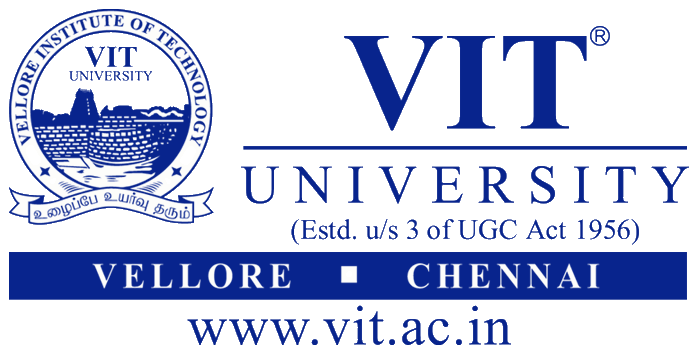
****

**Project Review  
Design Patterns**

**Consumer Service Website for Online Party Hall Booking**

Under Guidance of

Prof. Sridhar R (SCSE)

By - Kunwarjeet Singh 12MSE1086

**Abstract**

We are going to develop a consumer service website where people can book halls for various type of parties. Currently there are no available websites in the market for this kind of service. Business owners can advertise with us based on membership subscription and consumers can search for a hall , review them , book them for a party. We will also develop a review system for this where users can review, rate, share a hall and rankings will be given to halls based on these. We will be using using various Design Patterns. During Final Review we will show you live demo of website and use of design patterns in our project.

**Design Patterns to be used**

* **Creational Pattern** – Factory, Builder, Prototype, Singleton
* **Structural Patterns** – Façade, Adapter, Decorator, Proxy
* **Behavioural Patterns** – Iterator
* **Architectural Patterns** – MVC

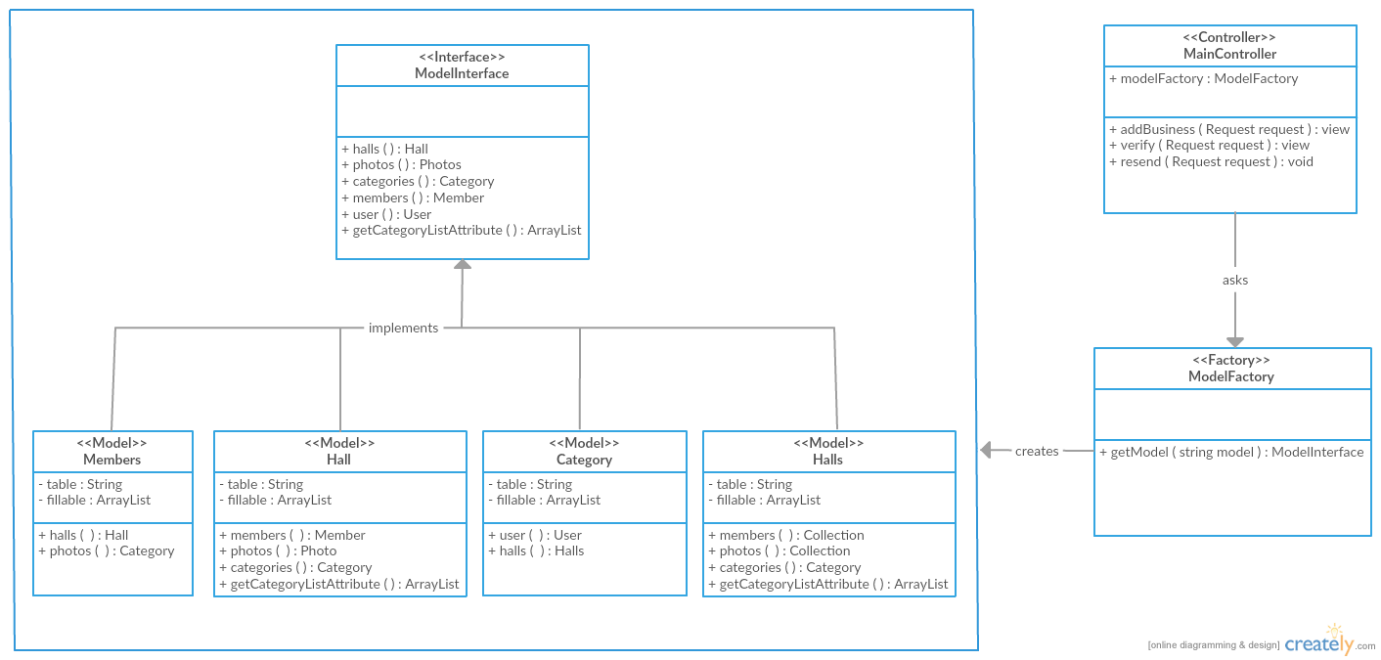
**Hardware/Software Used**

* PHP-Laravel Framework
* Front End – HTML5, CSS3, Bootstrap, JQuery, AJAX, Javascript
* Backend – Mysql, WAMP Server
* Notepad++ , Netbeans IDE

**Design Patterns Used**

1. **Factory Pattern**

One of the most used design pattern. Comes under creational Pattern. Here we create object without exposing creational logic. We have 4 Model Classes which implement ModelInterface and a ModelFactory which creates the object via ModelInterface. In our MainController we call ModelFactory by passing particular string to get object.



