation** fields where modified.

Search for a room

We start this part of the tutorial from the **AngularJS** side of the booking feature. Create a new folder in the **public/app** called **book** and then create two files inside it **book.html**and **book.js.** The first page of the booking system will be a form with two dates and an occupancy field where the user can set the check-in and check-out dates and the needed occupancy, plus a button to start the search.

The bottom side of the page will be used to list the results of the **search.** For each room, the name and the total reservation price will be presented. From here the customer can start the booking process for the room he select.

```
<div class="row">
    <h2 class="center">Search your room</h2>
</div>
<div class="row">
    <div class="col m6 s12 offset-m3">
        <form>
        <div class="card-panel">
            <div class="row">
                <div class="col m5 s12">
                    <div class="input-field">
                        <label for="start_dt">Check-in date</label>
                        <input input-date type="text" id="start_dt" ng-
model="search_param.start_dt"  format="d-m-yyyy"/>
                    </div>
                </div>
                <div class="col m5 s12">
```

```
                    <div class="input-field">
                        <label for="end_dt">Check-in date</label>
                        <input input-date type="text" id="end_dt" ng-
model="search_param.end_dt"  format="d-m-yyyy"  />
                    </div>
                </div>
                <div class="col m2 s12">
                    <label for="occupancy">Persons</label>
                    <select class="" id="occupancy" ng-
model="search_param.min_occupancy" material-select>
                        <option ng-repeat="value in select_occupancy"
value="{{value}}">{{value}}</option>
                    </select>
                </div>
            </div>
            <div class="row">
                <div class="col m4 offset-m4">
                    <button ng-click="search()" class="btn">Search</button>
                </div>
            </div>
         </div>
        </form>
    </div>
</div>

<div class="row">
    <div class="col s12 m6" ng-repeat = "room_type in available_room_types">
        <div class="card blue-grey darken-1">
            <div class="card-content white-text">
                <span class="card-title">{{room_type.name}}</span>
                <p>Total price : {{room_type.total_price}}</p>
            </div>
            <div class="card-action">
                <a href="" ng-click="book($index)">Book this</a>
            </div>
        </div>
    </div>
</div>
```

Let's start coding the controller. First of all we need, as always, to declare our module in the **app.js** file and load it in the **index.html.**

```
<script src="book/book.js"></script>
```

```
angular.module('myApp', [
  'ngRoute',
  'myApp.view1',
  'myApp.view2',
  'myApp.version',
  'myApp.roomAdmin',
  'myApp.book'
]).
```

The first function we want in the controller is the **search().** This function does nothing more then calling the **Laravel** backend and store the returned information in an array of the available room types.

```
'use strict';

angular.module('myApp.book', ['ngRoute','ui.materialize'])

    .config(['$routeProvider', function($routeProvider) {
        $routeProvider.when('/book', {
            templateUrl: 'book/book.html',
            controller: 'BookController'
        });
    }])

    .controller('BookController', function($scope,$http,$location) {


        $scope.select_occupancy = [1,2,3,4,5];
        $scope.available_room_types;
        $scope.search_param={};

        $scope.search = function(){

            $http.post('/api/searchavailabity',$scope.search_param).success(funct
ion(data){
                $scope.available_room_types = data;
            });

        };

    }
);
```

