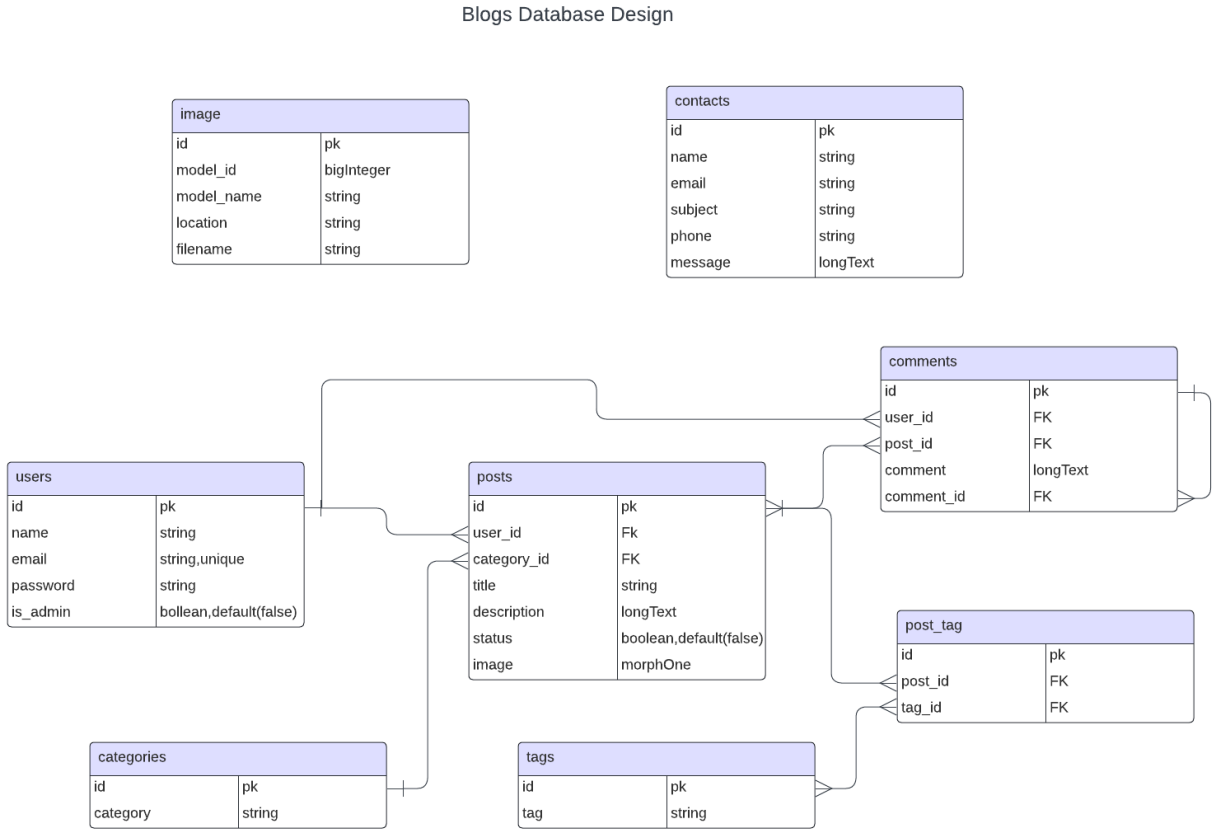
TOPIC TO COVER:

1. Database Design
2. CRUD, CRUD with AJAX, CRUD with API.
3. Database Design
4. One to One Relationship
5. One to Many Relationship
6. Many to Many Relationship
7. Contact Us & How to Send Mail.
8. Authentication and Middleware
9. Roles and Permission management using Saptie
10. Morph One and Morph Many
11. Laravel Module Package
12. Laravel Repository Design Pattern
13. Laravel Event and Listener
14. Pusher for Real Time Notifications
15. Livewire
16. Twilio
17. MailChimp
18. Laravel Package Development (Own Custom Package)
19. Use MongoDB(NoSQL) in Laravel
20. CronJob
21. Testing (Unit & Feature Testing)
22. Check User Status (Online or Offline)
23. Filters with AJAX

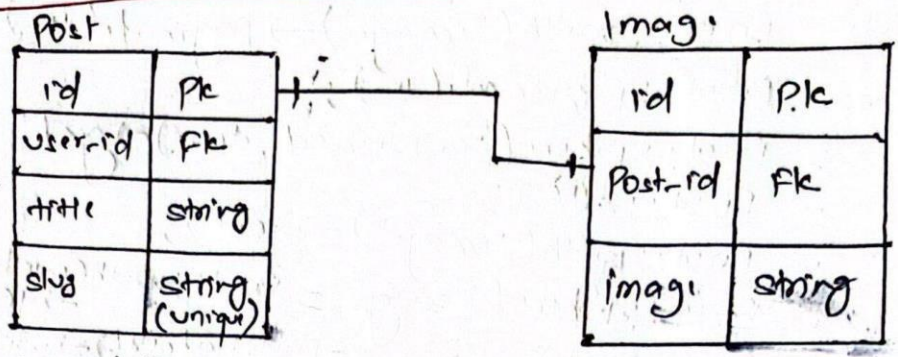
Database Design:




```php
// ====frontend=========
require __DIR__ . '/public.php';

// ====backend===========
Route::prefix('admin')->group(function () {
    require __DIR__ . '/admin.php';
});
```

(*) One to one Relationship:



- **Post model:**

```
public function image(){
    return $this->hasOne(Image::class);
}
protected $fillable = ['title', 'user_id', 'slug'];
```

- **Image model:**

```
protected $fillable = ['post_id', 'image'];
```

- **Post/Img controller:**

↳ **CREATE:**

```
public function addService($request){
DB::transaction(function () use ($request){   // roll back
    //add to parent:
    $post = Post::create($request);
            OR
    $post = Post::create([
        'user_id' => auth()->id();
        'title' => $request['title'],
        ...
    ]);
    //child
    Image::create([
        'image' => $request['image'],
        'post_id' => $post->id
    ]);
```

↳ READ:

```
public function fetchPost(){
    $posts = Post::with('image')->paginate(10);
           = Post::paginate(10);
           = Post::where('published', true)->get();
           = Post::all();
           = Post::with('image')->get();
           = Post::withCount('image')
                  ->get();
           = Post::latest()->with('image')
                  ->get();
```

↳ READ (single Post)

```
public function singlePost($blog){
    $post = Post::where('id', $blog->id)->first();
}
```

↳ DELETE:

```
public function delete($post){
    $post->delete();
        OR
    $post->image->delete();
    $post->delete();
}
```

↳ UPDATE:

```
public function updateService($request, $post){
    $post->update($request);
```

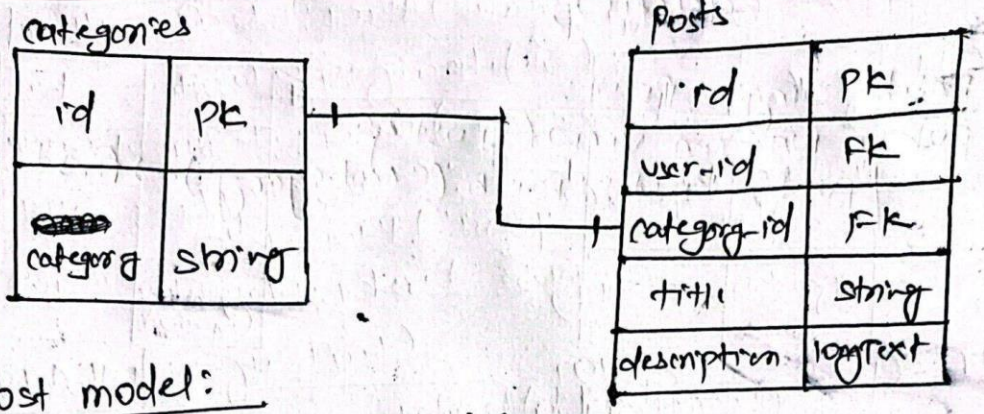
//parent:

```
    $post->update(['title' => $request['title'],
                  ]);
```

//child:

```
    $post->image->update([
        'image' => $request['image'],
    ]);
```

(*) One to One Inverse Relation:



- Post model:

```
public function category(){
    return $this->belongsTo(category::class,
                            'category_id');
}
protected $fillable = ['user_id', 'category_id',
                       'title', 'description'];
```

- category model:

```
protected $fillable = ['category'];
```

- READ:

```
public function fetchPost(){
    $posts = Post::latest()->with('category')->get();
}
```

- POST:

```
public function addservice($request){
    $post = Post::create([
        'user_id' => auth()->id,
        'category_id' => $request['category'],
        'title' => $request['title'],
    ]);
```

- UPDATE:

```
public function updateservice($request, $post){
    $post->update([ 'user_id' => auth()->id(),
                    'category_id' => $request['category']
    ]); }
```

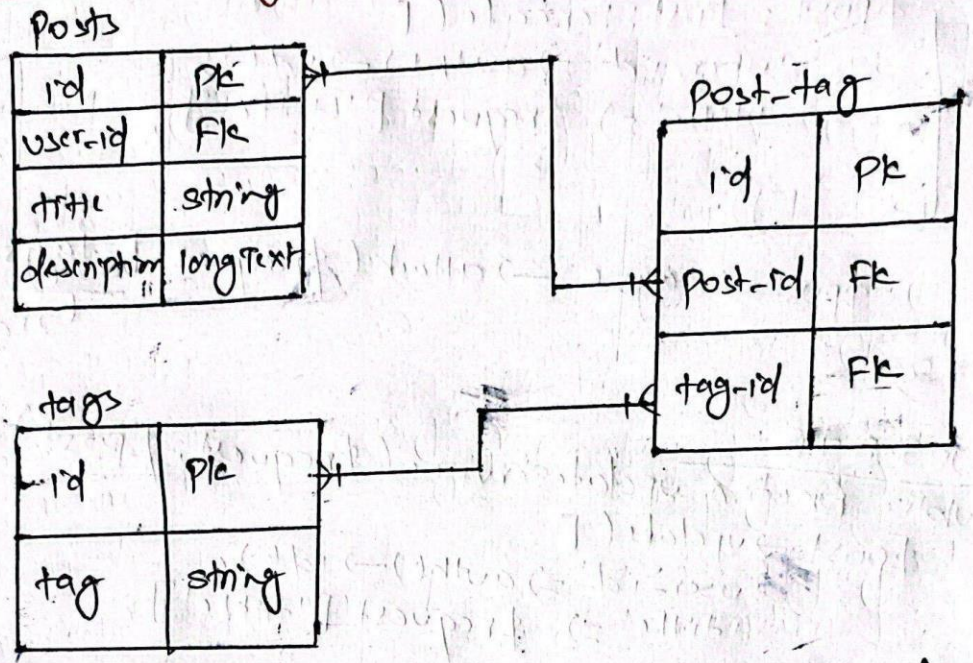
- DELETE:

```
public function deletePost($post){
    $post->delete();
}
```

- Fetch single post:

```
public function view($post){
    $post = Post::with('category')->first();
}
```

(*) Many to Many relationship:



- **Post model:**

```
public function tags(){
    return $this->belongsToMany(Tag::class);
}
protected $fillable = ['user_id', 'title', 'description'];
```

- **READ:**

```
public function fetchPost(){
    $posts = Post::latest()->with('tags')->get();
        OR
    $posts = Post::withCount('tags')->get();
        OR
    $posts = Post::with('tags')->get();
}
```

- **DELETE:**

```
public function delete($post){
    $post->tags()->detach(); → if no cascade on delete
    $post->delete();
}
```

- **CREATE:**

```
public function addService($request){
    $post = Post::create([
        'user_id' => auth()->id(),
        'title' => $request['title'],
        ...
    ]);
    $post->tags()->attach($request['tags']);
}
```

(`$request['tags']` → id)

- **UPDATE:**

```
public function updateService($request, $post){
    $post->update([
        'user_id' => auth()->id(),
        'title' => $request['title'],
        ...
    ]);
    $post->tags()->sync($request['tags']);
}
```

(`$request['tags']` → id)

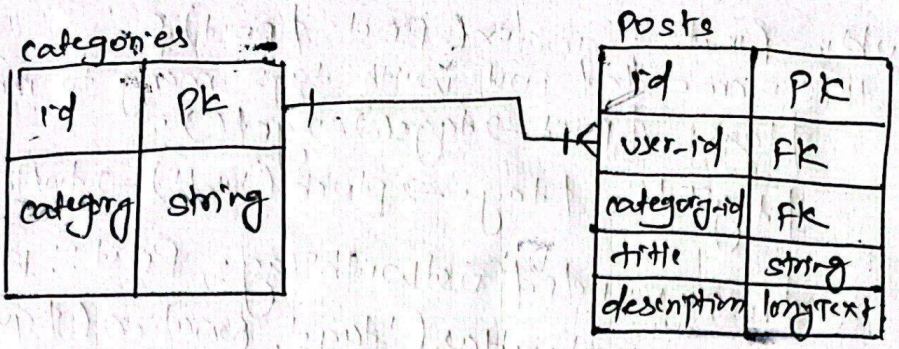
- **Single Post:**

```
public function view($post){
    $post = Post::with('tags')->first();
}
```

(*) Task :

(7)

(1)

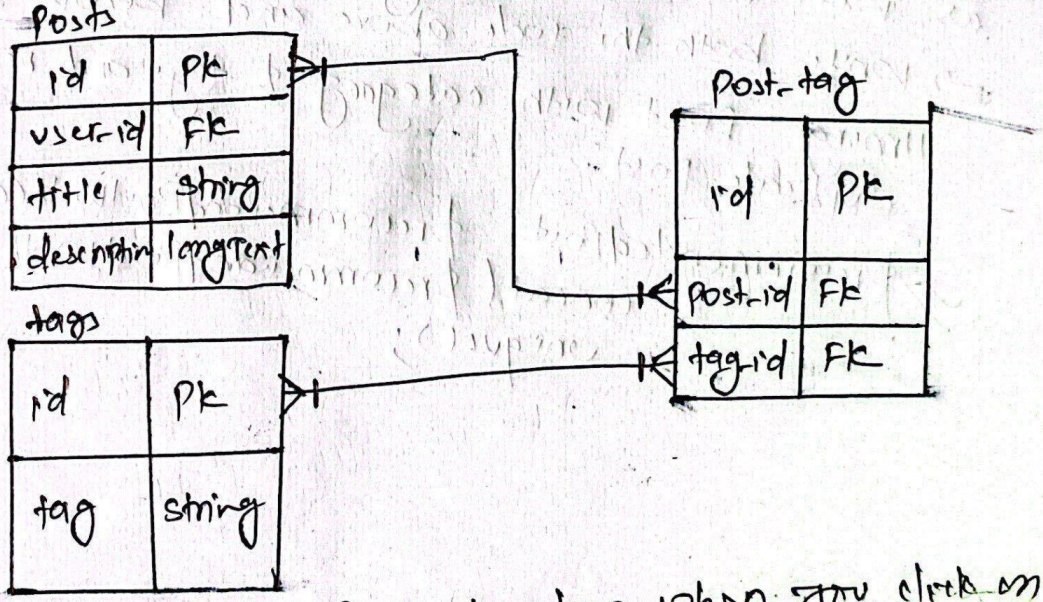


There is a list of posts. Now, when you click on individual post then view that post and also with all the similar posts as recommendation. all similar categories.

```
public function index(Post $post)
    //recommended post with category
    $recommendedPostWithCategory = Post::where(
            category_id, $post->category_id)->
            with('category')->get();
```

→ all the post with similar category will be return as collections in one to many relation.

(2)



There is a list of posts. Now when you click on individual post then view that post and also all the similar posts as recommendation. with all similar tags.

```
public function index(Post $post){
    // recommended post with tags many to many relations
    $tags = $post->tags()->get();
    $tagIds = $tags->pluck('id')->toArray();
    $recommendedPostWithTag = Post::with('tags')
        ->whereHas('tags', function($query) use
            ($tagIds){
            $query->whereIn('tag_id', $tagIds);
        })->get();
}
```

↳ All the Post with similar tags will be ~~return~~ as collection in many to many relation.

Task 3:

Get all the post with similar ~~post as~~ category as well as similar Tags.