**1a Member Model Class**

class Member extends Model implements ModelInterface{

protected $table = 'members';

protected $fillable = ['first\_name', 'last\_name', 'email', 'street', 'suburb','city','state','country','pincode','password','status','hash\_key'];

public function halls()

{

return $this->hasMany('App\Hall', 'member\_id', 'id' );

}

public function photos()

{

return $this->hasMany('App\Photo', 'member\_id', 'id' );

}

}

**1b Hall Model Class**

class Hall extends Model implements Modelinterface{

protected $table = 'halls';

protected $fillable = ['name', 'member\_id', 'description', 'street', 'suburb','city','state','country','pincode','website','primary\_email','alternate\_email','primary\_contact','alternate\_contact','n\_reviews', 'n\_likes', 'rating', 'subscription','avatar','cover','banner','platinum\_cover', 'gold\_cover','silver\_cover'];

public function member()

{

return $this->belongsTo('App\Member', 'member\_id', 'id' );

}

public function photos()

{

return $this->hasMany('App\Photo', 'hall\_id', 'id' );

}

public function categories()

{

return $this->belongsToMany('App\Category');

}

public function getCategoryListAttribute()

{

return $this->categories()->lists('id');

}

}

**1c Category Model Class**

class Category extends Model implements ModelInterface{

protected $table = 'categories';

protected $fillable = ['categoryName', 'urlname', 'userid', 'metakeyword', 'metatitle','metadesc','hits'];

public function user()

{

return $this->belongsTo('App\User', 'userid', 'id' );

}

public function hall()

{

return $this->belongsToMany('App\Hall' )->withTimestamps();

}

}

**1d User Model Class**

class User extends Model implements AuthenticatableContract, CanResetPasswordContract, ModelInterface

{

use Authenticatable, CanResetPassword;

/\*\*

\* The database table used by the model.

\*

\* @var string

\*/

protected $table = 'users';

/\*\*

\* The attributes that are mass assignable.

\*

\* @var array

\*/

protected $fillable = ['name', 'username', 'email', 'password','avatar','provider','provider\_id'];

/\*\*

\* The attributes excluded from the model's JSON form.

\*

\* @var array

\*/

protected $hidden = ['password', 'remember\_token'];

public function reviews()

{

return $this->hasMany('App\Review', 'userid', 'id');

}

public function tags()

{

return $this->hasMany('App\Tag', 'userid', 'id' );

}

public function isAnAdmin()

{

if ($this->usertype == 0)

{

return true;

}

}

}

**1e ModelFactory Class**

class ModelFactory {

public function getModel($modelType){

if($modelType == null)

return null;

if(strcasecmp($modelType,"Hall"))

return new Hall;

else if(strcasecmp($modelType,"User"))

return new User;

else if(strcasecmp($modelType,"Member"))

return new Member;

else if(strcasecmp($modelType,"Category"))

return new Category;

return null;

}

}

**1f ModelInterface**

interface ModelInterface

{

public function reviews(){}

public function tags(){}

public function isAnAdmin(){}

public function halls(){}

public function photos(){}

public function member(){}

public function photos(){}

public function categories(){}

public function getCategoryListAttribute(){}

public function user(){}

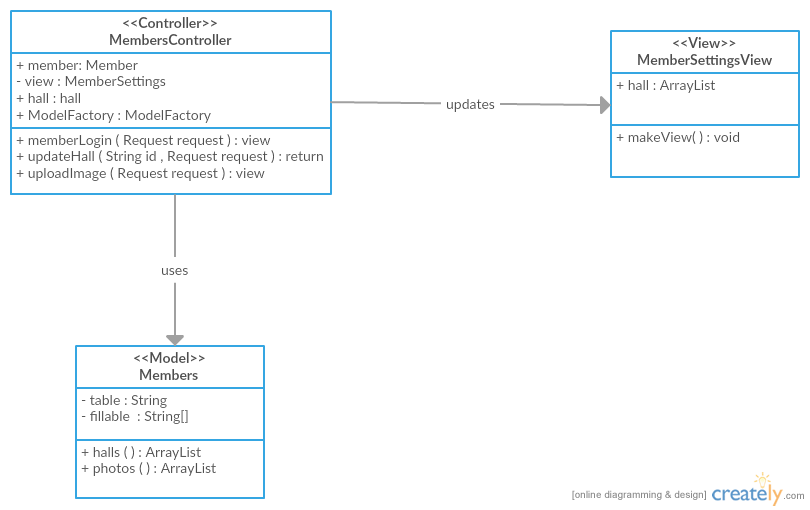
public function hall(){}

}

1. **MVC Pattern**

Model View Controller Design Pattern comes under Architectural Pattern.

* **Model** - Model represents an object or PHP Object carrying data. It can also have logic to update controller if its data changes.
* **View** - View represents the visualization of the data that model contains.
* **Controlle**r - Controller acts on both model and view. It controls the data flow into model object and updates the view whenever data changes. It keeps view and model separate.



**2a Members Controller**

public function memberLogin(\Illuminate\Http\Request $request){

$member=Member::where('email',$request->email)->where('password',$request->password)->get();

if(!$member->isEmpty())

{ $hall= Hall::where('member\_id',$member[0]->id)->get();

$hall=$hall[0];

return Redirect( 'members/settings' )

->with( 'hall', $hall );

}

return "Login Failed";

}

public function updateHall($id , \Illuminate\Http\Request $request){

$hall= Hall::findOrFail($id);

$hall->update($request->all());

return Redirect( 'members/settings' )

->with( 'hall', $hall );

}

public function uploadImage(\Illuminate\Http\Request $request){

if (Request::ajax()){

$fileName = $\_FILES["file"]["name"];

$kaboom = explode(".", $fileName);

$ext = end($kaboom);

$target\_path = 'images/halls/hall\_' .$request->id.'\_'.$request->type.'.'.$ext;

if(move\_uploaded\_file($\_FILES['file']['tmp\_name'], $target\_path)) {

// echo "The file ". basename( $\_FILES['file']['name']).