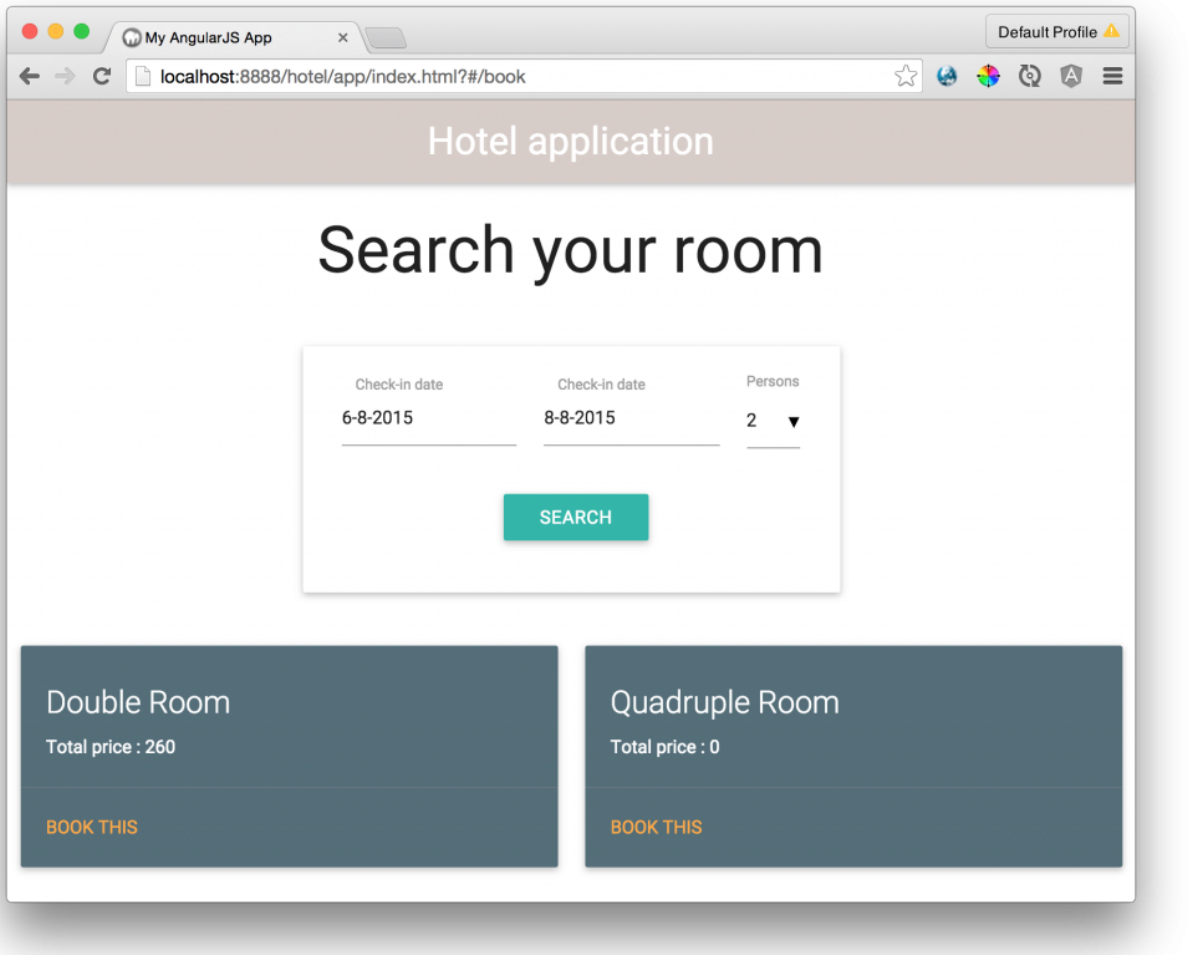
The first page of the application is complete but, as you can see, in the controller we are calling a function we never implemented in the **Laravel backend**. The **searchavailability**function will be coded in the **RoomCalendarController** on the **Laravel** side. Add the following code to the **Laravel controller** :

```
public function searchAvailability(Request $request){

        $start_dt = Carbon::createFromFormat('d-m-Y', $request['start_dt'])-
>toDateTimeString();
        $end_dt =  Carbon::createFromFormat('d-m-Y', $request['end_dt'])-
>toDateTimeString();

        $min_occupancy = $request['min_occupancy'];
        $room_types = RoomType::where('max_occupancy','>=',$min_occupancy)-
>get();

        $available_room_types=array();

        foreach( $room_types as $room_type){

            $count = RoomCalendar::where('day','>=',$start_dt)
                ->where('day','<',$end_dt)
                ->where('room_type_id','=',$room_type->id)
                ->where('availability','<=',0)->count();

            if($count==0){
```

```
                $total_price = RoomCalendar::where('day','>=',$start_dt)
                    ->where('day','<',$end_dt)
                    ->where('room_type_id','=',$room_type->id)
                    ->sum('rate');

                $room_type->total_price = $total_price;
                array_push($available_room_types,$room_type);
            }
        }

        return $available_room_types;
    }
```