→ write both the code above and below code:

```
// merge post with category & tags and remove the
    duplicate post.
$recommendedPost = $recommendedPostWithCategory->
    concat($recommendedPostWithTag)->
    unique();
```

Fetch all the data with Filter:

```php
// =========GET Products============
public function fetchService($request)
{
    $productQuery = Product::query()->latest();

    // filter
    if (!empty($request['keyword'])) {
        $productQuery->where('name', 'like', '%' . $request['keyword'] . '%')->orWhere('price', '=', $request['keyword'])
    }

    // pagination
    $perPage = $request['perPage'] ?? 10;
    $products = $productQuery->paginate($perPage);

    return $products;
}
```

Example of One to One Relationship:

```php
<?php

namespace App\Service;

use App\Models\Blog;
use App\Models\BlogInfo;
use Illuminate\Support\Facades\Storage;

// one to one relation
class BlogService
{

    // =============POST============
    public function addService($request)
    {
        // store single image in local storage folder
        if (isset($request['image'])) {
            $timestamp = now()->timestamp;
            $originalName = $request['image']->getClientOriginalName();
            $imageName = $timestamp . '-' . $originalName;
            $request['image']->storeAs('public/images/blogs', $imageName);

            // update the image name in the $request array
            $request['image'] = $imageName;
        };

        // store in database table
        // Blog::create($request);

        // =====one to one relation======
```

```
        // store in parent table
        $blog = Blog::create([
            'title' => $request['title'],
            'slug' => $request['slug'],
        ]);
        // store in child table
        BlogInfo::create([
            'image' => $request['image'],
            'description' => $request['description'],
            'blog_id' => $blog->id,
        ]);
    }

    // ========GET(Read all)================
    public function fetchBlogs()
    {
        //    $blogs=Blog::paginate(10);  //Paginate with 10 records per page
        //    $blogs=Blog::where('published',true)->get(); //where published=true
        //    $blogs=Blog::select('SELECT * FROM blogs'); //raw sql
        //    $blogs=Blog::all(); //fetch all the records
        $blogs = Blog::with('blog_info')->get();  //fetch all data with one to
one relation
        //    $blogs=Blog::with('blog_info')->paginate(10);  // fetch data
relationship with one to one
        return $blogs;
    }


    // =========DELETE==========
    public function delete($blog)
    {
        // delete image from local storage
        if (isset($blog->blog_info->image)) {
            Storage::delete('public/images/blogs/' . $blog->blog_info->image);
        }
        // if cascade on delete not used:
        // $blog->blog_info->delete(); //delete child record
        // $blog->delete(); //delete parent record
        // if cascade on delete used:
        $blog->delete();
    }

    // ========FETCH(Single Blog)======
    public function singleBlog($blog)
    {
```

```php
        // $blogs=Blog::where('slug',$blog->slug)->first();
        return $blog;
    }

    // =======UPDATE(PUT)==============
    public function updateService($request, $blog)
    {
        // $blog->update($request); //update data in db if no image & relation

        // Check if a new image is uploaded
        if (isset($request['image'])) {
            // Delete the old image from storage folder
            Storage::delete('public/images/blogs/' . $blog->blog_info->image);
            // Store the new image
            $timestamp = now()->timestamp;
            $originalName = $request['image']->getClientOriginalName();
            $imageName = $timestamp . '-' . $originalName;
            $request['image']->storeAs('public/images/blogs', $imageName);
            // Update the image name in the $request array
            $request['image'] = $imageName;
        } else {
            $request['image'] = $blog->blog_info->image;
        }

        // Update in parent table
        $blog->update([
            'title' => $request['title'],
            'slug' => $request['slug'],
        ]);

        // Update in child table
        $blog->blog_info->update([
            'image' => $request['image'],
            'description' => $request['description'],
        ]);
    }
}
```

Example of One to Many Relationship:

```php
<?php

namespace App\Service;

use App\Models\Blog;
use App\Models\BlogInfo;
use Illuminate\Support\Facades\Storage;

// one to many relation
class BlogService
{
    // =========POST======
    public function addService($request)
    {
        // Initialize an empty array to store image names
        $imageNames = [];

        // Store multiple images in local storage folder
        if (isset($request['image'])) {
            foreach ($request['image'] as $key => $image) {
                $timestamp = now()->timestamp;
                $originalName = $image->getClientOriginalName();
                $imageName = $timestamp . '-' . $originalName;
                $image->storeAs('public/images/blogs', $imageName);

                // Store the image name in the array
                $imageNames[] = $imageName;
            }
        }

        // Create the Blog entry
        $blog = Blog::create([
            'title' => $request['title'],
            'slug' => $request['slug'],
            'description' => $request['description'],
        ]);

        // Create BlogInfo entries associated with the Blog
        foreach ($imageNames as $imageName) {
            BlogInfo::create([
                'image' => $imageName,
                'blog_id' => $blog->id,
            ]);
```

```
        }
    }

    // ========GET(Read all)================
    public function fetchBlogs()
    {
        //    $blogs=Blog::paginate(10);  //Paginate with 10 records per page
        //    $blogs=Blog::where('published',true)->get(); //where published=true
        //    $blogs=Blog::select('SELECT * FROM blogs'); //raw sql
        //    $blogs=Blog::all(); //fetch all the records
        $blogs = Blog::with('blog_info')->get();  //fetch all data with one to
one relation
        //    $blogs=Blog::with('blog_info')->paginate(10);  // fetch data
relationship with one to one
        return $blogs;
    }

    // =========DELETE==========
    public function delete($blog)
    {
        // delete image from local storage
        if (isset($blog->blog_info)) {
            foreach ($blog->blog_info as $key => $image) {
                Storage::delete('public/images/blogs/' . $image->image);
            }
        }
        // if cascade on delete not used:
        // $blog->blog_info->delete(); //delete child record
        // $blog->delete(); //delete parent record
        // if cascade on delete used:
        $blog->delete();
    }

    // ========FETCH(Single Blog)======
    public function singleBlog($blog)
    {
        // $blogs=Blog::where('slug',$blog->slug)->first();
        return $blog;
    }

    // =======UPDATE(PUT)==============
    public function updateService($request, $blog)
    {
        // Check if new images are uploaded
        if (isset($request['image'])) {
```

```
            // Delete old images from storage folder and database
            foreach ($blog->blog_info as $oldImage) {
                // Delete from local storage
                Storage::delete('public/images/blogs/' . $oldImage->image);
                // Delete from database
                $oldImage->delete();
            }

            // Store the new images
            foreach ($request['image'] as $key => $image) {
                $timestamp = now()->timestamp;
                $originalName = $image->getClientOriginalName();
                $imageName = $timestamp . '-' . $originalName;
                $image->storeAs('public/images/blogs', $imageName);

                // Update in child table
                $blog->blog_info()->create([
                    'image' => $imageName,
                ]);
            }

            // Update in parent table
            $blog->update([
                'title' => $request['title'],
                'slug' => $request['slug'],
                'description' => $request['description'],
            ]);
        } else {
            // If no new images, update only the parent table
            $blog->update([
                'title' => $request['title'],
                'slug' => $request['slug'],
                'description' => $request['description'],
            ]);
        }
    }
}
```

Many to Many Relationship:

```
// ==========POST(Add)================
    public function addService($request)
    {
        DB::transaction(function () use ($request) {    //->roll back if not
inserted in relationship table
            // adding data in parent table
            $post = Post::create([
                'user_id' => auth()->id(),
                'caregory_id' => $request['category'],
                'title' => $request['title'],
                'description' => $request['description'],
                'status' => $request['status'],
            ]);

            // add data with many to many relation
            $post->tags()->attach($request['tags']);
        });
    }
    // ====================================

    // ==============GET All==================
    public function fetchPost()
    {
        $posts = Post::latest()->get();
        return $posts;
    }
    // ======================================

    // =========fetch single post===========
    public function view($post)
    {
        $singlePost = Post::find($post->id);
        return $singlePost;
    }
    // ====================================

    // ===========UPDATE POST==============
    public function updateService($request, $post)
    {
        // updating data in parent table
        $post->update([
            'user_id' => auth()->id(),
            'caregory_id' => $request['category'],
```

```
            'title' => $request['title'],
            'description' => $request['description'],
            'status' => $request['status'],
        ]);
        // updating data with many to many relation
        $post->tags()->sync($request['tags']);
    }
    // ====================================

    // =============DELETE=================
    public function deletePost($post)
    {
        //if not used cascadeOnDelete() then you should detach from many to many relation else you can directly use delete only
        $post->tags()->detach();
        $post->delete();
    }
    // ====================================
```

Routing:

```php
<?php

use App\Http\Controllers\Admin\BlogController;
use Illuminate\Support\Facades\Route;
// for multiple page
Route::get('/blogs',[BlogController::class,'home'])->name('blogs');
Route::get('/blogs/index', [BlogController::class, 'index'])->name('blogs.index');
Route::get('/blogs/create',[BlogController::class,'create'])->name('blogs.create');
Route::post('/blogs/store', [BlogController::class, 'store'])->name('blogs.store');
Route::get('/blogs/{slug}',[BlogController::class,'show'])->name('blogs.show');
Route::delete('/blogs/destroy/{slug}', [BlogController::class, 'destroy'])->name('blogs.destroy');
Route::get('/blogs/{slug}', [BlogController::class, 'show'])->name('blogs.show');
Route::get('/blogs/{slug}/edit', [BlogController::class, 'edit'])->name('blogs.edit');
Route::put('/blogs/update/{blog}', [BlogController::class, 'update'])->name('blogs.update');
```

```php
<?php



use App\Http\Controllers\Admin\BlogController;
use Illuminate\Support\Facades\Route;

// for single page
Route::get('/blogs',[BlogController::class,'home'])->name('blog');
Route::post('/blogs/store', [BlogController::class, 'store'])->name('blogs.store');
Route::put('/blogs/update', [BlogController::class, 'update'])->name('blogs.update');
Route::get('/blogs/index', [BlogController::class, 'index'])->name('blogs.index');
Route::get('/blogs/{slug}', [BlogController::class, 'view'])->name('blogs.view');
Route::get('/blogs/{slug}/edit', [BlogController::class, 'edit'])->name('blogs.edit');
Route::delete('/blogs/destroy/{slug}', [BlogController::class, 'destroy'])->name('blogs.destroy');
```

CRUD With AJAX:

Frontend(index.blade.php):

```
<script>
        $(document).ready(function() {
            // =====setup csrf token======
            $.ajaxSetup({
                headers: {
                    'X-CSRF-TOKEN': $('meta[name="csrf-token"]').attr('content')
                }
            });

            // =====Reset form============
            $('#addButton').click(function() {
                $('#ajaxForm')[0].reset();
                $('#categoryError').html('');
                $('#slugError').html('');
            });

            // ========ADDING DATA TO DB(POST)=============//
            var createFormData = $('#ajaxForm')[0];
            $('#saveBtn').click(function() {

                // setting empty text on error message
                $('#categoryError').html('');
                $('#slugError').html('');
                // getting form data
                var formData = new FormData(createFormData);
                // console.log(formData);
                $.ajax({
                    url: "{{ route('category.store') }}",
                    method: 'POST',
                    processData: false,
                    contentType: false,
                    data: formData,

                    success: function(response) {
                        table.draw();
                        $('#addCategory').modal('hide');
                        $('#ajaxForm')[0].reset();
                        toastify().success(response.success);
                    },
                    error: function(error) {
                        let errorMessage = error.responseJSON.errors;
                        if (errorMessage) {
```

```
                    if (errorMessage.name) {
                        $('#categoryError').html(errorMessage.name[0]);
                    }
                    if (errorMessage.slug) {
                        $('#slugError').html(errorMessage.slug[0]);
                    }
                } else {
                    toastify().error('Something went wrong');

                }

            }
        });
    });
    // ======================================================//

    // ===========READ DATA FROM DB(READ)====================//
    var table = $('#category-table').DataTable({
        "processing": true,
        "serverSide": true,
        "deferRender": true,
        "ordering": false,
        searchDelay: 3000,
        "ajax": {
            url: "{{ route('category.index') }}",
            type: 'GET',
            dataType: 'JSON'
        },
        "columns": [{
                data: 'DT_RowIndex',
                name: 'DT_RowIndex',
                searchable: false
            },
            {
                data: 'name',
                name: 'name',
            },
            {
                data: 'slug',
                name: 'slug'
            },
            {
                data: 'action',
                name: 'action',
                orderable: false,
```

```
                searchable: false
            },
        ],
        "lengthMenu": [
            [10, 20, 50, -1],
            [10, 20, 50, "All"]
        ],
        "pagingType": "simple_numbers"
    });
    // ====================================================//

    // ================DELETE CATEGORY==================//
    $('body').on('click', '.delButton', function() {
        let slug = $(this).data('slug');
        if (confirm('Are you sure you want to delete it')) {
            $.ajax({
                url: '{{ url('admin/category/', '') }}' + '/' + slug,
                method: 'DELETE',
                success: function(response) {
                    // refresh the table after delete
                    table.draw();
                    // display the delete success message
                    toastify().success(response.success);
                },
                error: function(error) {
                    console.log(error);
                }
            });
        }
    });
    // ========================================//

    // =====Fill Current Data to form while UPDATION========//
    let slug = '';
    $('body').on('click', '.editButton', function() {
        // get form slug
        slug = $(this).data('slug');

        $.ajax({
            url: '{{ url('admin/category', '') }}' + '/' + slug +'/edit',
            method: 'GET',
            success: function(response) {
                $('#editCategory').modal('show');
                $('#updateCategory').val(response.name);
                $('#updateSlug').val(response.slug);
```

```
                    $('#category_slug').val(response.slug);
                },
                error: function(error) {
                    console.log(error);
                }
            })
        });
        // ============================================================//

        // =====UPDATING CATEGORY TO DB(UPDATE/PUT)===========//
        var updateFormData = $('#ajaxFormUpdate')[0];
        $('#updateBtn').click(function() {
            // setting empty text on error message
            $('#categoryUpdateError').html('');
            $('#slugUpdateError').html('');
            // getting form data
            var formUpdateData = new FormData(updateFormData);
            $.ajax({
                url: '{{ url('admin/category/', '') }}' + '/' + slug,
                method: 'POST',
                processData: false,
                contentType: false,
                data: formUpdateData,

                success: function(response) {
                    // reload the latest row after added
                    table.draw();
                    //    hide model if success
                    $('#editCategory').modal('hide');

                    // clear form after successfully submitting
                    $('#ajaxFormUpdate')[0].reset();

                    // display success message if form submitted
                    toastify().success(response.success);
                },
                error: function(error) {
                    let errorMessage = error.responseJSON.errors;
                    // displaying error message
                    if (errorMessage) {
                        if (errorMessage.title) {
                            $('#categoryUpdateError').html(errorMessage.title
[0]);
                        }
                        if (errorMessage.slug) {
```

```
                                $('#slugUpdateError').html(errorMessage.slug[0]);
                            }
                        } else {
                            toastify().error('Something went wrong!');
                        }

                    }
                });
            });
            //
=========================================================================//
        });
    </script>
```

CategoryController:

```
 private $categoryService;

    public function __construct()
    {
        $this->categoryService = new CategoryService();
    }

    public function index(Request $request)
    {
        try {
            if ($request->ajax()) {
                $categories = $this->categoryService->fetchCategory();
                return DataTables::of($categories)
                    ->addIndexColumn()
                    ->addColumn('action', function ($row) {
                        return '<a href="javascript:void(0)" class="btn btn-info
editButton" data-slug="' . $row->slug . '">Edit</a>
                            <a href="javascript:void(0)" class="btn btn-danger
delButton" data-slug="' . $row->slug . '">Delete</a>';
                    })
                    ->rawColumns(['action'])
                    ->make(true);
            }
            return view('admin.category.index');
        } catch (\Throwable $th) {
            return back()->with('error', $th->getMessage());
        }
    }
```

```php
    public function store(CategoryRequest $request)
    {
        try {
            $this->categoryService->addService($request->validated());
            return response()->json([
                'success' => 'Category added successfully'
            ]);
        } catch (\Throwable $th) {
            return back()->with('error', $th->getMessage());
            // return response()->json(['error' =>'Something went wrong'],500);
        }
    }

    public function edit(string $slug)
    {
        try {
            $category = $this->categoryService->fetchSingleCategory($slug);
            if (!$category) {
                abort(404);
            }
            return $category;
        } catch (\Throwable $th) {
            return back()->with('error', $th->getMessage());
        }
    }

    public function update(Request $request, string $slug)
    {
        try {
            $category = Category::where('slug', $request->category_slug)->first();
            if (!$category) {
                abort(404);
            }
            $this->categoryService->updateCategory($request->except('_method', 'category_slug'), $category);
            return response()->json([
                'success' => 'Category Updated Successfully'
            ], 201);
        } catch (\Throwable $th) {
            return back()->with('error', $th->getMessage());
        }
    }

    public function destroy(string $slug)
```

```
    {
        try {
            $category = Category::where('slug', $slug)->first();
            if (!$category) {
                abort(404);
            }
            $this->categoryService->delete($category);
            return response()->json([
                'success' => 'Category Deleted Successfully'
            ], 201);
        } catch (\Throwable $th) {
            return back()->with('error', $th->getMessage());
        }
    }
```

Search Blogs without page load using AJAX:

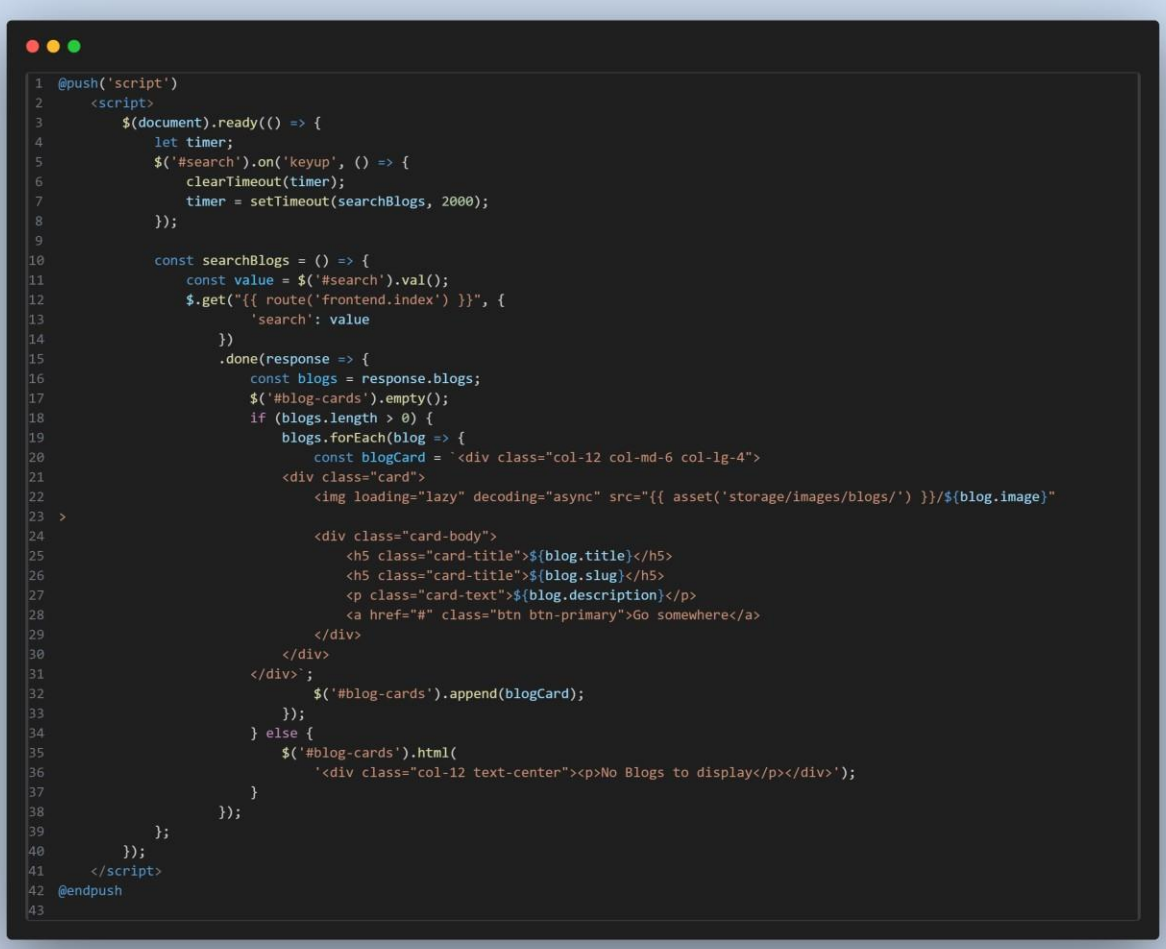
```php
// ====frontend=========
require __DIR__ . '/public.php';

// ====backend============
Route::prefix('admin')->group(function () {
    require __DIR__ . '/admin.php';
});
```

```php
Route::get('/',[HomeController::class,'index'])->name('index');
```

```php
public function index(Request $request)
{
    $query = Blog::query();
    if ($request->ajax()) {
        $blogs = $query->where('title', 'LIKE', '%' . $request->search . '%')
            ->orWhere('slug', 'LIKE', '%' . $request->search . '%')->get();
        return response()->json(['blogs' => $blogs]);
    } else {
        $blogs = $query->get();
        return view('frontend.home.index', compact('blogs'));
    }
}
```

```blade
@extends('frontend.layouts.master')
@section('content')
    <input type="search" name="search" id="search" placeholder="Search by title" autocomplete="off">
    <div class="row" id="blog-cards">
        @if (count($blogs) > 0)
            @foreach ($blogs as $blog)
                <div class="col-12 col-md-6 col-lg-4">
                    <div class="card">
                        <img loading="lazy" decoding="async" src="{{ asset('storage/images/blogs/' . $blog->image) }}"
                            alt="Post Thumbnail">
                        <div class="card-body">
                            <h5 class="card-title">{{ $blog->title }}</h5>
                            <h5 class="card-title">{{ $blog->slug }}</h5>
                            <p class="card-text">{{ $blog->description }}</p>
                            <a href="#" class="btn btn-primary">Go somewhere</a>
                        </div>
                    </div>
                </div>
            @endforeach
        @else
            <div class="col-12 text-center">
                <p>No Blogs to display</p>
            </div>
        @endif
    </div>
@endsection
```



```
@push('script')
    <script>
        $(document).ready(() => {
            let timer;
            $('#search').on('keyup', () => {
                clearTimeout(timer);
                timer = setTimeout(searchBlogs, 2000);
            });

            const searchBlogs = () => {
                const value = $('#search').val();
                $.get("{{ route('frontend.index') }}", {
                        'search': value
                    })
                    .done(response => {
                        const blogs = response.blogs;
                        $('#blog-cards').empty();
                        if (blogs.length > 0) {
                            blogs.forEach(blog => {
                                const blogCard = `<div class="col-12 col-md-6 col-lg-4">
                            <div class="card">
                                <img loading="lazy" decoding="async" src="{{ asset('storage/images/blogs/') }}/${blog.image}"
>
                                <div class="card-body">
                                    <h5 class="card-title">${blog.title}</h5>
                                    <h5 class="card-title">${blog.slug}</h5>
                                    <p class="card-text">${blog.description}</p>
                                    <a href="#" class="btn btn-primary">Go somewhere</a>
                                </div>
                            </div>
                        </div>`;
                                $('#blog-cards').append(blogCard);
                            });
                        } else {
                            $('#blog-cards').html(
                                '<div class="col-12 text-center"><p>No Blogs to display</p></div>');
                        }
                    });
            };
        });
    </script>
@endpush

```

Contact Us & Send Mail in Laravel:

1. index.blade.php

```
{{-- ===========contact us section================= --}}
<form id="contactForm">
    @csrf
    <div class="form-group">
        <label>Name:</label>
        <input type="text" class="form-control" name="name" required>
    </div>
    <div class="form-group">
        <label>Email:</label>
        <input type="email" class="form-control" name="email" required>
    </div>
    <div class="form-group">
        <label>Subject:</label>
        <input type="text" class="form-control" name="subject" required>
    </div>
    <div class="form-group">
        <label>Message:</label>
        <textarea class="form-control" name="message" required></textarea>
    </div>
    <button type="submit" class="btn btn-primary">Submit</button>
</form>
<div id="responseMessage" class="mt-3"></div>
{{-- ==========end of contact us section========== --}}
```

```
<script>
    $(document).ready(function() {

        // ==========contact us form=================
        $('#contactForm').on('submit', function(e) {
            e.preventDefault();

            $.ajax({
                url: "{{ route('contact.submit') }}",
                method: "POST",
                data: $(this).serialize(),
                beforeSend: function() {
                    $('button[type="submit"]').prop('disabled', true);
                },
                success: function(response) {
                    toastify().success(response.success);
                    $('#contactForm')[0].reset();
                    $('button[type="submit"]').prop('disabled', false);
                },
                error: function(response) {
                    let errors = response.responseJSON.errors;
                    $('button[type="submit"]').prop('disabled', false);
                }
            });
        });
        // =========end of contact us form============
    });
</script>
```

2. In web.php

```
Route::post('/contact/submit', [HomeController::class, 'submit'])->name('contact.submit');
```

3. In HomeController.php

```
public function submit(Request $request){
    $request->validate([
        'name' => 'required|string|max:255',
        'email' => 'required|email|max:255',
        'subject' => 'required|string|max:255',
        'message' => 'required|string',
    ]);

    Contact::create($request->all());
    return response()->json(['success' => 'Message Sent Successfully!']);
}
```

4. This will only save user contact information into db..so now we will create an event & Listener to send email that notify all the admin's that someone want to contact with us.
5. Make Event:

```
php artisan make:event ContactFormSubmitted
```

6. Inside your controller:

```php
public function submit(Request $request)
{
    $request->validate([
        'name' => 'required|string|max:255',
        'email' => 'required|email|max:255',
        'subject' => 'required|string|max:255',
        'message' => 'required|string',
    ]);

    $contactInfo = Contact::create($request->all());

    $data = [
        'name' => $contactInfo['name'],
        'email' => $contactInfo['email'],
        'subject' => $contactInfo['subject'],
        'message' => $contactInfo['message']
    ];
    event(new ContactFormSubmitted($data));

    return response()->json(['success' => 'Message Sent Successfully!']);
}
```

7. Inside your event ContactFormSubmitted.php

```php
public $data;

/**
 * Create a new event instance.
 */
public function __construct($data)
{
   $this->data=$data;
}
```

8. Make listener to listen your event(ContactFormSubmitted.php)

```
> php artisan make:listener SendContactFormEmail --event=ContactFormSubmitted
```

```php
public function handle(ContactFormSubmitted $event): void
{
    $users=User::get();
    foreach ($users as $key => $user) {
        // send mail
        Mail::to($user->email)->queue(new ContactMail($event->data));
    }

}
```

9. Inside EventServiceProvider.php

```php
protected $listen = [
    Registered::class => [
        SendEmailVerificationNotification::class,
    ],
    BlogCreated::class=>[
        NotifyUser::class
    ],
    ContactFormSubmitted::class=>[
        SendContactFormEmail::class
    ]
];
```

10. Make mail to send mail:

```
php artisan make:mail ContactMail
```

```php
public function __construct($data)
{
    $this->data=$data;
}
```

```php
public function content(): Content
{
    return new Content(
        markdown: 'contact.contact_mail',
    );
}
```

11. Inside views/contact/contact_mail.blade.php

```
@component('mail::message')
# Contact Form Submission

**Name:** {{ $data['name'] }}

**Email:** {{ $data['email'] }}

**Subject:** {{ $data['subject'] }}

**Message:**
{{ $data['message'] }}

Thanks,<br>
{{ config('app.name') }}
@endcomponent
```

12. Inside .env

```
MAIL_MAILER=smtp
MAIL_HOST=smtp.gmail.com
MAIL_PORT=465
MAIL_USERNAME=nontoxicyubs@gmail.com
MAIL_PASSWORD=mjlwtqjpqhonfynv
MAIL_ENCRYPTION=tls
MAIL_FROM_ADDRESS=nontoxicyubs@gmail.com
MAIL_FROM_NAME="${APP_NAME}"
```

Get password from security->2-step verification->Next->App Passwords

13. Queues allow you to defer time-consuming tasks, such as sending emails or generating reports, to be processed in the background, freeing up your application to handle other requests.

a) php artisan queue:table

b) in .env

```
QUEUE_CONNECTION=database
```

c) Inside ContactMail.php

```
class ContactMail extends Mailable implements ShouldQueue
```

d) php artisan migrate

e) php artisan queue:listen

❖ Authentication using Laravel UI Package:

bash Copy code

```
composer require laravel/ui
php artisan ui bootstrap --auth
npm install && npm run dev
```

Generate a mailable to handle the custom email:

bash Copy code

```
php artisan make:mail ResetPasswordMail
```

Edit the `ResetPasswordMail` class in `app/Mail/ResetPasswordMail.php`:

```php
<?php

namespace App\Mail;

use Illuminate\Bus\Queueable;
use Illuminate\Mail\Mailable;
use Illuminate\Queue\SerializesModels;

class ResetPasswordMail extends Mailable
{
    use Queueable, SerializesModels;

    public $token;
    public $email;

    /**
     * Create a new message instance.
     *
     * @return void
     */
    public function __construct($token, $email)
    {
        $this->token = $token;
        $this->email = $email;
    }

    /**
     * Build the message.
     *
     * @return $this
     */
    public function build()
    {
        return $this->view('emails.reset_password')
            ->with([
                'token' => $this->token,
                'email' => $this->email,
            ]);
    }
}

```

Create the Blade template at `resources/views/emails/reset_password.blade.php`:

```blade
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Password Reset</title>
    <style>
        .email-container {
            font-family: Arial, sans-serif;
            color: #333;
            padding: 20px;
            border: 1px solid #ddd;
            border-radius: 5px;
            width: 600px;
            margin: 0 auto;
        }
        .email-header {
            text-align: center;
            margin-bottom: 20px;
        }
        .email-header img {
            width: 150px;
        }
        .email-body {
            margin-bottom: 20px;
        }
        .email-footer {
            text-align: center;
            color: #aaa;
            font-size: 12px;
        }
        .btn {
            background-color: #007BFF;
            color: white;
            padding: 10px 20px;
            text-decoration: none;
            border-radius: 5px;
            display: inline-block;
        }
    </style>
</head>
<body>
    <div class="email-container">
        <div class="email-header">
            <img src="{{ asset('images/company-logo.png') }}" alt="Company Logo">
        </div>
        <div class="email-body">
            <p>Hello!</p>
            <p>You are receiving this email because we received a password reset request for your account.
</p>
            <p>
                <a href="{{ url('password/reset', $token) }}?email={{ urlencode($email) }}" class="btn">
Reset Password</a>
            </p>
            <p>This password reset link will expire in 60 minutes.</p>
            <p>If you did not request a password reset, no further action is required.</p>
        </div>
        <div class="email-footer">
            Regards, <br>
            Your Company
        </div>
    </div>
</body>
</html>

```

```php
<?php
// ==========user models=========

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Foundation\Auth\User as Authenticatable;
use Illuminate\Notifications\Notifiable;
use Laravel\Sanctum\HasApiTokens;
use Illuminate\Support\Facades\Mail;
use App\Mail\ResetPasswordMail;

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

    /**
     * Override the sendPasswordResetNotification method to use custom mailable.
     *
     * @param string $token
     * @return void
     */
    public function sendPasswordResetNotification($token)
    {
        Mail::to($this->email)->send(new ResetPasswordMail($token, $this->email));
    }

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

    /**
     * The attributes that should be hidden for serialization.
     *
     * @var array<int, string>
     */
    protected $hidden = [
        'password',
        'remember_token',
    ];

    /**
     * The attributes that should be cast.
     *
     * @var array<string, string>
     */
    protected $casts = [
        'email_verified_at' => 'datetime',
        'password' => 'hashed',
    ];
}

```

Email verification:

Step 1: (web.php)

```php
// ========Authentication=======
Auth::routes(['verify' => true]);
// =========end of Authentication===
```

```php
// ===========backend=========
Route::group(['middleware' => ['auth', 'verified']], function() {
    Route::prefix('admin')->group(function() {
        require __DIR__.'/admin.php';
    });
});
// ==========end of backend====
```

Step 2: (VerificationController.php)

```php
class VerificationController extends Controller
{
    use VerifiesEmails;

    protected $redirectTo = '/admin/home';

    public function __construct()
    {
        $this->middleware('auth');
        $this->middleware('signed')->only('verify');
        $this->middleware('throttle:6,1')->only('verify', 'resend');
    }

    public function verify(Request $request)
    {
        $user = $request->user();

        if ($user->hasVerifiedEmail()) {
            return redirect($this->redirectPath());
        }

        if ($user->markEmailAsVerified()) {
            event(new Verified($user));
        }

        return redirect($this->redirectPath())->with('verified', true);
    }
}
```

Step 3: (User modal)

```php
use HasApiTokens, HasFactory, Notifiable, HasRoles;
```

```php
/**
* Send the email verification notification.
*
* @return void
*/
public function sendEmailVerificationNotification()
{
    $this->notify(new VerifyEmailNotification);
}
```

Step 4: (VerifyEmailNotification.php)

```php
class VerifyEmailNotification extends BaseVerifyEmail
{
    /**
     * Build the mail representation of the notification.
     *
     * @param  mixed  $notifiable
     * @return \Illuminate\Notifications\Messages\MailMessage
     */
    public function toMail($notifiable)
    {
        $verificationUrl = $this->verificationUrl($notifiable);

        return (new MailMessage)
            ->view('emails.verify', ['user' => $notifiable, 'verificationUrl' => $verificationUrl]);
    }

    /**
     * Get the verification URL for the given notifiable.
     *
     * @param  mixed  $notifiable
     * @return string
     */
    protected function verificationUrl($notifiable)
    {
        return URL::temporarySignedRoute(
            'verification.verify',
            now()->addMinutes(60),
            ['id' => $notifiable->getKey(), 'hash' => sha1($notifiable->getEmailForVerification())]
        );
    }
}
```

Step 5: (resources/views/emails/verify.blade.php)

```blade
<a href="{{ $verificationUrl }}" target="_blank"
    style="color: #ffffff; text-decoration: none; font-weight: bold;">Click
    here to verify</a>
```

Configure Mail Settings

Make sure your `.env` file has the correct mail configuration:

env Copy code

```
MAIL_MAILER=smtp
MAIL_HOST=smtp.mailtrap.io
MAIL_PORT=2525
MAIL_USERNAME=null
MAIL_PASSWORD=null
MAIL_ENCRYPTION=null
MAIL_FROM_ADDRESS="hello@example.com"
MAIL_FROM_NAME="${APP_NAME}"
```

Replace the placeholder values with your actual SMTP server details.

```
MAIL_MAILER=smtp
MAIL_HOST=smtp.gmail.com
MAIL_PORT=465
MAIL_USERNAME=nontoxicyubs@gmail.com
MAIL_PASSWORD=kjzqaaolfsvvutrn
MAIL_ENCRYPTION=tls
MAIL_FROM_ADDRESS=nontoxicyubs@gmail.com
MAIL_FROM_NAME="${APP_NAME}"
```

Get password from security<2-step-verification<App Passwords

❖ Middleware:

```
php artisan make:middleware AdminMiddleware
```

```php
 */
0 references | 0 overrides
public function handle(Request $request, Closure $next): Response
{
    if (!auth()->check())  {
        return redirect()->route('login')->with('error', 'Please log in to access this page.');
    }

    return $next($request);
}
```

```php
protected $middlewareAliases = [
    'auth' => \App\Http\Middleware\Authenticate::class,
    'auth.basic' => \Illuminate\Auth\Middleware\AuthenticateWithBasicAuth::class,
    'auth.session' => \Illuminate\Session\Middleware\AuthenticateSession::class,
    'cache.headers' => \Illuminate\Http\Middleware\SetCacheHeaders::class,
    'can' => \Illuminate\Auth\Middleware\Authorize::class,
    'guest' => \App\Http\Middleware\RedirectIfAuthenticated::class,
    'password.confirm' => \Illuminate\Auth\Middleware\RequirePassword::class,
    'precognitive' => \Illuminate\Foundation\Http\Middleware\HandlePrecognitiveRequests::class,
    'signed' => \App\Http\Middleware\ValidateSignature::class,
    'throttle' => \Illuminate\Routing\Middleware\ThrottleRequests::class,
    'verified' => \Illuminate\Auth\Middleware\EnsureEmailIsVerified::class,
    'auth.admin' => \App\Http\Middleware\AdminMiddleware::class,
];
```

web.php

```php
<?php

use Illuminate\Support\Facades\Auth;
use Illuminate\Support\Facades\Route;


// ==========frontend==========
Route::name('frontend.')->group(function () {
    require __DIR__ . '/frontend.php';
});
// ==========end of frontend========


// =======auth================
Auth::routes();
// =========end of auth========

// ========admin===============
Route::middleware(['auth.admin'])->group(function () {
    Route::prefix('admin')->group(function () {
        require __DIR__ . '/admin.php';
    });
});
// =========end of auth=========
```

admin.php

```php
<?php

use App\Http\Controllers\Admin\HomeController;
use Illuminate\Support\Facades\Route;

Route::resources([
    // 'blogs'=>BlogController::class,
]);

Route::get('/dashboard',[HomeController::class,'index'])->name('admin.dashboard');
```

frontend.php

```php
<?php

use App\Http\Controllers\Frontend\HomeController;
use Illuminate\Support\Facades\Route;

Route::get('/',[HomeController::class,'index'])->name('index');
```

https://amanj0314.medium.com/how-to-use-laravel-middleware-to-protect-your-application-1c76aa0acded

---

❖ Spatie user Roles & Permission management:

https://spatie.be/docs/laravel-permission/v6/installation-laravel

```
composer require spatie/laravel-permission
```

in your `config/app.php` file:

```
'providers' => [
    // ...
    Spatie\Permission\PermissionServiceProvider::class,
];
```

```
php artisan vendor:publish --provider="Spatie\Permission\PermissionServiceProvider"
```