" has been uploaded";

} else{

// echo "There was an error uploading the file, please try again!";

}

Hall::where('id',$request->id)->update([$request->type=>'hall\_' .$request->id.'\_'.$request->type.'.'.$ext]);

$resized\_file = 'images/halls/hall\_' .$request->id.'\_'.$request->type.'.'.$ext;

if($request->type=='avatar'){

$w=250; $h=250;}

else if($request->type=='cover'){

$w=1200; $h=400;}

else if($request->type=='platinum\_cover'){

$w=600; $h=200;}

else if($request->type=='gold\_cover'){

$w=150; $h=200;}

else if($request->type=='silver\_cover'){

$w=150; $h=100;}

else if($request->type=='banner'){

$w=300; $h=100;}

list($w\_orig, $h\_orig) = getimagesize($target\_path);

$img = "";

$ext = strtolower($ext);

if ($ext == "gif"){

$img = imagecreatefromgif($target\_path);

} else if($ext =="png"){

$img = imagecreatefrompng($target\_path);

} else {

$img = imagecreatefromjpeg($target\_path);

}

$tci = imagecreatetruecolor($w, $h);

// imagecopyresampled(dst\_img, src\_img, dst\_x, dst\_y, src\_x, src\_y, dst\_w, dst\_h, src\_w, src\_h)

imagecopyresampled($tci, $img, 0, 0, 0, 0, $w, $h, $w\_orig, $h\_orig);

imagejpeg($tci, $resized\_file, 80);

echo url('/images/halls/hall\_'.$request->id.'\_'.$request->type.'.'.$ext);

}

}

**2b Members Model**

class Member extends Model {

protected $table = 'members';

protected $fillable = ['first\_name', 'last\_name', 'email', 'street', 'suburb','city','state','country','pincode','password','status','hash\_key'];

public function halls(){

return $this->hasMany('App\Hall', 'member\_id', 'id' );

}

public function photos()

{

return $this->hasMany('App\Photo', 'member\_id', 'id' );

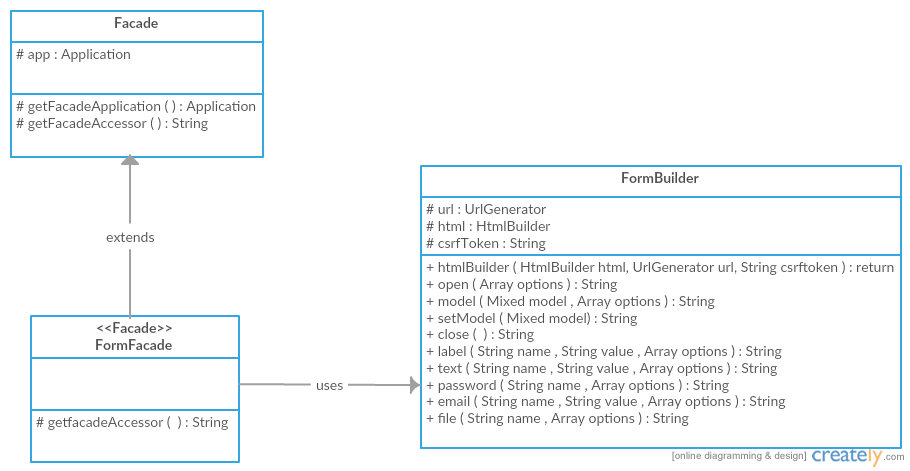
}

}

1. **Facade Pattern**

Facade pattern hides the complexities of the system and provides an interface to the client using which the client can access the system. This type of design pattern comes under structural pattern as this pattern adds an interface to existing system to hide its complexities.

This pattern involves a single class which provides simplified methods required by client and delegates calls to methods of existing system classes



**3a FormFacade**

class FormFacade extends Facade {

/\*\*

\* Get the registered name of the component.

\*

\* @return string

\*/

protected static function getFacadeAccessor() { return 'form'; }

}

**3b FormBuilder**

class FormBuilder {

use Macroable;

protected $html;

protected $url;

protected $csrfToken;

public function \_\_construct(HtmlBuilder $html, UrlGenerator $url, $csrfToken)

{

$this->url = $url;

$this->html = $html;

$this->csrfToken = $csrfToken;

}

public function open(array $options = array())

{

$method = array\_get($options, 'method', 'post');

$attributes['method'] = $this->getMethod($method);

$attributes['action'] = $this->getAction($options);

$attributes['accept-charset'] = 'UTF-8';

$append = $this->getAppendage($method);

if (isset($options['files']) && $options['files'])

{

$options['enctype'] = 'multipart/form-data';

}

$attributes = array\_merge(

$attributes, array\_except($options, $this->reserved)

);

$attributes = $this->html->attributes($attributes);

return '<form'.$attributes.'>'.$append;

}

public function model($model, array $options = array())

{

$this->model = $model;

return $this->open($options);

}

public function setModel($model)

{

$this->model = $model;

}

public function close()

{

$this->labels = array();

$this->model = null;

return '</form>';

}

public function email($name, $value = null, $options = array())

{

return $this->input('email', $name, $value, $options);

}

public function file($name, $options = array())

{

return $this->input('file', $name, null, $options);

}

}

1. **Adapter Pattern**

Adapter pattern works as a bridge between two incompatible interfaces. This type of design pattern comes under structural pattern as this pattern combines the capability of two independent interfaces.

In our project user uploads different types of images with different sizes and also there are around 8-10 types of images for a member to upload, so we can put them in separate classes and can use Adapter Pattern for easy integration of new types.