The code above need to be explained. The first query is made on the **room_type** table to find which room can accomodate the number of person stored in the min_occupancy variable. For each room type that satisfy the **min_occupancy** condition another query is made on the room calendar table to find where this room have an **availability equal to zero** in the dates interval selected by the user. If the count is zero, that mean there is at least one available, for the **room_type** an aggregate query is made to calculate the total price. When all the info is gathered, the **room_type** is pushed in the**$available_room_type** array and returned to the client. Note we used the **Carbon class**to operate with the date in Laravel.

Now if you try the application, you will get something like the screen below.



Book the room

When a **customer** select a **room** to book it, the first thing we need is to gather his information, like the name, the email and others. In our application we want a separate view to gather this information but if we change the view we lose all the reservation information we gathered till now.

To store the information and get it back when needed in another view we can use a service.
The **service** we are going to create is a simple **hashtable** with two functions, one to store and the other to retrieve the information. The code of the service can be placed directly inside the **app.js.**

```
.service('reservationData', function () {

  var hashtable = {};

  return {
    setValue: function (key, value) {
      hashtable[key] = value;
    },
    getValue: function (key) {
      return hashtable[key];
    }
  }
});
```

Now when the user click the "**Book this**" link of a room card, the book function will be called. The book function store all the information about **check-in, check-out** and **price**of the reservation and redirect the user to another screen. Remember to injected the reservationData service in the controller.

```
.controller('BookController', function($scope,$http,reservationData,$location)
```

Below the code for the book function in the **BookController.**

```
$scope.book = function (id) {

            reservationData.setValue('start_dt',$scope.search_param.start_dt);
            reservationData.setValue('end_dt',$scope.search_param.end_dt);
            reservationData.setValue('occupancy',$scope.search_param.min_occupanc
y);
            reservationData.setValue('room_info',
$scope.available_room_types[id]);

            console.log(reservationData);

            $location.path( "/finalize" );

        };
```

As we said before we are going to create another view to gather information about the customer and finalize the reservation. In the same **book folder** create other two files**finalize.js** and **finalize.html.** To load the view and the Js,as always, we have to insert the declaration of the module in the **app.js** and load the js file in the **index.html.**

```
'myApp.finalize'
<script src="finalize/finalize.js"></script>
```

The **HTML** file in the first section will present a recap of the reservation and right to it, a form where the user can complete with his information and finalize the reservation. The complete code of the page is shown below

```
<div class="row">
    <div class="col s12 m6" ng-if="!reser_done">
        <div class="card blue-grey darken-1">
            <div class="card-content white-text">
                <span class="card-title">Booking recap</span>
                <p>Check-in  : {{start_dt}}</p>
                <p>Check-out  : {{end_dt}}</p>
                <p>Passenger : {{occupancy}}</p>
                <p>Room : {{room_info.name}}</p>
                <p>Total price : {{room_info.total_price}}</p>
            </div>
        </div>
    </div>
    <div class="col s12 m6" ng-if="!reser_done">
        <div class="card-panel">
            <form >
                <h4>Customer info</h4>
                <div class="row">
                    <div class="input-field col s6">
                        <input  id="first_name" type="text" ng-
model="customer.first_name" >
                        <label for="first_name">First Name</label>
                    </div>
                    <div class="input-field col s6">
                        <input id="last_name" type="text" ng-
model="customer.last_name" >
                        <label for="last_name">Last Name</label>
                    </div>
                </div>
                <div class="row">
                    <div class="input-field col s12">
                        <input id="email" type="email" ng-model="customer.email"
>
                        <label for="email">Email</label>
                    </div>
                </div>
                <button class="btn" ng-click="book()">Book</button>
            </form>
        </div>
    </div>
</div>

<div class="row">
    <div class="col s12 m4 offset-m4" ng-if="reser_done">
        <div class="card blue darken-2">
            <div class="card-content white-text">
                <span class="card-title"> <h2>Reservation Done</h2></span>
                <h4>Customer info</h4>
                <p>Name : {{reservation_info.customer.first_name}}</p>
                <p>Last name : {{reservation_info.customer.last_name}}</p>
                <p>Email : {{reservation_info.customer.email}}</p>
                <h4>Reservation info</h4>
                <p>Check-in : {{reservation_info.checkin}}</p>
                <p>Check-out : {{reservation_info.checkout}}</p>
                <p>Adults : {{reservation_info.occupancy}}</p>
                <p>Total Price : {{reservation_info.total_price}}</p>
                <h4>Nights info</h4>
                <p ng-repeat="night in reservation_info.nights">
```

```
                    Day {{night.day}}  -  Rate : {{night.rate}}
                </p>
            </div>
        </div>
    </div>
</div>
```