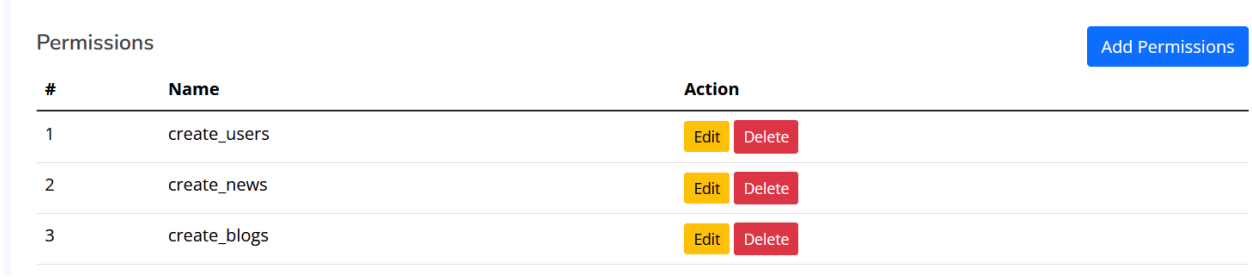
```
php artisan optimize:clear
# or
php artisan config:clear
```

```
php artisan migrate
```

Add the necessary trait to your User model:

```
// The User model requires this trait
use HasRoles;
```

Make Permission and Role Controller and do crud operation:



Roles

Add Roles

#	Name	Action
1	staff	Add/Edit Role Permission Edit Delete
2	user	Add/Edit Role Permission Edit Delete
3	admin	Add/Edit Role Permission Edit Delete
4	super_admin	Add/Edit Role Permission Edit Delete

Role: staff

Back

Permissions

- [ ] create_blogs
- [x] create_news
- [x] create_users

Submit

```php
Route::resources([
    'permissions'=>PermissionController::class,
    'roles'=>RoleController::class,
]);

Route::get('/roles/{roleId}/give-permissions',[RoleController::class,'addPermissionToRole'])->name('add.permissions.to.role');
Route::put('/roles/{roleId}/give-permissions',[RoleController::class,'givePermissionToRole'])->name('give.permissions.to.role');
```

```php
public function addPermissionToRole($roleId)
{
    try {
        $permissions = Permission::all();

        $role = Role::findOrFail($roleId);

        $rolePermissions = DB::table('role_has_permissions')
        ->where('role_has_permissions.role_id', $role->id)
        ->pluck('role_has_permissions.permission_id', 'role_has_permissions.permission_id')->all();


        return view('admin.roles.add-permissions', compact('permissions', 'role','rolePermissions'));
    } catch (\Throwable $th) {
        return back()->with('error', $th->getMessage());
    }
}

public function givePermissionToRole(Request $request,$roleId)
{
    try{
        $role = Role::findOrFail($roleId);
        $role->syncPermissions($request->permission);
        return redirect()->back()->with('success','Permission added to role');
    }catch(\Throwable $th){
        return back()->with('error',$th->getMessage());
    }
}
```

```blade
<form action="{{route('give.permissions.to.role',$role->id)}}" method="POST">
    @csrf
    @method('PUT')
    <div class="d-flex align-items-center" style="flex-wrap: wrap;gap:30px">
        @foreach ($permissions as $permission)
        <div class="form-group form-check">
            <input type="checkbox" class="form-check-input" id="{{ $permission->name }}" name="permission[]"
                value="{{ $permission->name }}" @checked(in_array($permission->id, $rolePermissions))
            >
            <label class="form-check-label" for="{{$permission->name}}">{{ $permission->name }}</label>
        </div>
        @endforeach

    </div>
    <button type="submit" class="btn btn-primary mt-3">Submit</button>
</form>
```

Make user controller and perform crud operation:

Users

Add User

#	Name	Email	Roles	Action
1	Saru	gocigicy@mailinator.com	admin staff	Edit Delete
2	Micah Sweeney	bedugolo@mailinator.com	staff	Edit Delete
3	Uma Swanson	varefo@mailinator.com	staff	Edit Delete
4	August Wilkins	bicozosivo@mailinator.com	super_admin staff	Edit Delete
5	Yubraj Koirala	yubrajkoirala7278@gmail.com	super_admin	Edit Delete

```blade
<td>
    {{-- get specific user roles --}}
    <div class="d-flex align-items-center" style="gap: 10px;flex-wrap:wrap">
        @if (count($user->getRoleNames())> 0)
            @foreach ($user->getRoleNames() as $role)
                <span class="badge rounded-pill text-bg-success">{{ $role }}</span>
            @endforeach
        @endif
    </div>
</td>
```

Edit Users

Back

Name

Saru

Email

gocigicy@mailinator.com

Password

Roles

× admin × staff

Submit

```
<select class="form-select" id="select-roles" data-placeholder="Choose Roles" multiple name="roles[]">
    @if (count($roles) > 0)
        @foreach ($roles as $role)
            <option value="{{ $role }}"
                {{ in_array($role, $userRoles) || in_array($role, old('roles', [])) ? 'selected' : '' }}>
                {{ $role }}
            </option>
        @endforeach
    @endif
</select>
```

```php
public function index()
    {
        try {
            $users = User::latest()->get();
            return view('admin.users.index', compact('users'));
        } catch (\Throwable $th) {
            return back()->with('error', $th->getMessage());
        }
    }

    public function create()
    {
        try {
            $roles = Role::pluck('name', 'name')->all();
            return view('admin.users.create', compact('roles'));
        } catch (\Throwable $th) {
            return back()->with('error', $th->getMessage());
        }
    }

    public function store(UserRequest $request)
    {
        try {
            $user = User::create([
                'name' => $request->name,
                'email' => $request->email,
                'password' => Hash::make($request->password),
            ]);

            $user->syncRoles($request->roles);

            return redirect()->route('users.index')->with('success', 'User added successfully!');
        } catch (\Throwable $th) {
            return back()->with('error', $th->getMessage());
        }
    }
```

```php
public function edit(User $user)
    {
        try {
            $roles = Role::pluck('name', 'name')->all();
            $userRoles = $user->roles->pluck('name', 'name')->all();
            return view('admin.users.edit', compact('user', 'roles', 'userRoles'));
        } catch (\Throwable $th) {
            return back()->with('error', $th->getMessage());
        }
    }

    public function update(UserRequest $request, User $user)
    {
        try {
            $data = [
                'name' => $request->name,
                'email' => $request->email,
            ];

            if (!empty($request->password)) {
                $data += [
                    'password' => Hash::make($request->password),
                ];
            }

            $user->update($data);

            $user->syncRoles($request->roles);

            return redirect()->route('users.index')->with('success', 'User updated successfully!');
        } catch (\Throwable $th) {
            return back()->with('error', $th->getMessage());
        }
    }

    public function destroy(User $user)
    {
        try {
            $user->delete();
            return back()->with('success', 'User deleted successfully!');
        } catch (\Throwable $th) {
            return back()->with('error', $th->getMessage());
        }
    }
```

Now, for roles and permissions:
Step1:
https://spatie.be/docs/laravel-permission/v6/basic-usage/middleware

In Laravel 9 and 10 you can add them in `app/Http/Kernel.php` :

```php
// Laravel 9 uses $routeMiddleware = [
//protected $routeMiddleware = [
// Laravel 10+ uses $middlewareAliases = [
protected $middlewareAliases = [
    // ...
    'role' => \Spatie\Permission\Middleware\RoleMiddleware::class,
    'permission' => \Spatie\Permission\Middleware\PermissionMiddleware::class,
    'role_or_permission' => \Spatie\Permission\Middleware\RoleOrPermissionMiddleware::class,
];
```

Step2:

In admin.php/web.php

```php
// roles and permission:
Route::group(['middleware' => ['role:super_admin|admin']], function () {
    // super_admin and admin can only access this routes
    Route::resources([
        'users' => UserController::class
    ]);
    // super_admin and admin with permission "add project" can only access this routes
    Route::get('/store/project', [UserController::class, 'addProject'])->name('add.project')
        ->middleware('permission:add project');
});
```

Roles and permission in resource controller: Step 1:

```php
// roles and permission:
Route::group(['middleware' => ['role:super_admin|admin']], function () {
    /* super_admin and admin can only access user controller.*/
    Route::resources([
        'users' => UserController::class
    ]);
});
```

Step 2: (user controller)

```php
0 references | 0 overrides
public function __construct() {
    // permission with "edit user" can only access edit and update routes of user controller
    $this->middleware('permission:edit user',['only'=>['edit','update']]);
    // permission with "delete user" can only access destroy route of user controller
    $this->middleware('permission:delete user',['only'=>['destroy']]);
}
```

Standard way to write the above roles and permission codes:

Step 1: Kernel.php

```php
'auth.admin' => \App\Http\Middleware\AdminMiddleware::class,
```

Step 2: admin.php

```php
// roles and permission:
Route::group(['middleware' => ['auth.admin']], function () {
    /* super_admin and admin can only access user controller.*/
    Route::resources([
        'users' => UserController::class
    ]);
});
```

Step 3: AdminMiddleware.php

```php
public function handle(Request $request, Closure $next): Response
{
    if (Auth::check()) {
        $user = Auth::user();
        if ($user->hasRole(['super_admin', 'admin'])) {
            return $next($request);
        }
        Auth::logout();
        return redirect()->route('login')->with('error', "You have been logged out because you don't
                have permission to access this page.");
        // abort(403, "User doesn't have permission to access this page!");
    }
    return redirect()->route('login')->with('error', 'Please log in to access this page.');
}
```

Step 4: Create RoleSeeder and AdminSeeder:

```php
public function run(): void
{
    Role::create([
        'name'=>'super_admin',
    ]);
}
```

```php
public function run(): void
{
    $user=User::create([
        'name' => 'Yubraj Koirala',
        'email' => 'yubrajkoirala7278@gmail.com',
        'password' => Hash::make('12345678')
    ]);
    $user->syncRoles(['super_admin']);
}
```

Laravel API CRUD:

Step:1

```
composer create-project laravel/laravel apicrud
```

Step:2

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

```php
use HasFactory;
public function getRouteKeyName()
{
    return 'slug';
}
protected $fillable = ['name', 'slug', 'image', 'price', 'description', 'cross_price', 'color'];
// append image_url column
protected $appends = ['image_url'];
public function getImageUrlAttribute()
{
    return $this->image ? url(Storage::url('images/products/' . $this->image)) : null;
}
```

Step 3:

```
php artisan make:request StorePostRequest
```

```php
return [
    'slug' => ['required', 'max:255', 'min:3', Rule::unique('products')->ignore($this->product)],
    'name' => ['required', 'max:255'],
    'image' => $this->isMethod('POST') ? ['required', 'mimes:png,jpg,jpeg,webp'] :
        ['nullable', 'sometimes', 'mimes:png,jpg,jpeg,webp'],
    /*'images' => $this->isMethod('POST')? ['required', 'array']
        : ['nullable', 'sometimes', 'array'],
    'images.*' => ['mimes:png,jpg,jpeg,webp'], */
    'price'=>['required'],
    'cross_price'=>['required'],
    'description'=>['required'],
    'color'=>['required']
];
```

Step 4:

```
php artisan make:resource PostResource
```

```php
public function toArray(Request $request): array
{
    // return parent::toArray($request);
    return [
        // 'id'=>$this->id,
        'name'=>$this->name,
        'slug'=>$this->slug,
        // 'image' => $this->image,
        'image_url'=>$this->image_url,
        'price'=>$this->price,
        'cross_price'=>$this->cross_price,
        'color'=>$this->color,
        'description'=>$this->description
    ];
}
```

Step 5:

```
php artisan make:controller Api/PostController --model=Post
```

- **GET**

```php
public function index(Request $request)
{
    try {
        $products=$this->productService->fetchService($request->all());
        return ProductResource::collection($products);
        // return new ProductCollection($products);
    } catch (\Exception $e) {
        // Return an error response
        return response()->json([
            'message' => 'Something went wrong. Please try again later.'
        ], 500);
    }
}
```

```php
public function fetchService($request)
{
    $productQuery = Product::query()->latest();

    // filter
    if (!empty($request['keyword'])) {
        $productQuery->where('name', 'like', '%' . $request['keyword'] . '%')
            ->orWhere('price', '=', $request['keyword']);
    }

    // pagination
    $perPage = $request['perPage'] ?? 10;
    $products = $productQuery->paginate($perPage);

    return $products;
}
```

- **POST**

```php
0 references | 0 overrides
public function store(ProductRequest $request)
{
    try {
        $product = $this->productService->addService($request->validated());
        // Return the created product as a JSON response
        return (new ProductResource($product))->additional(['message' => 'Product added successfully!']);
    } catch (\Exception $e) {
        // Return an error response
        return response()->json([
            'message' => 'Something went wrong. Please try again later.'
        ], 500);
    }
}
// ==========ADD Product============
1 reference | 0 overrides
public function addService($request)
{
    //  save image in storage folder
    if (isset($request['image'])) {
        $imageName = null;
        $timestamp = now()->timestamp;
        $originalName = $request['image']->getClientOriginalName();
        $imageName = $timestamp . '-' . $originalName;
        $request['image']->storeAs('public/images/products/', $imageName);
        $request['image'] = $imageName;
    }

    // store in db
    $product = Product::create($request);
    return $product;
}
```

- **DELETE**

```php
public function destroy(Product $product)
{
    try {
        $this->productService->deleteService($product);
        $message='Product deleted successfully!';
    } catch (\Exception $e) {
        // Return an error response
        $message='Something went wrong. Please try again later.';
    }
    return response()->json([
        'message' => $message
    ]);
}
```

```php
public function deleteService($product)
{
    // delete image from local storage
    if (isset($product->image)) {
        Storage::delete('public/images/products/' . $product->image);
    }
    $product->delete();
}
```

- **PUT/UPDATE**

```php
public function update(ProductRequest $request, Product $product)
{
    try {
        $updatedProduct = $this->productService->updateService($request->except('_method'), $product);
        return (new ProductResource($updatedProduct))->additional(['message' => 'Product updated successfully!']);
    } catch (\Exception $e) {
        // Return an error response
        return response()->json([
            'message' => 'Something went wrong. Please try again later.'
        ], 500);
    }
}
```

```php
public function updateService($request, $product)
{
    // Check if a new image is uploaded
    if (isset($request['image'])) {
        // Delete the old image from storage folder
        Storage::delete('public/images/products/' . $product->image);
        // Store the new image
        $timestamp = now()->timestamp;
        $originalName = $request['image']->getClientOriginalName();
        $imageName = $timestamp . '-' . $originalName;
        $request['image']->storeAs('public/images/products', $imageName);
        // Update the image name in the $request array
        $request['image'] = $imageName;
    }
    // update in db
    $product->update($request);
    return $product;
}
```

- **Get Single Product**

```php
public function show(Product $product)
{
    try {
        return new ProductResource($product);
    } catch (\Exception $e) {
        // Return an error response
        return response()->json([
            'message' => 'Something went wrong. Please try again later.'
        ], 500);
    }
}
```

Step 6:

For Laravel 11 : php artisan install:api; php artisan route:cache; php artisan route:list

```php
<?php

use App\Http\Controllers\Api\ProductController;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Route;

Route::group(['prefix'=>'v1'],function(){
    // Route::apiResource('products',ProductController::class);
    Route::apiResource('products', ProductController::class)->except([
        'create', 'show', 'edit'
    ]);
});
```

API Integration in VUE JS:

Step 1: (Project Setup References)

https://github.com/yubrajkoirala7278/laravel-vue-api-crud

Step 2: (.env.development)

.env.development

```
VITE_API_HOST=http://127.0.0.1:8000/api/v1/
```

Step 3: (service.js)

service.js

src > modules > backend > products > service.js > ...

```
import axios from "axios";
import { displaySuccessMessage, displayErrorMessage } from "@/utils";

const api = axios.create({
  baseURL: import.meta.env.VITE_API_HOST,
});

========FETCH(GET) Product with filter==============
export const fetchProductsApi = async (productFilter = {}) => {
  try {
    const response = await api.get("products", {
      params: productFilter,
    });
    return response;
  } catch (error) {
    displayErrorMessage("Something went wrong!");
    console.log(error);
  }
};
// ================================================
// ==============DELETE=========================
export const deleteProductApi = async (slug) => {
  try {
    const response = await api.delete(`products/${slug}`);
    displaySuccessMessage(response.data.message);
  } catch (error) {
    displayErrorMessage("Something went wrong");
    console.log(error);
  }
};
```

```
// =========POST(ADD PRODUCT)====================
// 'Content-Type':'application/json'
// 'Content-Type':'multipart/form-data',
export const addProductApi = async (data) => {
  try {
    const response = await api.post("products", data, {
      headers: {
        "Content-Type": "multipart/form-data",
      },
    });
    displaySuccessMessage(response.data.message);
    return response;
  } catch (error) {
    displayErrorMessage(error.response.data.message);
    return false;
  }
};
// ==============================================

// ===============Fetch single Product===========
export const fetchSingleProductsApi = async (slug) => {
  try {
    const response = await api.get(`products/${slug}`);
    return response;
  } catch (error) {
    displayErrorMessage("Something went wrong");
    console.log(error);
  }
};
// ==============================================
```

```
// =======UPDATE(PUT)============================
export const updateProductApi = async (data, slug) => {
  try {
    const response = await api.post(`products/${slug}`, data, {
      headers: {
        "Content-Type": "multipart/form-data",
      },
      params: {
        _method: "put",
      },
    });
    displaySuccessMessage(response.data.message);
    return response;
  } catch (err) {
    displayErrorMessage(err.response.data.message);
    return false;
  }
};
// ==============================================
```