1. **Decorator Pattern**

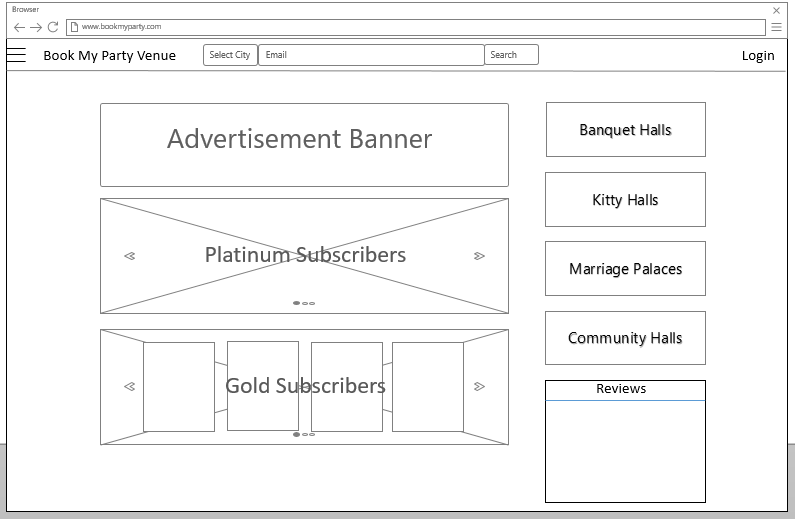
Decorator pattern allows a user to add new functionality to an existing object without altering its structure. This type of design pattern comes under structural pattern as this pattern acts as a wrapper to existing class.

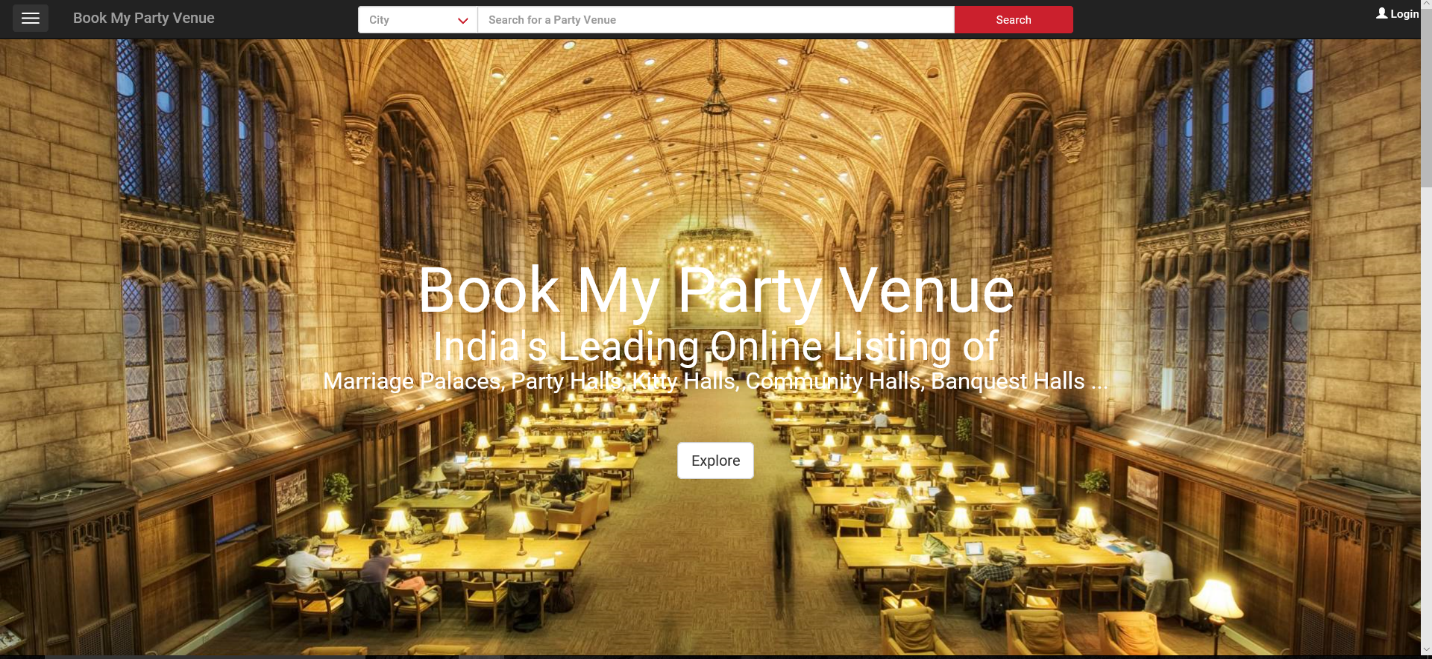
This pattern creates a decorator class which wraps the original class and provides additional functionality keeping class methods signature intact.

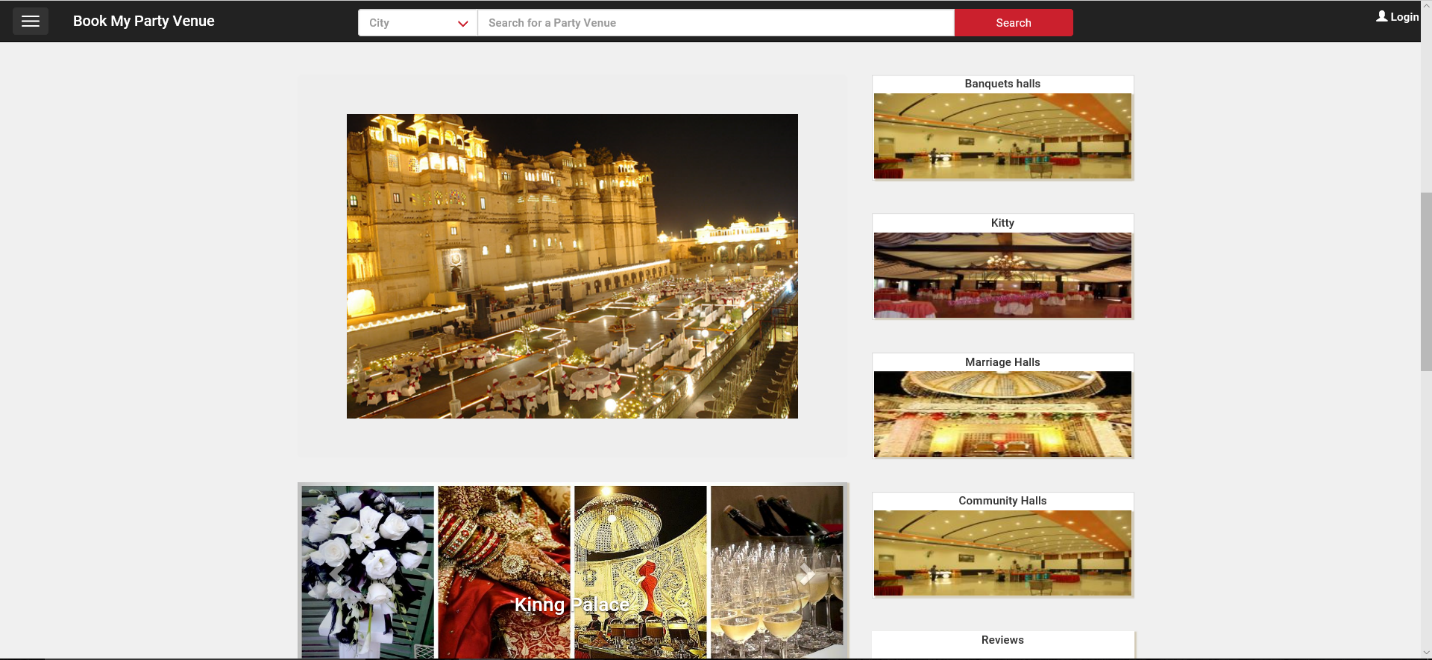
In our project we have search/filters, so each filter can be dynamically added via AJAX call and filtered results will be fetched. So it’s very easy to add new filter without changing internal logic again and again.