We defined two row hidden or shown by **ng-if directive** on a conditional **reser_done**variable. When the reservation is done and the info about it was retrieved from the backend a recap of the complete booking will be shown completed by all the info about**customer**, **room**, and **night rates.**

The code for the controller of the **finalize** page:

```
/**
 * Created by andreaterzani on 06/08/15.
 */
'use strict';

angular.module('myApp.finalize', ['ngRoute','ui.materialize'])

    .config(['$routeProvider', function($routeProvider) {
        $routeProvider.when('/finalize', {
            templateUrl: 'book/finalize.html',
            controller: 'FinalizeController'
        });
    }])

    .controller('FinalizeController',
function($scope,$http,reservationData,$location) {

        $scope.start_dt;
        $scope.end_dt;
        $scope.room_info;
        $scope.occupancy;
        $scope.reservation_info;
        $scope.reser_done;
        $scope.customer={};

        $scope.init = function(){


            $scope.start_dt = reservationData.getValue('start_dt');
            $scope.end_dt = reservationData.getValue('end_dt');
            $scope.occupancy=reservationData.getValue('occupancy');;
            $scope.room_info=reservationData.getValue('room_info');

            if($scope.room_info==null){
                 $location.path('/book');
            }
        };

        $scope.init();

        $scope.book = function(){

            var reservationInfo;
            console.log($scope.customer);

            reservationInfo = {
```

```
                'customer':$scope.customer,
                'room_info':$scope.room_info,
                'start_dt':$scope.start_dt,
                'end_dt':$scope.end_dt,
                'occupancy':$scope.occupancy
            };

            $http.post('/api/createreservation',reservationInfo).success(function
(data){

                $scope.reser_done=true;
                $scope.reservation_info = data;


            });
        }
    }
);
```

Basically in the controller there is two function. The first is the **init** function when we retrieve all the information stored in the **hashtable** service. There is also a check about the **room_info;** this is needed so if a user access this page without selecting the room and if we don't have all the information he will get redirected to the home.

The second function is the most important. It's the book function, when we collect all the information about the reservation and the customer push the Book button the AngularJS application make a **REST** call to the **Laravel backend** asking to create the reservation. In the callback function we set the **reser_done** variable to hide the form and show the recap we talked some line before.

The last part needed is the **Laravel Reservation controller**. Create it with artisan

```
php artisan make:controller ReservationController
```