Step 4: (store.js)

```
import { defineStore } from "pinia";
import { ref } from "vue";
import {
  fetchProductsApi,
  deleteProductApi,
  addProductApi,
  fetchSingleProductsApi,
  updateProductApi,
} from "./service";

export const useProducts = defineStore("products", () => {
  // ===========reactive state====================
  const products = ref([]);
  const isLoading = ref(false);
  const error = ref(null);
  const singleProduct = ref({});
  const totalProducts=ref(null);
  const newProduct=ref(null);
  // ==============================================
```

```
// ============FETCH(GET) Product with filter==========
const fetchProducts = async (productFilter) => {
  try {
    isLoading.value = true;
    const apiData = await fetchProductsApi(productFilter);
    totalProducts.value=apiData.data.meta.total;
    products.value = apiData.data.data.map((item, index) => {
      return { sno: '#', ...item };
    });
  } catch (err) {
    error.value = err;
  } finally {
    isLoading.value = false;
  }
};
// ================================================

// ===============DELETE==============================
const deleteProduct = async (slug) => {
  try {
    await deleteProductApi(slug);
  } catch (error) {
    console.log(error);
  }
};
// ================================================
// =========POST(ADD PRODUCT)=======================
const addProduct = async (data) => {
  try {
    const response = await addProductApi(data);
    newProduct.value=response.data.data;
  } catch (error) {
    return false;
  }
};
// ================================================
```

```
// ============FETCH(GET) Single Product with id==========
const fetchSingleProduct = async (slug) => {
  try {
    isLoading.value = true;
    const apiData = await fetchSingleProductsApi(slug);
    singleProduct.value = apiData.data.data;
  } catch (err) {
    error.value = err;
  } finally {
    isLoading.value = false;
  }
};
// ====================================================

  // ================UPDATE(PUT)============================
  const updateProduct=async(data,slug)=>{
    try{
        let response=await updateProductApi(data,slug);
        return response.data.data;
    }catch(err){
        error.value=err;
    }
    }
  // ========================================================

  // ==========return all the method and values=========
  return {
    isLoading,
    error,
    fetchProducts,
    products,
    deleteProduct,
    addProduct,
    fetchSingleProduct,
    singleProduct,
    updateProduct,
    totalProducts,
    newProduct
  };
  // ================================================
});
```

Step 5: (utils.js)

```
// =====dataTable headers=====
export const headers = [
    {
      title: "S/N",
      key: "sno",
      sortable: false,
      align: "center",
    },
    {
      title: "Color",
      key: "color",
      sortable: true,
      align: "center",
    },
    {
      title: "Actions",
      key: "action",
      sortable: false,
      align: "center",
    },
  ];
  // ======================================
```

```
// ======product category with color======
export function getProductStatusClass(status) {
  switch (status) {
    case "men's clothing":
      return "green";
    case "jewelery":
      return "yellow";
    case "electronics":
      return "pink";
    default:
      return "red";
  }
}
// ======================================
```

Step 6: (product_view.vue)

```vue
<script setup>
import { reactive, ref, computed, watch } from "vue";
import { useProducts } from "./store";
import { storeToRefs } from "pinia";
import { headers } from "./utils.js";
// ===============setup==============
const productStore = useProducts();
const products = computed(() => productStore.products);
// ========================================

// =========fetch, pagination and filter products=============
const filter = reactive({
    sortBy: [],
    perPage: 10,
    page: 1,
});
let timeOutId = null;
function loadItems({ page, itemsPerPage }) {
    clearTimeout(timeOutId);
    filter.perPage = itemsPerPage;
    filter.page = page;
    timeOutId = setTimeout(() => {
        productStore.fetchProducts(filter);
    }, 2000);
}
// ================================================

// ======Delete Product====================
async function deleteProducts(slug) {
    var result = confirm("Want to delete?");
    if (result) {
        await productStore.deleteProduct(slug);
        productStore.products = productStore.products.filter((currProduct) => {
            return currProduct.slug != slug;
        });
    }
}
// ========================================
```

```vue
<!-- =======display data with dataTable========== -->
  <v-data-table-server
    :items-per-page="filter.itemsPerPage"
    :loading="productStore.isLoading"
    :headers="headers"
    :items="products"
    :items-length="productStore.totalProducts"
    :sort-by="filter.sortBy"
    :items-per-page-options="[
      { title: '5', value: 5 },
      { title: '10', value: 10 },
      { title: 'All', value: productStore.totalProducts },
    ]"
    @update:options="loadItems"
    fixed-header
    height="500"
    :hover="true"
  >
    <!-- display image -->
    <template v-slot:[`item.image_url`]="{ item }">
      <v-img :width="50" cover :src="item.image_url"></v-img>
    </template>
    <!-- display actions -->
    <template v-slot:[`item.action`]="{ item }">
      <div class="d-flex justify-space-evenly">
        <v-btn
          icon="mdi:mdi-pencil"
          variant="text"
          density="compact"
          color="warning"
          @click="
            updateForm(item);
            dialog = true;
            btnName = 'Update';
          "
        ></v-btn>
        <v-btn
          icon="mdi:mdi-trash-can-outline"
          variant="text"
          density="compact"
          color="red"
          @click="deleteProducts(item.slug)"
        ></v-btn>
        <router-link
          :to="{
            name: 'admin-project-details',
            params: { slug: item.slug },
          }"
        >
          <v-btn
            icon="mdi:mdi-eye"
            variant="text"
            density="compact"
            color="green"
          ></v-btn>
        </router-link>
      </div>
    </template>
  </v-data-table-server>
```

```vue
<!-- ==========Add/Update Product================== -->
  <div class="text-start pa-4">
    <v-btn
      @click="
        resetForm();
        dialog = true;
        btnName = 'Submit';
      "
      color="green"
      class="text-capitalize"
    >
      Add Product
    </v-btn>

    <v-dialog v-model="dialog" width="auto" persistent scrollable>
      <v-card max-width="700" min-width="700" title="Add Product">
        <!-- form -->
        <v-form :valid="valid" @submit.prevent="submit" ref="productForm">
          <v-container>
            <v-row>
              <!-- name -->
              <v-col cols="12">
                <v-text-field
                  v-model="product.name"
                  :rules="[rules.required]"
                  label="Product Name"
                ></v-text-field>
              </v-col>
              <!-- slug -->
              <v-col cols="12">
                <v-text-field
                  v-model="product.slug"
                  :rules="[rules.required]"
                  label="Product Slug"
                ></v-text-field>
              </v-col>
              <!-- Image -->
              <v-col cols="12">
                <v-file-input
                  label="Product Image"
                  accept="image/png, image/jpeg, image/svg,image/jpg , image/gif"
                  prepend-icon=""
                  prepend-inner-icon="mdi-camera"
                  v-model="product.image"
                  show-size
                  counter
                  chips
                ></v-file-input>
              </v-col>
```

```
            <!-- price -->
            <v-col cols="12">
              <v-text-field
                v-model="product.price"
                :rules="[rules.price]"
                label="Product Price"
              ></v-text-field>
            </v-col>
            <!-- cross price -->
            <v-col cols="12">
              <v-text-field
                v-model="product.cross_price"
                :rules="[rules.required]"
                label="Product Cross Price"
              ></v-text-field>
            </v-col>
            <!-- Description -->
            <v-col cols="12">
              <v-textarea
                v-model="product.description"
                :rules="[rules.required]"
                label="Product Description"
              ></v-textarea>
            </v-col>
            <!-- color -->
            <v-col cols="12">
              <v-text-field
                v-model="product.color"
                :rules="[rules.required]"
                label="Product Color"
              ></v-text-field>
            </v-col>
          </v-row>
        </v-container>

        <v-card-actions>
          <v-spacer></v-spacer>
          <v-btn
            text="Cancel"
            @click="
              dialog = false;
              productForm.reset();
            "
            class="text-capitalize"
            variant="tonal"
            >Cancel</v-btn
          >
          <v-btn
            type="submit"
```

```
              :text="btnName"
              color="green"
              variant="flat"
              class="text-capitalize"
              :disabled="disableSubmitBtn"
              >Submit</v-btn
            >
          </v-card-actions>
        </v-form>
      </v-card>
    </v-dialog>
  </div>
  <!-- ======================================== -->
```

```
//============Display current row value in form for updation=========
const productSlug = ref(null);
function updateForm(item) {
  Object.assign(product, {
    slug: item.slug,
    name: item.name,
    price: item.price,
    cross_price: item.cross_price,
    description: item.description,
    color: item.description,
  });
  productSlug.value = item.slug;
}
// ==================================
```

```
// =======Add Product======================
const disableSubmitBtn = ref(false);
const product = reactive({
  slug: null,
  name: null,
  image: null,
  price: null,
  cross_price: null,
  description: null,
  color: null,
});
// rules
const rules = {
  required: (v) => !!v || "Required",
  price: (v) => (!!v && v > 10) || "Amount must be greater than 10",
};
```

```
// generate automatic slug from name of product
watch(
  () => product.name,
  (newValue) => {
    product.slug = generateSlug(newValue);
  }
);

function generateSlug(text) {
  if (text != null) {
    return text.trim().toLowerCase().replace(/\s+/g, "-");
  }
  return null;
}

// submit form if validated
const productForm = ref(null);
const dialog = ref(false);
const btnName = ref("");
const { valid } = storeToRefs(productForm);
function submit() {
  productForm.value.validate().then(async ({ valid }) => {
    if (valid) {
      disableSubmitBtn.value = true;
      if (btnName.value == "Submit") {
        // ----------add product-----------
        let response = await productStore.addProduct(product);
        // append in frontend
        if (response != false) {
          products.value.unshift({ sno: "#", ...productStore.newProduct });
          dialog.value = false;
        }
      } else {
        // ----------update product-----------
        let response = await productStore.updateProduct(
          product,
          productSlug.value
        );
        // update in frontend
        if (response != false) {
          productStore.products = productStore.products.map((currProduct) => {
            if (currProduct.slug == productSlug.value) {
              return { sno: "#", ...response }; // Replace the object with
updatedObject
            }
            return currProduct; // Keep the original object if it doesn't match the
ID
          });
          dialog.value = false;
```

```
        }
      }
      disableSubmitBtn.value = false;
    }
  });
}
// reset form
const resetForm = () => {
  Object.assign(product, {
    slug: null,
    name: null,
    image: null,
    price: null,
    cross_price: null,
    description: null,
    color: null,
  });
};

// ========================================
```

Step 7: (view single product)=>product_details.vue

```vue
<script setup>
import { useRoute } from "vue-router";
import { reactive, onMounted, computed } from "vue";
import { useProducts } from "../store";

// ====get id from url======
const route = useRoute();
const slug = route.params.slug;
// =========================

// ========fetch products=========
const productStore = useProducts();
const filter = reactive({
    productSlug: slug,
});
onMounted(() => {
    productStore.fetchSingleProduct(filter.productSlug);
});
const product = computed(() => productStore.singleProduct);
// ======================================
</script>
<template>
    <h1>Title: {{ product.name }}</h1>
    <p>Price: {{ product.price }}</p>
    <p>Description: {{ product.description }}</p>
    <p>Category: {{ product.cross_price }}</p>
    <p>Category: {{ product.color }}</p>
    <v-img :width="300" aspect-ratio="16/9" cover :src="product.image_url"></v-img>
</template>
```

Step 8: (router/index.js)

```vue
import { createRouter, createWebHistory } from "vue-router";

const routes = [
  // =======backend============
  {
    path: "/admin",
    name: "private",
    component: () => import("@/layouts/auth/auth_layout.vue"),
    children: [
      {
        path: "/admin/home",
        name: "admin-home",
        component: () => import("@/modules/backend/home/home_view.vue"),
        meta: {
          layout: "auth",
        },
      },
      {
        path: "/admin/products",
        name: "admin-products",
        component: () => import("@/modules/backend/products/product_view.vue"),
        meta: {
          layout: "auth",
        },
      },
      {
        path: "/project-details/:slug",
        name: "admin-project-details",
        component: () =>
          import("@/modules/backend/products/components/project_details.vue"),
        meta: {
          layout: "auth",
        },
      },
    ],
  },
  // ========frontend==============
```

```
  // ========frontend==============
  {
    path: "/",
    name: "public",
    component: () => import("@/layouts/app/app_layout.vue"),
    children: [
      {
        path: "/login",
        name: "login",
        component: () => import("@/modules/auth/login_view.vue"),
      },
      {
        path: "/",
        name: "home",
        component: () => import("@/modules/frontend/home/home_view.vue"),
      },
      {
        path: "/:pathMatch(.*)*",
        name: "error-page",
        component: () => import("@/modules/frontend/home/home_view.vue"),
      },
    ],
  },
];

const router = createRouter({
  history: createWebHistory(process.env.BASE_URL),
  routes,
});

export default router;
```

Step 9: Data Table with no pagination

```vue
<!-- ==========Display data with dataTable======== -->
 <v-data-table-virtual
   v-if="products.length"
   :loading="productStore.isLoading"
   :headers="headers"
   :items="products"
   :sort-by="filter.sortBy"
   fixed-header
   fixed-footer
   :hover="true"
   :height="products.length > 10 ? '100vh' : '100vh'"
 >
   <!-- display category with unique colors -->
   <template v-slot:[`item.category`]="{ item }">
     <v-alert :color="getProductStatusClass(item.category)" height="15">
       {{ item.category }}
     </v-alert>
   </template>

   <!-- display actions -->
   <template v-slot:[`item.action`]="{ item }">
     <div class="d-flex justify-space-evenly">
       <v-btn
         icon="mdi:mdi-pencil"
         variant="text"
         density="compact"
         color="warning"
         @click="
           updateForm(item);
           dialog = true;
           btnName = 'Update';
         "
       ></v-btn>
       <v-btn
         icon="mdi:mdi-trash-can-outline"
         variant="text"
         density="compact"
         color="red"
         @click="deleteProducts(item.id)"
       ></v-btn>
       <router-link :to="{ name: 'product-details', params: { id: item.id } }">
         <v-btn
           icon="mdi:mdi-eye"
           variant="text"
           density="compact"
           color="green"
         ></v-btn>
       </router-link>
     </div>
   </template>
 </v-data-table-virtual>
 <!-- ============================================== -->
```

❖ Authentication using Laravel UI Package:

bash Copy code

```
composer require laravel/ui
php artisan ui bootstrap --auth
npm install && npm run dev
```

Generate a mailable to handle the custom email:

bash Copy code

```
php artisan make:mail ResetPasswordMail
```

Edit the `ResetPasswordMail` class in `app/Mail/ResetPasswordMail.php`:

```php
<?php

namespace App\Mail;

use Illuminate\Bus\Queueable;
use Illuminate\Mail\Mailable;
use Illuminate\Queue\SerializesModels;

class ResetPasswordMail extends Mailable
{
    use Queueable, SerializesModels;

    public $token;
    public $email;

    /**
     * Create a new message instance.
     *
     * @return void
     */
    public function __construct($token, $email)
    {
        $this->token = $token;
        $this->email = $email;
    }

    /**
     * Build the message.
     *
     * @return $this
     */
    public function build()
    {
        return $this->view('emails.reset_password')
            ->with([
                'token' => $this->token,
                'email' => $this->email,
            ]);
    }
}

```

Create the Blade template at `resources/views/emails/reset_password.blade.php`:

```blade
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Password Reset</title>
    <style>
        .email-container {
            font-family: Arial, sans-serif;
            color: #333;
            padding: 20px;
            border: 1px solid #ddd;
            border-radius: 5px;
            width: 600px;
            margin: 0 auto;
        }
        .email-header {
            text-align: center;
            margin-bottom: 20px;
        }
        .email-header img {
            width: 150px;
        }
        .email-body {
            margin-bottom: 20px;
        }
        .email-footer {
            text-align: center;
            color: #aaa;
            font-size: 12px;
        }
        .btn {
            background-color: #007BFF;
            color: white;
            padding: 10px 20px;
            text-decoration: none;
            border-radius: 5px;
            display: inline-block;
        }
    </style>
</head>
<body>
    <div class="email-container">
        <div class="email-header">
            <img src="{{ asset('images/company-logo.png') }}" alt="Company Logo">
        </div>
        <div class="email-body">
            <p>Hello!</p>
            <p>You are receiving this email because we received a password reset request for your account.
</p>
            <p>
                <a href="{{ url('password/reset', $token) }}?email={{ urlencode($email) }}" class="btn">
Reset Password</a>
            </p>
            <p>This password reset link will expire in 60 minutes.</p>
            <p>If you did not request a password reset, no further action is required.</p>
        </div>
        <div class="email-footer">
            Regards, <br>
            Your Company
        </div>
    </div>
</body>
</html>

```

```php
<?php
// ==========user models=========

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Foundation\Auth\User as Authenticatable;
use Illuminate\Notifications\Notifiable;
use Laravel\Sanctum\HasApiTokens;
use Illuminate\Support\Facades\Mail;
use App\Mail\ResetPasswordMail;

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

    /**
     * Override the sendPasswordResetNotification method to use custom mailable.
     *
     * @param string $token
     * @return void
     */
    public function sendPasswordResetNotification($token)
    {
        Mail::to($this->email)->send(new ResetPasswordMail($token, $this->email));
    }

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

    /**
     * The attributes that should be hidden for serialization.
     *
     * @var array<int, string>
     */
    protected $hidden = [
        'password',
        'remember_token',
    ];

    /**
     * The attributes that should be cast.
     *
     * @var array<string, string>
     */
    protected $casts = [
        'email_verified_at' => 'datetime',
        'password' => 'hashed',
    ];
}

```

Configure Mail Settings

Make sure your `.env` file has the correct mail configuration:

env Copy code

```
MAIL_MAILER=smtp
MAIL_HOST=smtp.mailtrap.io
MAIL_PORT=2525
MAIL_USERNAME=null
MAIL_PASSWORD=null
MAIL_ENCRYPTION=null
MAIL_FROM_ADDRESS="hello@example.com"
MAIL_FROM_NAME="${APP_NAME}"
```

Replace the placeholder values with your actual SMTP server details.

```
MAIL_MAILER=smtp
MAIL_HOST=smtp.gmail.com
MAIL_PORT=465
MAIL_USERNAME=nontoxicyubs@gmail.com
MAIL_PASSWORD=kjzqaaolfsvvutrn
MAIL_ENCRYPTION=tls
MAIL_FROM_ADDRESS=nontoxicyubs@gmail.com
MAIL_FROM_NAME="${APP_NAME}"
```