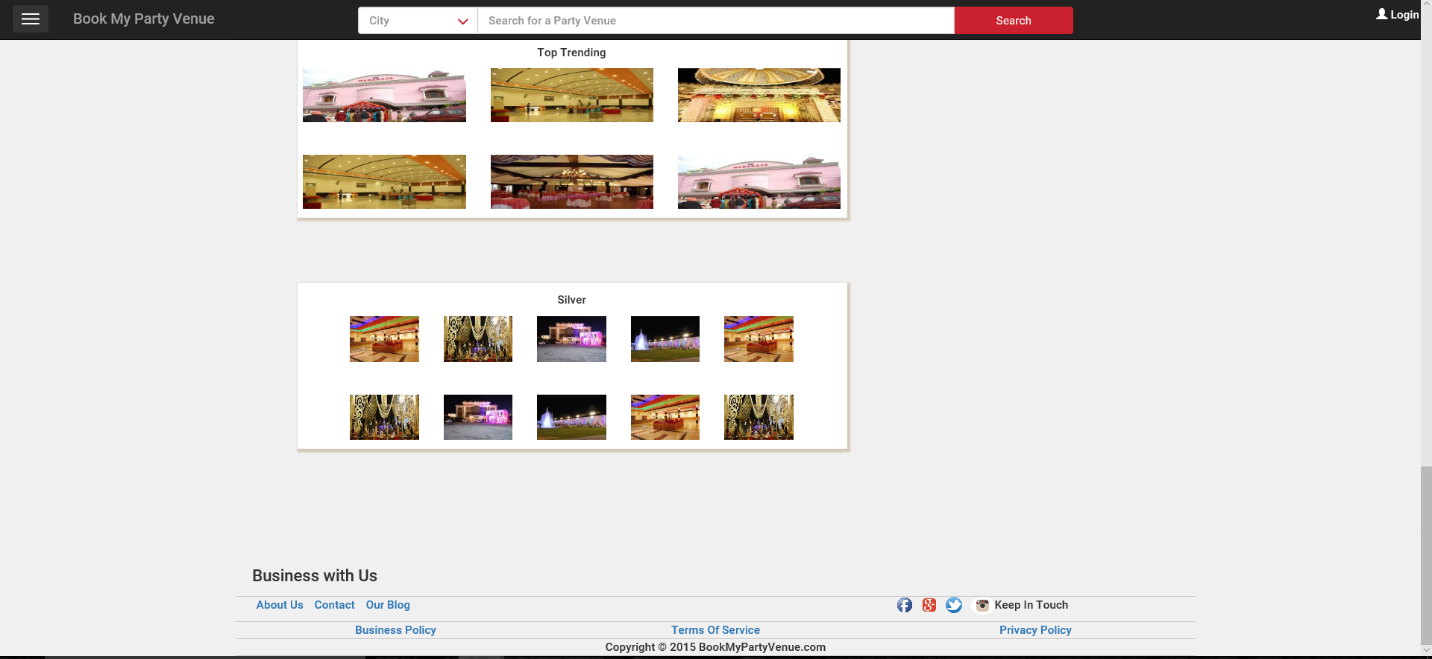
**Screenshots**

**Home Page Layout**

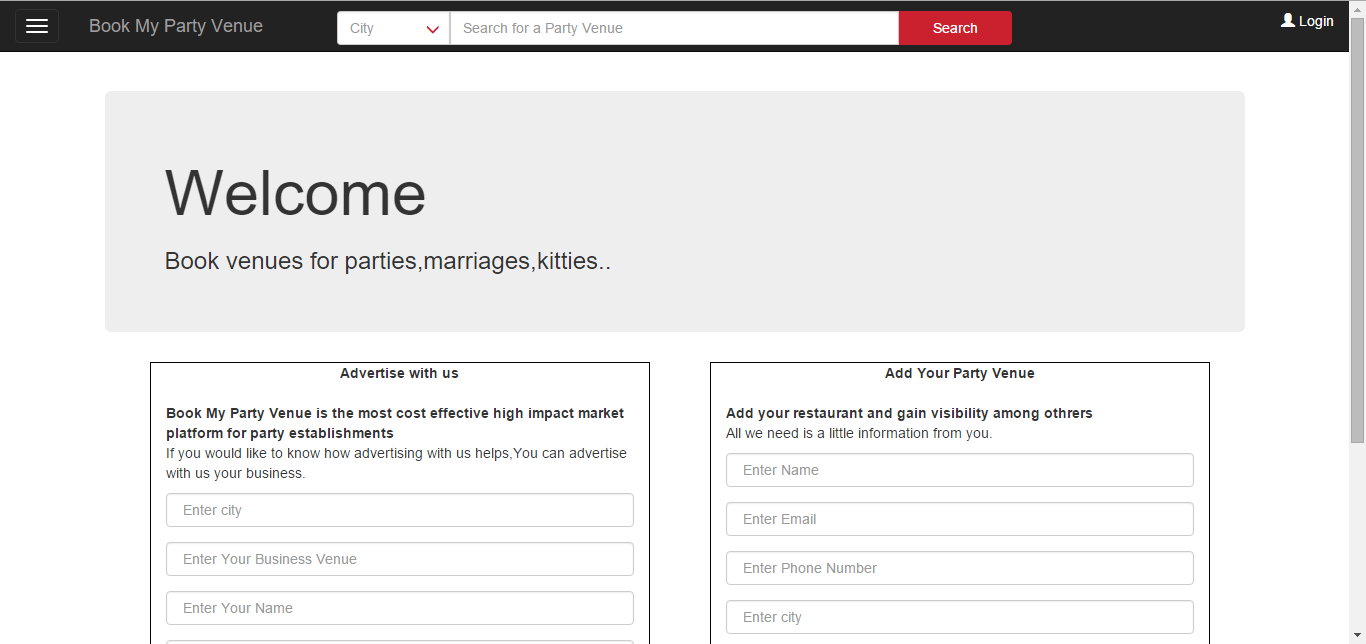