and paste the code below in it

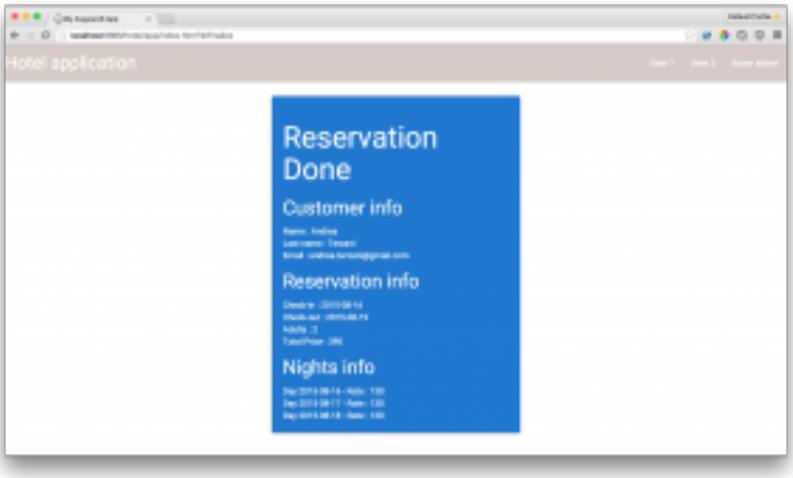
```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

use App\Http\Requests;
use App\Http\Controllers\Controller;
use App\Reservation;
use App\RoomCalendar;
use App\ReservationNight;
use App\Customer;
use Carbon\Carbon;

class ReservationController extends Controller
{
    public function createReservation(Request $request){

        $room_info  =$request['room_info'];
```

```
        $start_dt =Carbon::createFromFormat('d-m-Y', $request['start_dt'])-
>toDateString();
        $end_dt= Carbon::createFromFormat('d-m-Y', $request['end_dt'])-
>toDateString();

        $customer = Customer::firstOrCreate($request['customer']);

        $reservation  = Reservation::create();
        $reservation->total_price=$room_info['total_price'];
        $reservation->occupancy=$request['occupancy'];
        $reservation->customer_id=$customer->id;
        $reservation->checkin=$start_dt;
        $reservation->checkout=$end_dt;

        $reservation->save();

        $date=$start_dt;

        while (strtotime($date) < strtotime($end_dt)) {

            $room_calendar = RoomCalendar::where('day','=',$date)
                ->where('room_type_id','=',$room_info['id'])->first();

            $night =  ReservationNight::create();
            $night->day=$date;

            $night->rate=$room_calendar->rate;
            $night->room_type_id=$room_info['id'];
            $night->reservation_id=$reservation->id;

            $room_calendar->availability--;
            $room_calendar->reservations++;

            $room_calendar->save();
            $night->save();

            $date = date ("Y-m-d", strtotime("+1 day", strtotime($date)));

        }

        $nights = $reservation->nights;
        $customer = $reservation->customer;

        return $reservation;

    }
}
```

In the controller we have just one function. The **createReservation()** in it first part gather the request information then it use this to create a new **Reservation** record. Once the Reservation row is saved in the database and we have a valid **id,** we loop trough the dates, from **check-in** to **check-out,** and for each day we make two things. First we get the information about the rate of the room from the **RoomCalendar** table and we store it in the **ReservationNight** setting also a reference to the previously created **Reservation** and the **RoomType,** then we update the **RoomCalendar** row adding one to the reservations fields and removing one from the availability.

When the loop trough the date is complete we return the full **Reservation** object complete with the customer info and the nights array. The last thing in order to make the app working is to add the needed routes to the **routes.php** file. Below the complete file at this point :

```
<?php

Route::group(['prefix' => 'api'], function() {
    Route::post('searchavailabity', 'RoomCalendarController@searchAvailability');
    Route::post('createreservation', 'ReservationController@createReservation');
});

Route::group(['prefix' => 'adminapi'], function()
{
    Route::resource('room_type', 'RoomTypeController');
    Route::post('setpriceinrange',
'RoomCalendarController@setPriceInRangeForRoomType');


});
```



Payment

Laravel itself come with a module to directly integrate with Stripe, it's name is **Cashier**. Stripe is an online credit card payment system where, after a simple registration, you can start receiving payment via credit card using some exposed API and pay a little commission to it.

The good about this system is that all the information relative to the customer credit card will never touch your servers. This is very helpful for privacy and security and give you the possibility to avoid heavy code implementation to secure data relative to the payment process.

The frontend is made with **AngularJS** so we need a way to interact with **Stripe** from the **Angular** frontend and make our backend only ask for the charge using a provided token. **Token** in stripe represent the association with a credit card and a customer where you can charge for money.