Get password from security<2-step-verification<App Passwords

❖ Middleware:

```
php artisan make:middleware AdminMiddleware
```

```php
 */
0 references | 0 overrides
public function handle(Request $request, Closure $next): Response
{
    if (!auth()->check())  {
        return redirect()->route('login')->with('error', 'Please log in to access this page.');
    }

    return $next($request);
}
```

```php
protected $middlewareAliases = [
    'auth' => \App\Http\Middleware\Authenticate::class,
    'auth.basic' => \Illuminate\Auth\Middleware\AuthenticateWithBasicAuth::class,
    'auth.session' => \Illuminate\Session\Middleware\AuthenticateSession::class,
    'cache.headers' => \Illuminate\Http\Middleware\SetCacheHeaders::class,
    'can' => \Illuminate\Auth\Middleware\Authorize::class,
    'guest' => \App\Http\Middleware\RedirectIfAuthenticated::class,
    'password.confirm' => \Illuminate\Auth\Middleware\RequirePassword::class,
    'precognitive' => \Illuminate\Foundation\Http\Middleware\HandlePrecognitiveRequests::class,
    'signed' => \App\Http\Middleware\ValidateSignature::class,
    'throttle' => \Illuminate\Routing\Middleware\ThrottleRequests::class,
    'verified' => \Illuminate\Auth\Middleware\EnsureEmailIsVerified::class,
    'auth.admin' => \App\Http\Middleware\AdminMiddleware::class,
];
```

web.php

```php
<?php

use Illuminate\Support\Facades\Auth;
use Illuminate\Support\Facades\Route;


// ==========frontend==========
Route::name('frontend.')->group(function () {
    require __DIR__ . '/frontend.php';
});
// ==========end of frontend========


// =======auth================
Auth::routes();
// =========end of auth========

// ========admin===============
Route::middleware(['auth.admin'])->group(function () {
    Route::prefix('admin')->group(function () {
        require __DIR__ . '/admin.php';
    });
});
// =========end of auth=========
```

admin.php

```php
<?php

use App\Http\Controllers\Admin\HomeController;
use Illuminate\Support\Facades\Route;

Route::resources([
    // 'blogs'=>BlogController::class,
]);

Route::get('/dashboard',[HomeController::class,'index'])->name('admin.dashboard');
```

frontend.php

```php
<?php

use App\Http\Controllers\Frontend\HomeController;
use Illuminate\Support\Facades\Route;

Route::get('/',[HomeController::class,'index'])->name('index');
```

https://amanj0314.medium.com/how-to-use-laravel-middleware-to-protect-your-application-1c76aa0acded

---

❖ Spatie user Roles & Permission management:

https://spatie.be/docs/laravel-permission/v6/installation-laravel

```
composer require spatie/laravel-permission
```

in your `config/app.php` file:

```
'providers' => [
    // ...
    Spatie\Permission\PermissionServiceProvider::class,
];
```

```
php artisan vendor:publish --provider="Spatie\Permission\PermissionServiceProvider"
```

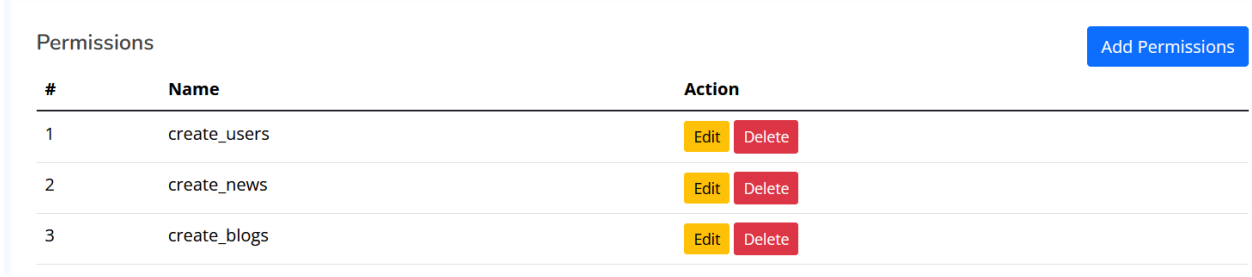
```
php artisan optimize:clear
# or
php artisan config:clear
```

```
php artisan migrate
```

Add the necessary trait to your User model:

```
// The User model requires this trait
use HasRoles;
```

Make Permission and Role Controller and do crud operation:



Roles

Add Roles

#	Name	Action
1	staff	Add/Edit Role Permission Edit Delete
2	user	Add/Edit Role Permission Edit Delete
3	admin	Add/Edit Role Permission Edit Delete
4	super_admin	Add/Edit Role Permission Edit Delete

Role: staff

Back

Permissions

- [ ] create_blogs
- [x] create_news
- [x] create_users

Submit

```php
Route::resources([
    'permissions'=>PermissionController::class,
    'roles'=>RoleController::class,
]);

Route::get('/roles/{roleId}/give-permissions',[RoleController::class,'addPermissionToRole'])->name('add.permissions.to.role');
Route::put('/roles/{roleId}/give-permissions',[RoleController::class,'givePermissionToRole'])->name('give.permissions.to.role');
```

```php
public function addPermissionToRole($roleId)
{
    try {
        $permissions = Permission::all();

        $role = Role::findOrFail($roleId);

        $rolePermissions = DB::table('role_has_permissions')
        ->where('role_has_permissions.role_id', $role->id)
        ->pluck('role_has_permissions.permission_id', 'role_has_permissions.permission_id')->all();


        return view('admin.roles.add-permissions', compact('permissions', 'role','rolePermissions'));
    } catch (\Throwable $th) {
        return back()->with('error', $th->getMessage());
    }
}

public function givePermissionToRole(Request $request,$roleId)
{
    try{
        $role = Role::findOrFail($roleId);
        $role->syncPermissions($request->permission);
        return redirect()->back()->with('success','Permission added to role');
    }catch(\Throwable $th){
        return back()->with('error',$th->getMessage());
    }
}
```

```blade
<form action="{{route('give.permissions.to.role',$role->id)}}" method="POST">
    @csrf
    @method('PUT')
    <div class="d-flex align-items-center" style="flex-wrap: wrap;gap:30px">
        @foreach ($permissions as $permission)
        <div class="form-group form-check">
            <input type="checkbox" class="form-check-input" id="{{ $permission->name }}" name="permission[]"
                value="{{ $permission->name }}" @checked(in_array($permission->id, $rolePermissions))
            >
            <label class="form-check-label" for="{{$permission->name}}">{{ $permission->name }}</label>
        </div>
        @endforeach

    </div>
    <button type="submit" class="btn btn-primary mt-3">Submit</button>
</form>
```

Make user controller and perform crud operation:

Users

Add User

#	Name	Email	Roles	Action
1	Saru	gocigicy@mailinator.com	admin staff	Edit Delete
2	Micah Sweeney	bedugolo@mailinator.com	staff	Edit Delete
3	Uma Swanson	varefo@mailinator.com	staff	Edit Delete
4	August Wilkins	bicozosivo@mailinator.com	super_admin staff	Edit Delete
5	Yubraj Koirala	yubrajkoirala7278@gmail.com	super_admin	Edit Delete

```blade
<td>
    {{-- get specific user roles --}}
    <div class="d-flex align-items-center" style="gap: 10px;flex-wrap:wrap">
        @if (count($user->getRoleNames())> 0)
            @foreach ($user->getRoleNames() as $role)
                <span class="badge rounded-pill text-bg-success">{{ $role }}</span>
            @endforeach
        @endif
    </div>
</td>
```

Edit Users

Back

Name

Saru

Email

gocigicy@mailinator.com

Password

Roles

× admin × staff

Submit

```
<select class="form-select" id="select-roles" data-placeholder="Choose Roles" multiple name="roles[]">
    @if (count($roles) > 0)
        @foreach ($roles as $role)
            <option value="{{ $role }}"
                {{ in_array($role, $userRoles) || in_array($role, old('roles', [])) ? 'selected' : '' }}>
                {{ $role }}
            </option>
        @endforeach
    @endif
</select>
```

```php
public function index()
    {
        try {
            $users = User::latest()->get();
            return view('admin.users.index', compact('users'));
        } catch (\Throwable $th) {
            return back()->with('error', $th->getMessage());
        }
    }

    public function create()
    {
        try {
            $roles = Role::pluck('name', 'name')->all();
            return view('admin.users.create', compact('roles'));
        } catch (\Throwable $th) {
            return back()->with('error', $th->getMessage());
        }
    }

    public function store(UserRequest $request)
    {
        try {
            $user = User::create([
                'name' => $request->name,
                'email' => $request->email,
                'password' => Hash::make($request->password),
            ]);

            $user->syncRoles($request->roles);

            return redirect()->route('users.index')->with('success', 'User added successfully!');
        } catch (\Throwable $th) {
            return back()->with('error', $th->getMessage());
        }
    }
```

```php
public function edit(User $user)
    {
        try {
            $roles = Role::pluck('name', 'name')->all();
            $userRoles = $user->roles->pluck('name', 'name')->all();
            return view('admin.users.edit', compact('user', 'roles', 'userRoles'));
        } catch (\Throwable $th) {
            return back()->with('error', $th->getMessage());
        }
    }

    public function update(UserRequest $request, User $user)
    {
        try {
            $data = [
                'name' => $request->name,
                'email' => $request->email,
            ];

            if (!empty($request->password)) {
                $data += [
                    'password' => Hash::make($request->password),
                ];
            }

            $user->update($data);

            $user->syncRoles($request->roles);

            return redirect()->route('users.index')->with('success', 'User updated successfully!');
        } catch (\Throwable $th) {
            return back()->with('error', $th->getMessage());
        }
    }

    public function destroy(User $user)
    {
        try {
            $user->delete();
            return back()->with('success', 'User deleted successfully!');
        } catch (\Throwable $th) {
            return back()->with('error', $th->getMessage());
        }
    }
```

Now, for roles and permissions:
Step1:
https://spatie.be/docs/laravel-permission/v6/basic-usage/middleware

In Laravel 9 and 10 you can add them in `app/Http/Kernel.php` :

```php
// Laravel 9 uses $routeMiddleware = [
//protected $routeMiddleware = [
// Laravel 10+ uses $middlewareAliases = [
protected $middlewareAliases = [
    // ...
    'role' => \Spatie\Permission\Middleware\RoleMiddleware::class,
    'permission' => \Spatie\Permission\Middleware\PermissionMiddleware::class,
    'role_or_permission' => \Spatie\Permission\Middleware\RoleOrPermissionMiddleware::class,
];
```

Step2:

In admin.php/web.php

```php
// roles and permission:
Route::group(['middleware' => ['role:super_admin|admin']], function () {
    // super_admin and admin can only access this routes
    Route::resources([
        'users' => UserController::class
    ]);
    // super_admin and admin with permission "add project" can only access this routes
    Route::get('/store/project', [UserController::class, 'addProject'])->name('add.project')
        ->middleware('permission:add project');
});
```

Roles and permission in resource controller: Step 1:

```php
// roles and permission:
Route::group(['middleware' => ['role:super_admin|admin']], function () {
    /* super_admin and admin can only access user controller.*/
    Route::resources([
        'users' => UserController::class
    ]);
});
```

Step 2: (user controller)

```php
0 references | 0 overrides
public function __construct() {
    // permission with "edit user" can only access edit and update routes of user controller
    $this->middleware('permission:edit user',['only'=>['edit','update']]);
    // permission with "delete user" can only access destroy route of user controller
    $this->middleware('permission:delete user',['only'=>['destroy']]);
}
```

Standard way to write the above roles and permission codes:

Step 1: Kernel.php

```php
'auth.admin' => \App\Http\Middleware\AdminMiddleware::class,
```

Step 2: admin.php

```php
// roles and permission:
Route::group(['middleware' => ['auth.admin']], function () {
    /* super_admin and admin can only access user controller.*/
    Route::resources([
        'users' => UserController::class
    ]);
});
```

Step 3: AdminMiddleware.php

```php
public function handle(Request $request, Closure $next): Response
{
    if (Auth::check()) {
        $user = Auth::user();
        if ($user->hasRole(['super_admin', 'admin'])) {
            return $next($request);
        }
        Auth::logout();
        return redirect()->route('login')->with('error', "You have been logged out because you don't
                have permission to access this page.");
        // abort(403, "User doesn't have permission to access this page!");
    }
    return redirect()->route('login')->with('error', 'Please log in to access this page.');
}
```

Step 4: Create RoleSeeder and AdminSeeder:

```php
public function run(): void
{
    Role::create([
        'name'=>'super_admin',
    ]);
}
```

```php
public function run(): void
{
    $user=User::create([
        'name' => 'Yubraj Koirala',
        'email' => 'yubrajkoirala7278@gmail.com',
        'password' => Hash::make('12345678')
    ]);
    $user->syncRoles(['super_admin']);
}
```

Morph Many (Comments):

1. Make comment migration and model
2. In CommentService.php

```php
<?php

namespace App\Service;

use App\Models\Comment;

class CommentService{
    // Add
    public function addService($model,$comments=[]){
        if(!empty($comments)){
            $commentArr=[];
            foreach ($comments as $key => $comment) {
                $commentModel=new Comment();
                $commentModel->model_type=$model;
                $commentModel->model_id=$model->id;
                $commentModel->comments=$comment;
                $commentArr[]=$commentModel;
            }
            $model->comments()->saveMany($commentArr);
        }
        return;
    }
    // delete
    public function deleteComments($comments){
       foreach ($comments as $key => $comment) {
        $comment->delete();
       }
    }

    // update
    public function updateComment($model,$comments){
       $this->deleteComments($model->comments);
       $this->addService($model,$comments);
    }


}
```

3. Comment Migration:

```php
public function up(): void
{
    Schema::create('comments', function (Blueprint $table) {
        $table->id();
        $table->bigInteger('model_id');
        $table->string('model_type');
        $table->longText('comments');
        $table->timestamps();
    });
}
```

4. In any model you want to use

```php
public function comments(){
    return $this->morphMany(Comment::class,'model');
}
```

MorphOne Image:

composer require intervention/image

```
public function up(): void
{
    Schema::create('images', function (Blueprint $table) {
        $table->id();
        $table->bigInteger('model_id');
        $table->string('model_type');
        $table->string('location');
        $table->string('filename');
        $table->timestamps();
    });
}
```

```
<?php

namespace App\Service;

use App\Models\Image;
use Illuminate\Support\Facades\Storage;
use Intervention\Image\ImageManager;
use Illuminate\Support\Str;

class ImageService
{
    // Method to add image to a model
    public function saveImage($model, $image, $directory)
    {
        // Check if there are images to save
        if (!empty($image)) {
            $directory = 'public/images/' . $directory;
            // Create directory if it doesn't exist
            $this->makeDirectory($directory);
            // Generate unique filename
            $imageNames = $this->makeName($image);
            // Move and resize image
            $this->moveImage($image, $directory, $imageNames['saveName'], 1024,
720);
            // Create a new Image model instance
            $imageModel = new Image();
            // Assign model type (e.g., User, Product, etc.)
            $imageModel->model_type = $model;
            // Assign model ID
            $imageModel->model_id = $model->id;
            // Assign filename
            $imageModel->filename = $imageNames['saveName'];
            // Assign location
            $imageModel->location = $directory;
```

```
            // Save the image in the database
            $model->image()->save($imageModel);
        }

        return; // Return nothing
    }

    // Method to create a directory if it doesn't exist
    private function makeDirectory($path)
    {
        if (!Storage::exists($path)) {
            Storage::makeDirectory($path);
        }

        return; // Return nothing
    }

    // Method to generate a unique filename for an image
    public function makeName($image, $type = null)
    {
        // Get the original filename
        $originalName = Str::replace(' ', '-', trim($image-
>getClientOriginalName()));
        // Limit the filename length
        $NameWithoutExtension = Str::limit(pathinfo($originalName,
PATHINFO_FILENAME), 200);
        // Generate a unique save filename
        $saveName = time() . '-' . $NameWithoutExtension . '.' . $image-
>getClientOriginalExtension();
        return [
            'saveName' => $saveName, // Return the unique filename
            'originalName' => $originalName, // Return the original filename
        ];
    }

    // Method to move and resize an image
    public function moveImage($image, $directory, $saveName, $width = 544, $height
= 356)
    {
        // Read and resize the image
        $image = ImageManager::gd()->read($image);
        $image = $image->resize($width, $height);
        $image = $image->toWebp(70);
        // Store the image in the specified directory
        return Storage::put($directory . '/' . $saveName, $image, [
            'visibility' => 'public',
        ]);
    }
```

```php
    // Method to delete image
    public function deleteImage($image)
    {
        // Delete image from the database
        $image->delete();
        // Delete image from local storage
        $path = $image->location . '/' . $image->filename;
        Storage::delete($path);
    }

      // Method to update images for a model
      public function updateImage($model, $image, $directory, $deletePrevious =
false)
      {
          // Check if there are new images to update
          if (!empty($image)) {
              // If deletePrevious flag is set to true, delete previous images
              if ($deletePrevious) {
                  $this->deleteImage($model->image); // Delete previous images
              }
              // Make the directory if it doesn't exist
              $this->makeDirectory($directory);
              // Save new images
              $this->saveImage($model, $image,  $directory);
          }
      }
}
```

In BlogService:

```php
// =======POST==============
public function addService($request){
    // adding data to db
    $blog = Blog::create($request);
    // if image exist
    if (isset($request['image'])) {
        $this->imageservice->saveImage($blog, $request['image'], 'blog');
    }
}
```

```php
// ===============UPDATE=====================
public function updateService($request, $blog)
{
    if (!empty($request['image'])) {
        $this->imageService->updateImages($blog, $request['image'], 'blog', true);
    }
    $blog->update([
        'title' => $request['title'],
        'slug' => $request['slug'],
        'description' => $request['description']
    ]);
}
```

In any model:

```php
public function image(){
    return $this->morphOne(Image::class,'model');
}
```