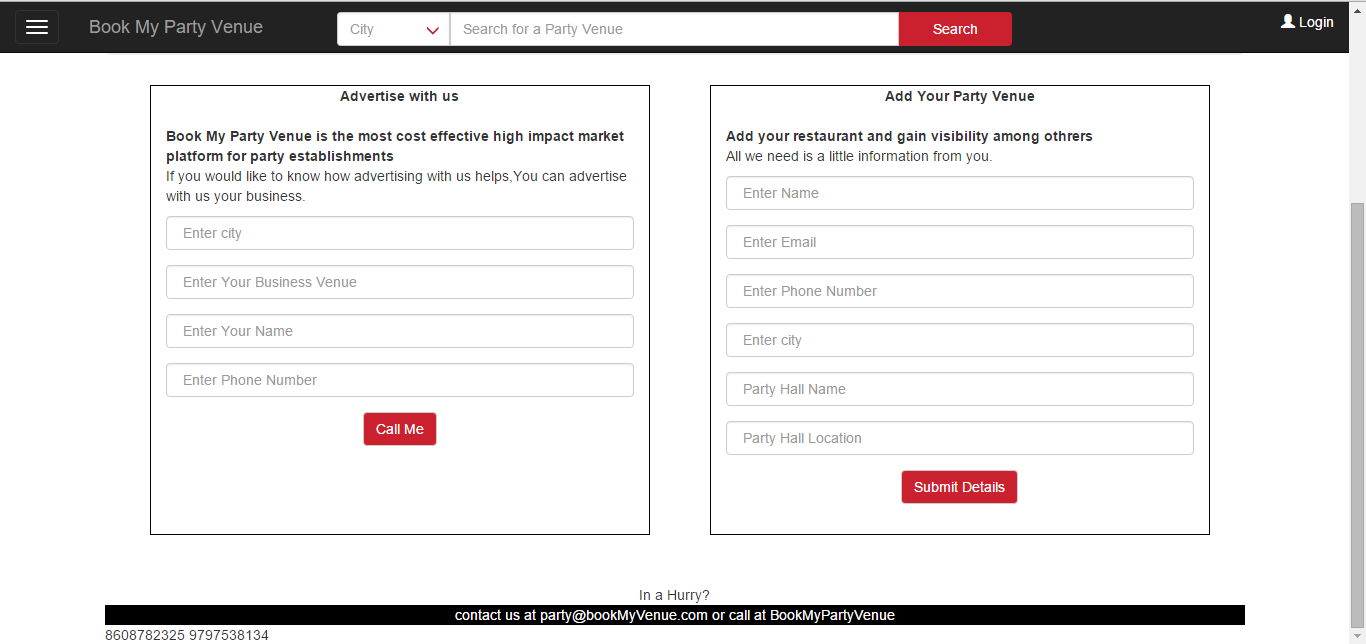
****

****

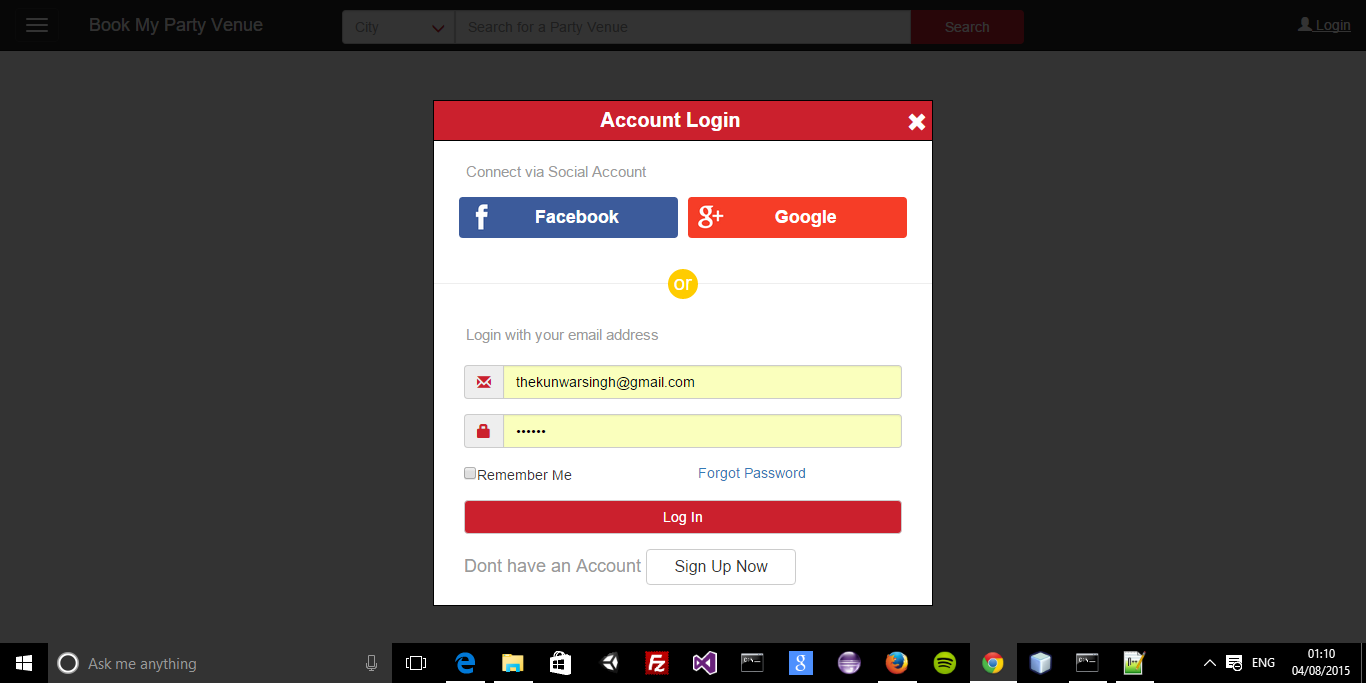
****

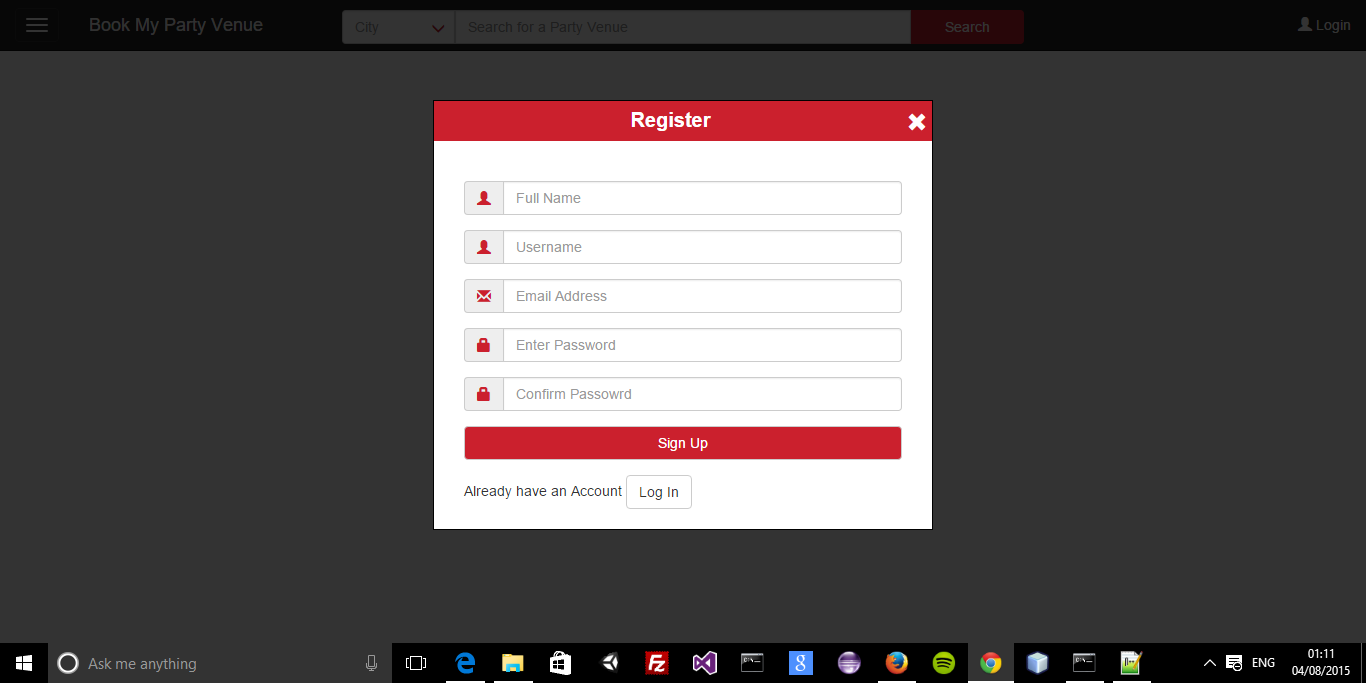