To use **Stripe with Angular** there is more then one module. In this tutorial we will use **angular-payments**

For the demo code, angular-payments module is installed, but you can manully install it by moving to the public folder of the project download the module using bower :
`bower install angular-payments`

Now that the module is installed in our application we need to load it in the `index.html` file. Add the lines below in the `index.html` file :

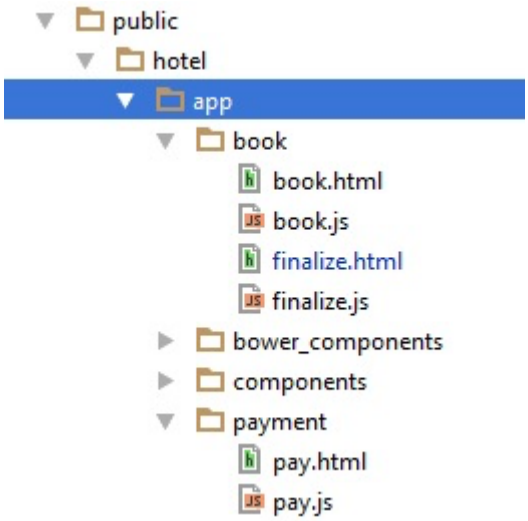
```
<script type="text/javascript" src="https://js.stripe.com/v2/"></script>

<script type="text/javascript">
  Stripe.setPublishableKey('pk_test_p0QQDCeJQA7W9X9y0VP7QOz5');
</script>

<script src="bower_components/angular-payments/lib/angular-payments.js"></script>
```

We loaded the module and also the Stripe.com library also initializing it passing the public key provided by it dashboard.

Another modification to the `index.html` is about loading the **AngularJS** module we are going to create, so other another line is needed :

```
<script src="payment/pay.js"></script>
```

In the app folder of the **AngularJS** application we can create a new folder payment and inside it 2 new files `pay.js` and `pay.html` like show in the picture.



In the `pay.html` file we are going to code the form needed to collect all the information about the credit card and a row with the response message hidden by ng-if

```
<div class="row" ng-if="!paid">
    <div class="col s12 m4 offset-m4">
        <div class="card-panel">
            <form stripe-form="handleStripe" name="myForm">
                <div class="row">
                    <div class="input-field col s12">
                        <label for="name">Name on card </label>
                        <input type="text" id="name">
                    </div></div>
                <div class="row">
                    <div class="input-field col s12">
                        <label for="cardn">Card number</label>
```

```
                        <input type="text" name="number" id="cardn"  ng-
model="number" payments-validate="card" payments-format="card" payments-type-
model="type" ng-class="myForm.number.$card.type"/>
                    </div>
                </div>
                <div class="row">
                    <div class="input-field col s6">
                        <label for="exp">Expiry</label>
                        <input type="text" id="exp" ng-model="expiry" payments-
validate="expiry" payments-format="expiry" />
                    </div>
                    <div class="input-field col s6">
                        <label for="cvc">CVC</label>
                        <input type="text" id="cvc" ng-model="cvc" payments-
validate="cvc" payments-format="cvc" payments-type-model="type"/>
                    </div>
                </div>
                <div class="row">
                    <div class="input-field col s12">
                        <button ng-if='loaded==true && paid==false' type="submit"
class="btn btn-primary btn-large">Pay {{reservation_info.total_price}}</button>
                    </div>
                </div>
            </form>
        </div>
    </div>
</div>
<div class="row"  ng-if="paid">
    <div class="col s12 m4 offset-m4">
        <div class="card-panel">
    <h2><p>{{message}}</p></h2>
        </div>
    </div>
</div>
```

As you can see the form is not a standard one. We used an attribute at the begin `stripe-form` where we specified also the function to callback once Stripe.com reply to us with the token. Also some fields like the number of the credit card and the expire date have some useful attribute for validation and format used for the input. All this custom attributes come from the **angular-payments** module.

Last thing to notice on the form is about the pay button. We used a `ng-if` directive to hide it before the reservation information is loaded. We will see better this once the controller code is implemented.