MorphMany Images:

```php
<?php

namespace App\Service;

use App\Models\Image;
use Illuminate\Support\Str;
use Intervention\Image\ImageManager;
use Illuminate\Support\Facades\Storage;

class ImageService
{

    // Method to add images to a model
    public function addImages($model, $images = [], $directory)
    {
        // Save images to the specified directory
        $this->saveImages($model, $images, 'public/images/' . $directory);
        return; // Return nothing
    }

    // Method to save images to storage
    private function saveImages($model, $images, $directory)
    {
        if (!empty($images)) { // Check if there are images to save
            $imageArr = []; // Array to store image objects
            $this->makeDirectory($directory); // Create directory if it doesn't
exist
            foreach ($images as $image) {
                $imageNames = $this->makeName($image); // Generate unique filename
                $this->moveImage($image, $directory, $imageNames['saveName'], 1024,
720); // Move and resize image
                // Create a new Image model instance
                $imageModel = new Image();
                // Assign model type (e.g., User, Product, etc.)
                $imageModel->model_type = $model;
                // Assign model ID
                $imageModel->model_id = $model->id;
                // Assign filename
                $imageModel->filename = $imageNames['saveName'];
                // Assign location
                $imageModel->location = $directory;
                // Add image to the array
                $imageArr[] = $imageModel;
            }
            // Save images in the database
            $model->image()->saveMany($imageArr);
        }
```

```
        return; // Return nothing
    }

    // Method to move and resize an image
    public function moveImage($image, $directory, $saveName, $width = 544, $height
= 356)
    {
        // Read and resize the image
        $image = ImageManager::gd()->read($image);
        $image = $image->resize($width, $height);
        $image = $image->toWebp(70);
        // Store the image in the specified directory
        return Storage::put($directory . '/' . $saveName, $image, [
            'visibility' => 'public',
        ]);
    }

    // Method to generate a unique filename for an image
    public function makeName($image, $type = null)
    {
        // Get the original filename
        $originalName = Str::replace(' ', '-', trim($image-
>getClientOriginalName()));
        // Limit the filename length
        $NameWithoutExtension = Str::limit(pathinfo($originalName,
PATHINFO_FILENAME), 200);
        // Generate a unique save filename
        $saveName = time() . '-' . $NameWithoutExtension . '.' . $image-
>getClientOriginalExtension();
        return [
            'saveName' => $saveName, // Return the unique filename
            'originalName' => $originalName, // Return the original filename
        ];
    }

    // Method to update images for a model
    public function updateImages($model, $images, $directory, $deletePrevious =
false)
    {
        // Check if there are new images to update
        if (!empty($images)) {
            // If deletePrevious flag is set to true, delete previous images
            if ($deletePrevious) {
                $this->deleteImages($model->image); // Delete previous images
            }
            // Make the directory if it doesn't exist
            $this->makeDirectory($directory);
            // Save new images
            $this->saveImages($model, $images, 'public/images/' . $directory);
```

```
        }
    }

    // Method to delete images
    public function deleteImages($images)
    {
        foreach ($images as $image) {
            // Delete image from the database
            $image->delete();
            // Delete image from local storage
            $path = $image->location . '/' . $image->filename;
            Storage::delete($path);
        }
    }

    // Method to create a directory if it doesn't exist
    private function makeDirectory($path)
    {
        if (!Storage::exists($path)) {
            Storage::makeDirectory($path);
        }

        return; // Return nothing
    }
}
```

Laravel Module Package:

Used to manage and organize big projects. Example: School Management System has Student,Teacher and Fees modules.

https://nwidart.com/laravel-modules/v6/installation-and-setup

a. composer require nwidart/laravel-modules
b. php artisan vendor:publish --provider="Nwidart\Modules\LaravelModulesServiceProvider"
c. In composer.json

d. composer dump-autoload
e. Now create a module which have its own Controller, Models, Views and Routes.

1. Create the Module

First, create your module if you haven't already. Use the `**module:make**` command to generate the basic structure of your module:

bash Copy code

```
php artisan module:make Blogs
```

This command will create the necessary directories and files for your `**Blogs**` module under `**Modules/Blogs**`.

2. Create Models and Migrations

Example: Creating a `Post` Model with Migration

bash Copy code

```
php artisan module:make-model Post --migration Blogs
```

This command will create:

- The `**Post**` model (`**Modules/Blogs/Entities/Post.php**`)
- The migration file (`**Modules/Blogs/Database/Migrations/xxxx_xx_xx_xxxxxx_create_posts_table.php**`)

3. Create Controllers

Example: Creating Controllers

```bash
php artisan module:make-controller PostController Blogs
php artisan module:make-controller CommentController Blogs
```

This will create:

- `Modules/Blogs/Http/Controllers/PostController.php`
- `Modules/Blogs/Http/Controllers/CommentController.php`

4. Define Routes

Next, define routes for your controllers in the `routes/web.php` file within your module (`Modules/Blogs/Routes/web.php`).

Example: Route Definitions

```php
<?php

use Illuminate\Support\Facades\Route;
use Modules\Blogs\Http\Controllers\PostController;
use Modules\Blogs\Http\Controllers\CommentController;

Route::group(['prefix' => 'blogs', 'namespace' => 'Modules\Blogs\Http\Controllers'], funct
    Route::get('/posts', [PostController::class, 'index'])->name('blogs.posts.index');
    Route::post('/posts', [PostController::class, 'store'])->name('blogs.posts.store');
    Route::get('/comments', [CommentController::class, 'index'])->name('blogs.comments.ind
    Route::post('/comments', [CommentController::class, 'store'])->name('blogs.comments.st
    // Add more routes as needed
});
```

5. Run Migrations

After creating your migrations, migrate the database to create the necessary tables for your models:

```bash
php artisan module:migrate Blogs
```

This command will execute the migrations located in your `Blogs` module.

Make request for validation in specific module:

```
php artisan module:make-request CreatePostRequest Blogs
```

Laravel Repository Design Pattern:

1. Create app/Repositories/Interfaces/CategoryRepositoryInterface.php file

```php
<?php
namespace App\Repositories\Interfaces;

Interface CategoryRepositoryInterface{

    public function all();

}
```

2. Create app/Repositories/CategoryRepository.php

```php
<?php
namespace App\Repositories;
use App\Repositories\Interfaces\CategoryRepositoryInterface;
use App\Models\Category;

class CategoryRepository implements CategoryRepositoryInterface{

    public function all(){
        return Category::all();
    }
}
```

3. Inside CategoryController.php

```php
private $categoryRepository;

public function __construct(CategoryRepositoryInterface $categoryRepository){
    $this->categoryRepository = $categoryRepository;
}

public function index(){

    $categories = $this->categoryRepository->all();


    return view('categories.list',['categories' => $categories]);
}
```

4. Inside AppServiceProvider.php

```php
public function register()
{
    $this->app->bind(CategoryRepositoryInterface::class,CategoryRepository::class);
}
```

Event & Listener in Laravel:

Laravel's events provide a simple observer pattern implementation, allowing you to subscribe and listen for various events that occur within your application.

Task: If any user create a blog then send email to all the users that blog has been created and notify the admin that blog has been created. Here, When Blog is created (event occurred) then listener will listen the event and execute automatically.

1. Create Event:

```
php artisan make:event BlogCreated
```

2. Inside your controller

```
public function store(BlogRequest $request)
{
    try {
        $dataToStore = $request->only('title', 'slug', 'description', 'image', 'is_published');
        $blog = $this->blogRepository->store($dataToStore);
        $data = ['title' => $blog['title'], 'author' => auth()->user()->name, 'email' => auth()->user()->email];
        event(new BlogCreated($data));
        return redirect()->route('admin.blog')->with('success', 'Blog added successfully');
    } catch (\Throwable $th) {
        return back()->with('error', $th->getMessage());
    }
}
```

3. Inside BlogCreated.php

```
public $blog;

/**
 * Create a new event instance.
 */
public function __construct($blog)
{
    $this->blog=$blog;
}
```

4. Make Listener NotifyUser(any name) to listen the BlogCreated event.

```
php artisan make:listener NotifyUser --event=BlogCreated
```

```
public function handle(BlogCreated $event): void
{
    $users=User::get();
    foreach ($users as $user) {
        Mail::to($user->email)->queue(new UserMail($event->blog));
    }
}
```

5. Inside EventServiceProvider.php

```php
protected $listen = [
    Registered::class => [
        SendEmailVerificationNotification::class,
    ],
    BlogCreated::class=>[
        NotifyUser::class
    ]
];
```

6. Make mail to send mail:

```
php artisan make:mail UserMail
```

```php
public function __construct($blog)
{
   $this->blog=$blog;
}
```

```php
public function content(): Content
{
    return new Content(
        markdown: 'notify_user',
    );
}
```

Inside notify_user.blade.php

```
@component('mail::message')
# Title: {{ $blog['title'] }}
# Email: {{ $blog['email'] }}
# Name: {{ $blog['author'] }}

Thanks,<br>
{{ config('app.name') }}
@endcomponent
```

Attach Image:

```
<img src="{{ $message->embed(public_path('assets/img/logo.png')) }}">
```

7. Queues allow you to defer time-consuming tasks, such as sending emails or generating reports, to be processed in the background, freeing up your application to handle other requests.

Php artisan queue:table

in .env

```
QUEUE_CONNECTION=database
```

Inside **UserMail.php**

```
class UserMail extends Mailable implements ShouldQueue
```

php artisan migrate

```
php artisan queue:listen
```

Pusher For Real Time:

Official docs:
https://medium.com/@parthpatel0516/laravel-real-time-notifications-with-pusher-3ac72531e15

1. Install Pusher

```
composer require pusher/pusher-php-server
```

```
npm install --save laravel-echo pusher-js
```

2. Uncomment below code from `config/app.php`

```
App\Providers\BroadcastServiceProvider::class,
```

3. In .env

```
BROADCAST_DRIVER=pusher
```

4. Uncomment below code from `resources/js/bootstrap.js`

```
import Echo from 'laravel-echo';

import Pusher from 'pusher-js';
window.Pusher = Pusher;

window.Echo = new Echo({
    broadcaster: 'pusher',
    key: import.meta.env.VITE_PUSHER_APP_KEY,
    cluster: import.meta.env.VITE_PUSHER_APP_CLUSTER ?? 'mt1',
    wsHost: import.meta.env.VITE_PUSHER_HOST ?? `ws-${import.meta.env.VITE_PUSHER_APP_CLUSTER}.pusher.com`,
    wsPort: import.meta.env.VITE_PUSHER_PORT ?? 80,
    wssPort: import.meta.env.VITE_PUSHER_PORT ?? 443,
    forceTLS: (import.meta.env.VITE_PUSHER_SCHEME ?? 'https') === 'https',
    enabledTransports: ['ws', 'wss'],
});
```

5. Get all the App Keys from https://dashboard.pusher.com/

```
PUSHER_APP_ID=1823897
PUSHER_APP_KEY=aa0cad2629660de0e17f
PUSHER_APP_SECRET=3ea6c7a64c232d423c42
PUSHER_APP_CLUSTER=mt1
PUSHER_HOST=
PUSHER_PORT=443
PUSHER_SCHEME=https
```

6. **Inside broadcasting.php**

```php
'pusher' => [
    'driver' => 'pusher',
    'key' => env('PUSHER_APP_KEY'),
    'secret' => env('PUSHER_APP_SECRET'),
    'app_id' => env('PUSHER_APP_ID'),
    'options' => [
        'cluster' => env('PUSHER_APP_CLUSTER'),
        'host' => env('PUSHER_HOST') ?: 'api-'.env('PUSHER_APP_CLUSTER', 'mt1').'.pusher.com',
        'port' => env('PUSHER_PORT', 443),
        'scheme' => env('PUSHER_SCHEME', 'https'),
        'encrypted' => true,
        'useTLS' => env('PUSHER_SCHEME', 'https') === 'https',
    ],
    'client_options' => [
        // Guzzle client options: https://docs.guzzlephp.org/en/stable/request-options.html
    ],
],
```

7. Create an event

```
php artisan make:event BlogCreated
```

8. Inside call the event from your controller:

```php
public function store(BlogRequest $request)
{
    try {
        $dataToStore = $request->only('title', 'slug', 'description', 'image', 'is_published');
        $blog = $this->blogRepository->store($dataToStore);
        $data = ['title' => $blog['title'], 'author' => auth()->user()->name, 'email' => auth()->user()->email];
        event(new BlogCreated($data));
        return redirect()->route('admin.blog')->with('success', 'Blog added successfully');
    } catch (\Throwable $th) {
        return back()->with('error', $th->getMessage());
    }
}
```

9. Inside your event:

```php
class BlogCreated implements ShouldBroadcast
{
    public $blog;

    /**
     * Create a new event instance.
     */
    public function __construct($blog)
    {
        $this->blog=$blog;
    }
```

```php
    public function broadcastOn()
    {
        return new Channel('my-channel');
    }

    public function broadcastAs() {
        return 'form-submitted';
    }
```

10. Inside any blade file where you want notification to be displayed:
->**You can get the below code from pusher dashboard in Getting Started.**

```
<script src="https://js.pusher.com/8.2.0/pusher.min.js"></script>
<script>

  // Enable pusher logging - don't include this in production
  Pusher.logToConsole = true;

  var pusher = new Pusher('aa0cad2629660de0e17f', {
    cluster: 'mt1'
  });

  var channel = pusher.subscribe('my-channel');
  channel.bind('my-event', function(data) {
    alert(JSON.stringify(data));
  });
</script>
```

Custom

```
{{-- pusher --}}
<script>
    // Enable pusher logging - don't include this in production
    Pusher.logToConsole = true;

    var pusher = new Pusher('aa0cad2629660de0e17f', {
        cluster: 'mt1'
    });

    var channel = pusher.subscribe('my-channel');
    channel.bind('form-submitted', function(data) {
        console.log(data);
        toastify().success(`${data.blog.subject} by ${data.blog.author}`);
    });
</script>
```

11. **php artisan queue:listen**

Real Time Bell Icon Notification Using Pusher & Livewire:

1. Follow upto step 6 from above setup.
2. Create an event : php artisan make:event:CommissionEarned and notification php artisan make:notification CommissionEarnedNotification.
3. Call event and notification from controller/livewire component.

Send Notification to only auth user:

```php
// send notification
Auth::user()->notify(new CommissionEarnedNotification($commission));
Auth::user()->notifications()->latest()->first();
event(new CommissionEarned($commission));
```

4. Inside CommissionEarned event:

```php
class CommissionEarned implements ShouldBroadcast
{
    use Dispatchable, InteractsWithSockets, SerializesModels;
    public $commission;

    /**
     * Create a new event instance.
     */
    public function __construct($commission)
    {
        $this->commission=$commission;
    }

    /**
     * Get the channels the event should broadcast on.
     *
     * @return array<int, \Illuminate\Broadcasting\Channel>
     */
    public function broadcastOn()
    {
       return new Channel('tickets');
    }

    public function broadcastAs(){
        return 'ticket-created';
    }
}
```

5. Inside CommissionEarnedNotification.php

```php
public $commission;

/**
 * Create a new notification instance.
 */
public function __construct($commission)
{
    $this->commission=$commission;
}
```

```php
/**
 * Get the notification's delivery channels.
 *
 * @return array<int, string>
 */
public function via($notifiable)
{
    return ['database', 'broadcast'];
}
```

```php
public function toArray($notifiable)
{
    return [
        'name' => Auth::user()->name,
        'earned_money'=>$this->commission->commission,
        'id'=>Auth::user()->id,
    ];
}
```

```php
public function toBroadcast($notifiable)
{
    return new BroadcastMessage([
        'name' => Auth::user()->name,
        'earned_money'=>$this->commission->commission,
        'id'=>Auth::user()->id
    ]);
}
```

6. Inside master.blade.php

```blade
{{-- notifications --}}
<li class="nav-item dropdown dropdown-large">
    @livewire('admin.notification.bell-notification')
</li>
{{-- end of notifications --}}
```

7. Create BellNotification.php livewire component.
8. Inside BellNotification.php

```php
class BellNotification extends Component
{
    public $notifications;
    public $unreadCount;
    private $notificationRepository;


    protected $listeners = ['notificationReceived' => 'fetchNotifications'];

    public function boot(NotificationRepositoryInterface $notificationRepository)
    {
        $this->notificationRepository = $notificationRepository;
    }

    public function mount()
    {
        $this->fetchNotifications();
    }

    public function fetchNotifications()
    {
        $notificationInfo = $this->notificationRepository->fetchNotifications(10);
        $this->notifications = $notificationInfo['notifications'];
        $this->unreadCount = $notificationInfo['unreadCount'];
    }

    public function markAsRead($notificationId)
    {
        $notification = $this->notificationRepository->markAsRead($notificationId);
        if ($notification) {
            $notification->markAsRead();
            $this->fetchNotifications();
        }
    }

    public function render()
    {
        return view('livewire.admin.notification.bell-notification');
    }
}
```

9. Inside NotificationRepository.php:

```php
class NotificationRepository implements NotificationRepositoryInterface
{
    public function fetchNotifications($totalNotification)
    {
        $user = Auth::user();
        if($totalNotification=='all'){
            $notifications = $user->notifications()->latest()->get();
        }else{
            $notifications = $user->notifications()->latest()->take($totalNotification)->get();
        }
        $unreadCount = $user->unreadNotifications->count();
        return [
            'notifications'=>$notifications,
            'unreadCount'=>$unreadCount
        ];
    }
    public function markAsRead($notificationId){
        return Auth::user()->notifications()->find($notificationId);
    }
}
```

10. Inside bell-notification.blade.php

```blade
<script src="https://js.pusher.com/8.2.0/pusher.min.js"></script>
<script>
    document.addEventListener('DOMContentLoaded', function() {
        Pusher.logToConsole = true;

        var pusher = new Pusher('aa0cad2629660de0e17f', {
            cluster: 'mt1',
            encrypted: false,
            forceTLS: false
        });

        var channel = pusher.subscribe('tickets');
        channel.bind('ticket-created', function(data) {
            console.log('i am here');
            @this.call('fetchNotifications');
            console.log(data.ticket.id);
        });
    });

    function handleNotificationClick(notificationId, url) {
        @this.call('markAsRead', notificationId).then(() => {
            window.location.href = url;
        });
    }
    function handleNotification(notificationId){
        @this.call('markAsRead', notificationId)
    }
</script>
```

```blade
<div class="header-notifications-list">
    @foreach ($notifications as $notification)
        @if ($notification->type == 'App\Notifications\TicketCreatedNotification')
            @php
                $created_at = \Illuminate\Support\Carbon::parse($notification->created_at);
                $time_ago = $created_at->diffForHumans();
            @endphp
            <a class="dropdown-item {{ $notification->read_at ? '' : 'unread' }}" href="#"
                onclick="handleNotificationClick('{{ $notification->id }}', '{{ route('admin.support.chat', $notification->data['id']) }}')">
                <div class="d-flex align-items-center">
                    <div class="user-online">
                        <img src="{{ asset($notification->data['image'] ? $notification->data['image'] : 'assets/img/avatars/Asset 20.png') }}"
                            class="msg-avatar" alt="user avatar">
                    </div>
                    <div class="flex-grow-1">
                        <h6 class="msg-name">
                            {{ Auth::user()->hasRole('super_admin') ? 'Ticket number: ' . $notification->data['ticket_number'] . ' (opened)' : 'Ticket number: '
                            <span class="msg-time float-end">{{ $time_ago }}</span>
                        </h6>
                        <p class="msg-info">
                            {{ Auth::user()->hasRole('super_admin') ? $notification->data['user'] : 'Admin responds to your ticket.' }}
                        </p>
                    </div>
                </div>
            </a>
        @endif
```