**Advertise/Business with us**

****

****

**Login/ Register**

****

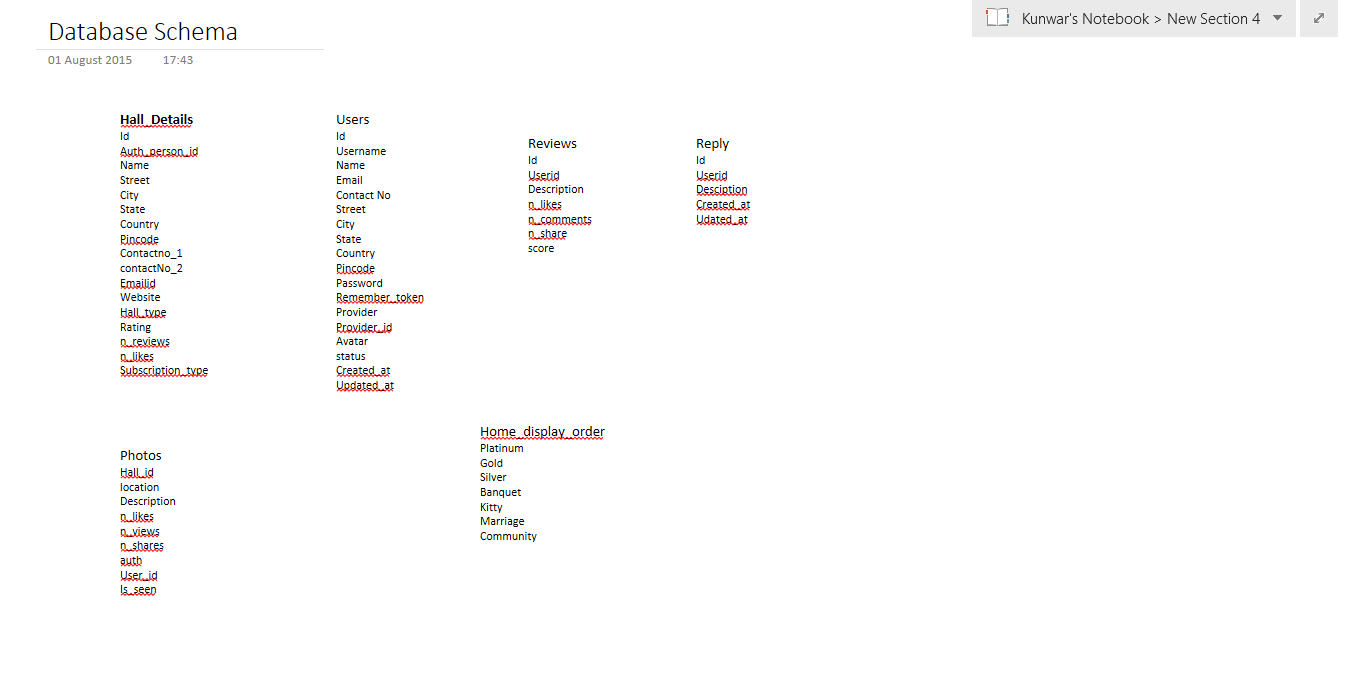