The other file `pay.js` content is shown below:

```
'use strict';

angular.module('myApp.pay', ['ngRoute','ui.materialize'])

    .config(['$routeProvider', function($routeProvider) {
        $routeProvider.when('/pay/:reserv_id', {
            templateUrl: 'payment/pay.html',
            controller: 'PayController'
        });
    }])

    .controller('PayController',
function($scope,$http,reservationData,$location,$routeParams) {

        $scope.res_id = $routeParams.reserv_id;
        $scope.paid = false;
```

```
        $scope.handleStripe = function(status, response){
            if(response.error) {
                $scope.paid= false;
                $scope.message = "Error from Stripe.com"
            } else {
                var $payInfo = {
                    'token' : response.id,
                    'customer_id' : $scope.reservation_info.customer_id,
                    'total':$scope.reservation_info.total_price
                };

                $http.post('/api/payreservation',
$payInfo).success(function(data){
                    if(data.status=="OK"){
                        $scope.paid= true;
                        $scope.message = data.message;
                    }else{
                        $scope.paid= false;
                        $scope.message = data.message;
                    }
                });

            }
        };

        $scope.init = function(){
            $scope.loaded = false;

            $http.get('/api/reservation/'+$scope.res_id).success(function(data){
                $scope.reservation_info = data;
                $scope.loaded=true;
            });
        };

        $scope.init();

    }
);
```

The first function executed on the controller is the init() function where all the information about the reservation to be paid is loaded by calling the **Laravel backend API**.

The other, and most important, function is the callback function handleStripe called after a response from Stripe is received. If the response is an error we set the message according, otherwise, using the `id` in the response and some other information taken from the reservation_info we call the **Laravel API** to proceed with the payment.

To access the payment pages, we use a button added on the reservation confirmation page `(finalize.html)`

```
<div class="row">
    <div class="col s12 m4 offset-m4" ng-if="reser_done">
        <div class="card blue darken-2">
            <div class="card-content white-text">
                <span class="card-title"> <h2>Reservation</h2></span>
                <h4>Customer info</h4>
                <p>Name : {{reservation_info.customer.first_name}}</p>
                <p>Last name : {{reservation_info.customer.last_name}}</p>
                <p>Email : {{reservation_info.customer.email}}</p>
                <h4>Reservation info</h4>
```

```
                <p>Check-in : {{reservation_info.checkin}}</p>
                <p>Check-out : {{reservation_info.checkout}}</p>
                <p>Adults : {{reservation_info.occupancy}}</p>
                <p>Total Price : {{reservation_info.total_price}}</p>
                <h4>Nights info</h4>
                <p ng-repeat="night in reservation_info.nights">
                    Day {{night.day}}  -  Rate : {{night.rate}}
                </p>
                <a href="#/pay/{{reservation_info.id}}" style="text-align:
center"><button class="btn">Pay ${{reservation_info.total_price}}</button></a>
            </div>
        </div>
    </div>
</div>
```

That's all about the **Angular** side. We received a **token** from Stripe and we can proceed to charge the customer for the reservation amount via the Laravel backend.

Laravel and Stripe

First of all we need to setup **Cashier** in **Laravel** starting by adding it using composer. In the `composer.json` add :

```
"require-dev": {
        "fzaninotto/faker": "~1.4",
        "mockery/mockery": "0.9.*",
        "phpunit/phpunit": "~4.0",
        "phpspec/phpspec": "~2.1",
        "laravel/cashier": "~5.0"
    },
```

and run `composer update` from the command line.

After the module is downloaded we have to set the API keys provided by **Stripe** in the**service.php** file adding this lines :

```
'stripe' => [
        'model'  => App\Customer::class,
        'key'    => 'pk_test_p0XXXXXXXXXXX0VP7QOz5',
        'secret' => 'sk_test_vkLyXXXXXXXXXa2r90up',
    ],
```

Now that **Laravel** is ready to use **Cashier** and **Stripe** we have to modify existing application to add some **API** needed and load some implementation on the **Customer model.**

Change the previously created model file **Customer.php** pasting the code below

```
<?php

namespace App;
use Laravel\Cashier\Billable;
use Laravel\Cashier\Contracts\Billable as BillableContract;

use Illuminate\Database\Eloquent\Model;
```

```
class Customer extends Model implements BillableContract
{
    use Billable;

    protected $fillable = ['first_name', 'last_name','email'];

}
```

As you can see from the code we included the **Cachier** library in the file and made our class implementing **BillableContract**.

The other file we need to modify is the **ReservationController**. In the **AngularJS** side we called the **REST API** to retrieve all the information about the reservation but we never implemented it. Will be enough to add the line below to the controller to provide this functionality

```
public function show($id){
        return  Reservation::find($id);
    }
```

In the `routes.php` in the api group add the line below to make the function accessible

```
Route::get('reservation/{id}', 'ReservationController@show');
```

To maintain separation by concept, as we made till now in this tutorial we need a new controller where all the payments relative API will be coded. From the command line using `artisan create` then a **PaymentController**.

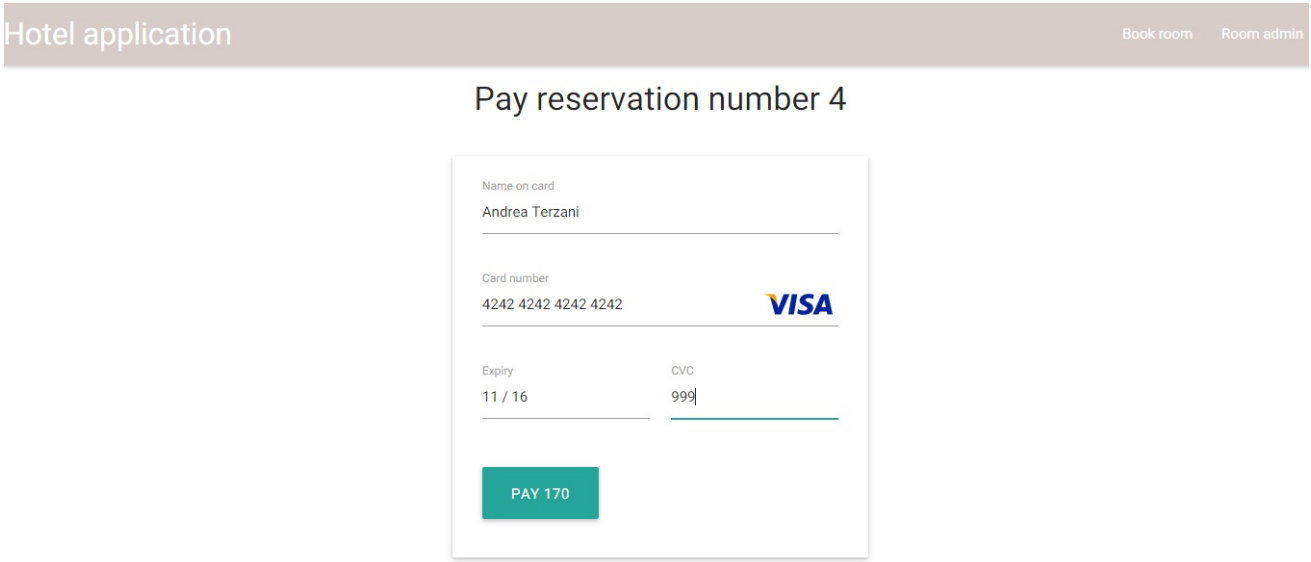
```
php artisan make:controller PaymentController
```

The code for the controller is show below

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

use App\Http\Requests;
use App\Http\Controllers\Controller;
use App\Customer;

class PaymentController extends Controller
{

    public function pay(Request $request){

        $token  = $request['token'];
        $cutomer_id  =$request['customer_id'];
        $total_price  =$request['total']*100;

        $customer = Customer::find($cutomer_id);

        if($customer->charge($total_price,
            [
            'source' => $token,
             'receipt_email' => $customer->email
            ]))
```

```
        {
            $mess = ['status' => "OK","message" =>"Payment ok"];
            return $mess;
        }else{
            $mess = ['status' => "ERROR","message" =>"Error submitting payment"];
            return $mess;
        }
    }

}
```

The controller code is pretty easy to understand. There is only one function to process the payment. In the first part of it we retrieve all the information from the request. The total price of the reservation is multiplied by 100 because **Stripe** want the amount in cents.



After all the information are retrieved the Customer who made the reservation is retrieved from the database and the charge method is called on it passing the token acquired by Angular and the customer email. The charge() method will return true if the payment is ok and false if something where wrong. The response of the function in both case will be an HTTP 200 but the message and status is different.

Have fun with your new website

YJ@2015