****

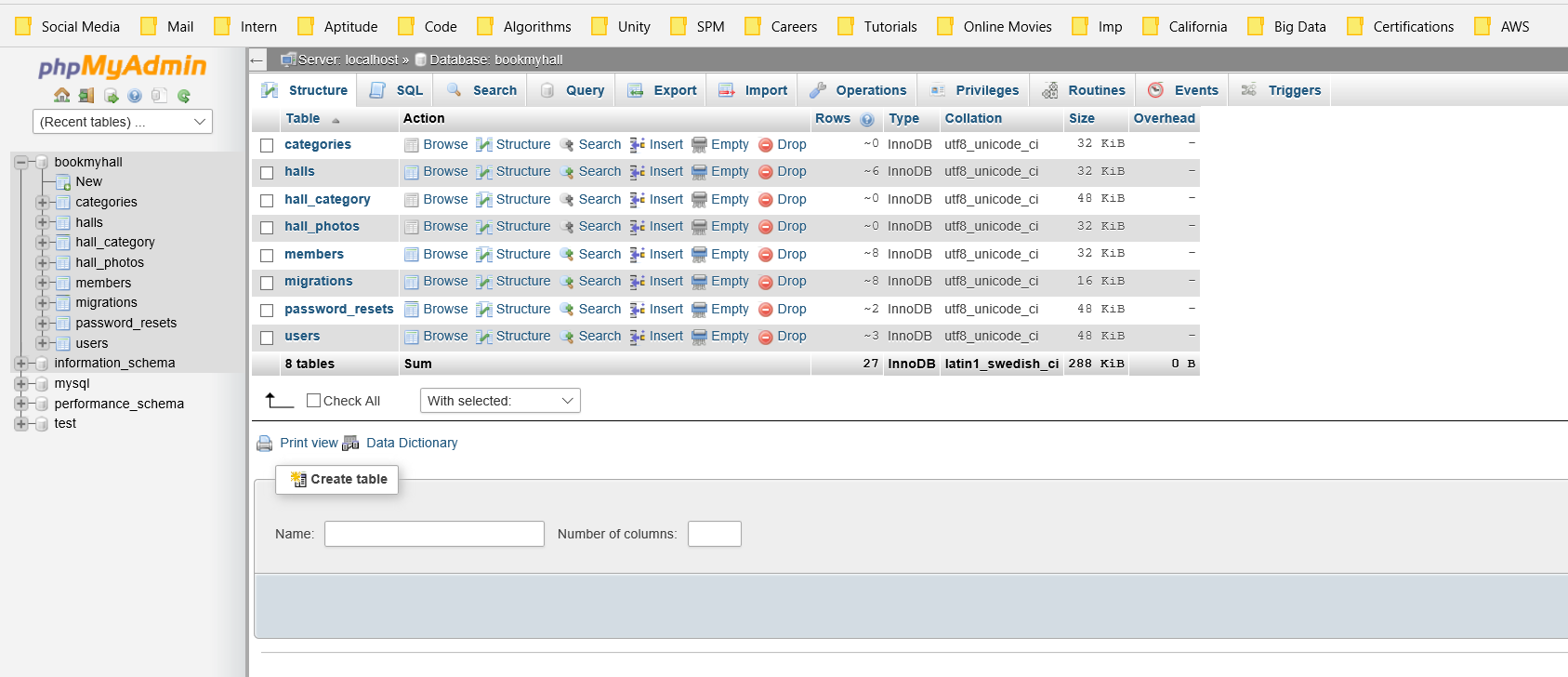
**Members verification**

****

****

Database Schema





Members Dashboard