Bell Icon Notification Using Pusher and AJAX:

1. Follow upto step 6 from above setup.
2. Create new event: php artisan make:event BlogCreated and notification php artisan make:notification PostCreatedNotification.
3. Inside your method where you want to send notification:

```php
public function store(PostRequest $request)
{
    try {
        $post = $this->postService->addService($request->validated());
        // Notify all users about the new post (except the one creating the post)
        $users = User::where('id', '!=', Auth::user()->id)->get();
        foreach ($users as $user) {
            $user->notify(new PostCreatedNotification($post));
        }
        event(new BlogCreated($post));
        return redirect()->route('posts.index')->with('success', 'Post added successfully!');
    } catch (\Throwable $th) {
        return back()->with('error', $th->getMessage());
    }
}
```

4. Inside BlogCreated event:

```php
class BlogCreated implements ShouldBroadcast
{
    use Dispatchable, InteractsWithSockets, SerializesModels;

    public $blog;


    /**
     * Create a new event instance.
     */
    public function __construct($blog)
    {
        $this->blog=$blog;
    }

    /**
     * Get the channels the event should broadcast on.
     *
     * @return array<int, \Illuminate\Broadcasting\Channel>
     */
    public function broadcastOn()
    {
       return new Channel('my-channel');
    }

    public function broadcastAs(){
        return 'form-submitted';
    }

}
```

5. Inside PostCreatedNotification.php:

```php
class PostCreatedNotification extends Notification
{
    use Queueable;

    public $post;

    public function __construct(Post $post)
    {
        $this->post = $post;
    }

    public function via($notifiable)
    {
        return ['database', 'broadcast'];
    }

    public function toArray($notifiable)
    {
        return [
            'id' => $this->id,
            'post_id' => $this->post->id,
            'title' => $this->post->title,
            'user' => $this->post->user->name,
        ];
    }
}
```

6. Inside master.blade.php

```blade
{{-- notification --}}
<li class="nav-item notification-item pe-3 position-relative" data-bs-toggle="modal"
    data-bs-target="#staticBackdrop">
    <i class="fa-solid fa-bell fs-4 text-dark"></i>
    <span class="notification-badge bg-danger text-white rounded-circle" id="notification-badge">
        {{ auth()->user()->unreadNotifications->count() }}
    </span>
</li>
{{-- end notification --}}
```

```blade
<div class="modal-body notification-list">
    @foreach (auth()->user()->notifications as $notification)
        <form action="{{ route('admin.notification.markAsRead', $notification->id) }}"
            method="POST">
            @csrf
            <button class="notification-item d-flex align-items-start mb-3 w-100 border-0"
                style="background-color: {{ $notification->read_at ? '#ffffff' : '#daf1f9' }};">
                <img src="{{ asset('frontend/images/mahesh.jpg') }}" alt="User"
                    class="rounded-circle me-3">
                <div class="notification-content">
                    <h6 class="mb-1">{{ $notification->data['title'] }}</h6>
                    <p class="mb-0 text-muted">New post by {{ $notification->data['user'] }}</p>
                </div>
                <span
                    class="text-muted small ms-auto">{{ $notification->created_at->diffForHumans() }}</span>
            </button>
        </form>
    @endforeach
</div>
<div class="modal-footer">
    <button type="button" class="btn btn-primary w-100">View All Notifications</button>
</div>
```

```blade
{{-- =======notification========= --}}
{{-- for human readable time --}}
<script src="https://cdnjs.cloudflare.com/ajax/libs/moment.js/2.29.4/moment.min.js"></script>
{{-- pusher link --}}
<script src="https://js.pusher.com/8.2.0/pusher.min.js"></script>
<script>
    // Enable pusher logging
    Pusher.logToConsole = true;

    var pusher = new Pusher('38806878aa78497c43a3', {
        cluster: 'mt1'
    });

    var creatorId = "{{ auth()->id() }}";
    var channel = pusher.subscribe('my-channel');
    channel.bind('form-submitted', function(data) {
        // Fetch the latest notification and append it
        fetchLatestNotification();
        // Update the notification count
        updateNotificationCount();
    });
```

```blade
function fetchLatestNotification() {
    // Use AJAX to fetch the latest notification
    fetch('/latest-notification')
        .then(response => response.json())
        .then(notification => {
            console.log(notification);
            var notificationHtml = `
          <form action="/admin/notifications/${notification.id}/mark-as-read" method="POST">
                @csrf
                <button class="notification-item d-flex align-items-start mb-3 w-100 border-0" style="background-color: ${notification.read_at ? '#ffffff' : '#daf1f9'};">
                    <img src="{{ asset('frontend/images/mahesh.jpg') }}" alt="User" class="rounded-circle me-3">
                    <div class="notification-content">
                        <h6 class="mb-1">${notification.data.title}</h6>
                        <p class="mb-0 text-muted">New post by ${notification.data.user}</p>
                    </div>
                    <span class="text-muted small ms-auto">${moment(notification.created_at).fromNow()}</span>
                </button>
            </form>
        `;

            // Append the new notification to the notification list
            document.querySelector('.notification-list').insertAdjacentHTML('afterbegin', notificationHtml);

        })
        .catch(error => console.error('Error fetching notification:', error));
}
```

```blade
function updateNotificationCount() {
    // Use AJAX to get the unread notification count
    fetch('/unread-notification-count')
        .then(response => response.json())
        .then(count => {
            // Update the badge with the new count
            console.log(count);
            document.getElementById('notification-badge').innerText = count;
        })
        .catch(error => console.error('Error fetching unread count:', error));
}
```

7. Inside web.php

```php
Route::get('/latest-notification', function () {
    return auth()->user()->notifications()->latest()->first();
});

Route::get('/unread-notification-count', function () {
    return auth()->user()->unreadNotifications->count();
});
```

8. Inside NotificationController.php

```php
public function markAsRead($id)
{
    // mark notification as read
    $notification = auth()->user()->notifications()->findOrFail($id);
    $notification->markAsRead();

    return redirect()->route('admin.notification');
}
```

Livewire:

1. Follow below docs:
   https://livewire.laravel.com/docs/quickstart

2. Life cycle hooks in livewire:
   https://laravel-livewire.com/docs/2.x/lifecycle-hooks

Twilio (Send msg in mobile):

In the context of a Laravel application, Twilio is often used to implement features like SMS messaging, voice calls, and two-factor authentication in mobile phone.
Read the docs and do by yourself:
https://fajarwz.com/blog/sending-sms-in-laravel-using-twilio-a-step-by-step-tutorial/
https://medium.com/@laraveltuts/laravel-10-send-sms-using-twilio-tutorial-example-2676c20f25f8

MailChimp:

Mailchimp is an all-in-one marketing platform that allows businesses and organizations to manage their email marketing campaigns, create landing pages, and track their marketing performance. It is a popular email marketing tool that enables businesses to send bulk emails to their subscribers and customers, automate email campaigns, and analyze the results of their email marketing efforts.

https://github.com/spatie/laravel-newsletter

https://www.youtube.com/watch?v=cDhMcvZOcPA

Laravel Package Development (Own Custom Package):

1. Make your-project/packages/laraphant/contactform/src folder
2. In contactform directory write command: **composer init**

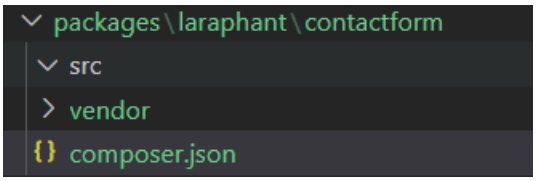
```
Package name (<vendor>/<name>) [yubra/contactform]: laraphant/contactform
Description []: laravel contact form package
Author [Yubraj koirala <yubraj.sixsigma@gmail.com>, n to skip]: Yubraj Koirala
Minimum Stability []: dev
Package Type (e.g. library, project, metapackage, composer-plugin) []: package
License []: MIT

Define your dependencies.

Would you like to define your dependencies (require) interactively [yes]?
Search for a package:
Would you like to define your dev dependencies (require-dev) interactively [yes]?
Search for a package:
Add PSR-4 autoload mapping? Maps namespace "Laraphant\Contactform" to the entered relative path. [src/, n to skip]:
```

```
Do you confirm generation [yes]?
Generating autoload files
Generated autoload files
PSR-4 autoloading configured. Use "namespace Laraphant\Contactform;" in src/
Include the Composer autoloader with: require 'vendor/autoload.php';
```

3. Below folder structure will be created:



4. Now, In your main project directory make provider:

```
php artisan make:provider ContactFormServiceProvider
```

5. Move this ContactFormServiceProvider to your custom package src folder /src/ContactFormServiceProvider.php and modify namespace like below:

```
namespace Laraphant\Contactform;
```

6. Write below code in your custom package composer.json

```
"autoload": {
    "psr-4": {
        "Laraphant\\Contactform\\": "src/"
    }
},
"extra":{
    "laravel":{
        "providers":[
            "Laraphant\\Contactform\\ContactFormServiceProvider"
        ]
    }
},
```

7. In your main composer.json file:

```
"require": {
    "php": "^8.1",
    "guzzlehttp/guzzle": "^7.2",
    "laravel/framework": "^10.10",
    "laraphant/contactform":"dev-main"
},
```

And create repositories array with object:

```
"repositories": [
    {
        "type":"path",
        "url":"packages/laraphant/contactform",
        "options": {
            "symlink": true
        }
    }
],
"extra": {
    "laravel": {
        "dont-discover": []
    }
},
```

8. Write **composer update** command.
9. Setup has been completed to make your package.
10. Create file in contactform/routes/web.php and then you can define your own routes inside your package:

```
<?php


use Illuminate\Support\Facades\Route;

Route::get('/test/project',function(){
    dd('i am testing my package');
});
```

11. To use this routes load routes in your ContactFormServiceProvider.php

```
public function boot(): void
{
   $this->loadRoutesFrom(__DIR__.'/../routes/web.php');
}
```

12. Make controller **php artisan make:controller ContactFormController** and move this file to laraphant/contactform/src/Http/Controllers/ContactFormController.php and modify namespace as below:

```
namespace Laraphant\Contactform\Http\Controllers;
```

13. You can create your blade files in laraphant/contactform/resources/views/create.blade.php

14. To use the blade files you should load blade files in ContactFormServiceProvider.php like below:

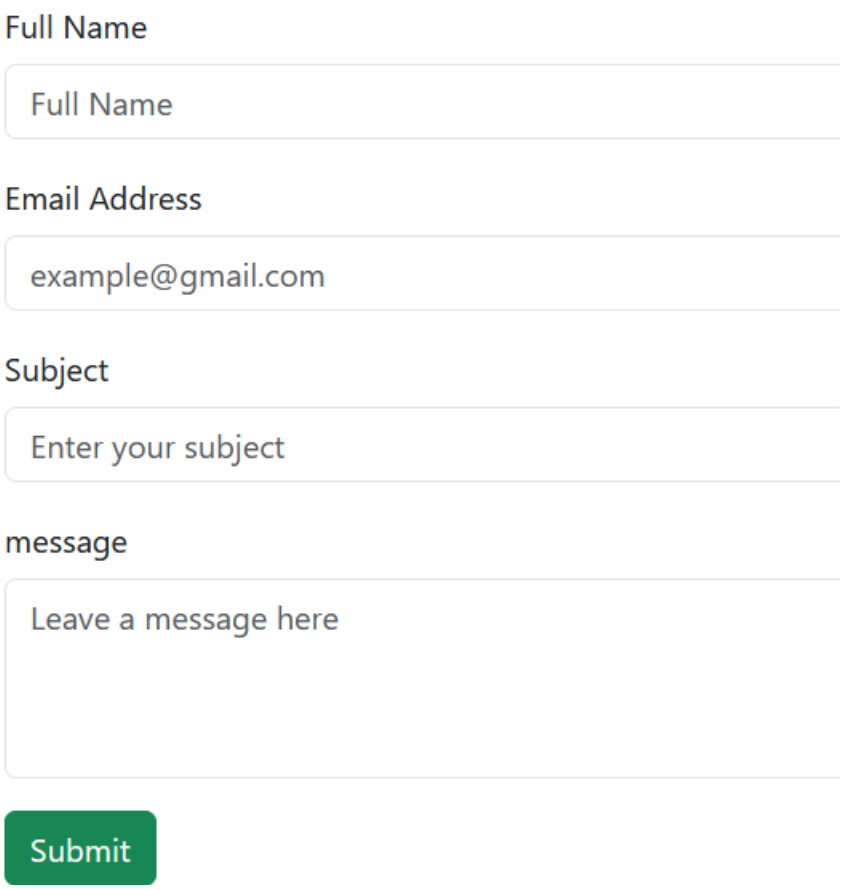
```php
public function boot(): void
{
    $this->loadRoutesFrom(__DIR__.'/../routes/web.php');
    $this->loadViewsFrom(__DIR__.'/../resources/views','contactform');
}
```

15. Now, you can use blade file from your controller like below:

```php
Route::middleware(['guest','web'])->group(function(){
    Route::get('/contact',[ContactFormController::class,'create']);
    Route::post('/submit/message',[ContactFormController::class,'store'])->name('submit.message');
});
```

```php
class ContactFormController extends Controller
{
    public function create(){
        return view('contactform::create');
    }
}
```

16. Make form in create.blade.php like below:



17. When submit called the store method of ControllerFormController.php:

```php
public function store(Request $request){
    $validated=$request->validate([
        'name'=>'required|max:255',
        'email'=>'required|email|max:255',
        'subject'=>'required|max:255',
        'message'=>'required',
    ]);
}
```

18. Create contact model with migration : **php artisan make:model Contact –m** and move Contact.php model in /src/Models/Contact.php folder of your package and then change namespace like below:

```php
namespace Laraphant\Contactform\Models;
```

19. Move the contacts migration from original directory to /contactform/database/migrations/your_migrations_file folder.

```php
public function up(): void
{
    Schema::create('contacts', function (Blueprint $table) {
        $table->id();
        $table->string('name');
        $table->string('email');
        $table->string('subject');
        $table->text('message');
        $table->timestamps();
    });
}
```

```php
protected $fillable=['name','email','message','subject'];
```

20. Load migrations in ContactFormServiceProvider.php

```php
public function boot(): void
{
   $this->loadRoutesFrom(__DIR__.'/../routes/web.php');
   $this->loadViewsFrom(__DIR__.'/../resources/views','contactform');
   $this->loadMigrationsFrom(__DIR__.'/../database/migrations');
}
```

21. Contact us details have been saved into db. Now, the next task is to send email notification to admin that someone wants to contact him.
22. Make config.php file in contactform/config/config.php:

```php
<?php

return[
    'admin_email'=>env('ADMIN_EMAIL',''),
];
```

23. Inside .env file:

```
ADMIN_EMAIL="yubrajkoirala7278@gmail.com"
```

24. Load config file inside ContactFormServiceProvider.php:

```
public function boot(): void
{
    $this->publishes([
        __DIR__.'/../config/config.php'=>config_path('contactform.php')
    ],'contactform-config');
   $this->loadRoutesFrom(__DIR__.'/../routes/web.php');
   $this->loadViewsFrom(__DIR__.'/../resources/views','contactform');
   $this->loadMigrationsFrom(__DIR__.'/../database/migrations');
}
```

And then write command:

```
php artisan vendor:publish
```

Since we have created **contactform-config** so it will be listed like below:

```
Tag: contactform-config ..... 18
```

Here the number is 18 so enter 18 and then enter.

25. Make mail: **php artisan make:mail InqueryEmail** and then move InqueryEmail.php to contactform/src/Mail/InqueryEmail.php folder. And change namespace as below:

```
namespace Laraphant\Contactform\Mail;
```

26. After storing the contact information use mail to send mail to admin:

```
public function store(Request $request)
{
    $validated = $request->validate([
        'name' => 'required|max:255',
        'email' => 'required|email|max:255',
        'subject' => 'required|max:255',
        'message' => 'required',
    ]);

    Contact::create($validated);

    // ===sending email to admin==
    //->getting email of admin from .env file
    $admin_email = \config('contactform.admin_email');

    if ($admin_email === null || $admin_email === '') {
        dd('there is an error');
    } else {
        Mail::to($admin_email)->send(new InqueryEmail($validated));
    }
    // ===end of sending email to admin===
    return back();
}
```

27. Inside InqueryEmail.php: (to configure how to send email is already described above)

```php
public function __construct($data)
{
    $this->data=$data;
}
```

```php
public function envelope(): Envelope
{
    return new Envelope(
        subject: $this->data['subject'],
    );
}
```

```php
public function content(): Content
{
    return new Content(
        markdown: 'contactform::emails.inqueryemail',
    );
}
```

28. Now, Create blade file in contactform/resources/views/emails/inqueryemail.blade.php:

```
Hello admin
<br>
This is testing email

Name: {{$data['name']}}
<br><br>
Email : {{$data['email']}}
<br><br>
Message:
<br><br>
{{$data['message']}}
```

29. Congratulations! Package has been created successfully.
30. Publishing package so that it can be used any projects. Follow below steps to publish your package.
31. Add .gitignore in contactform/.gitignore:

packages > laraphant > contactform > .gitignore

```
/vendor
```

32. Also create README.md in contactform/README.md to write brief description about your package.

packages > laraphant > contactform > README.md

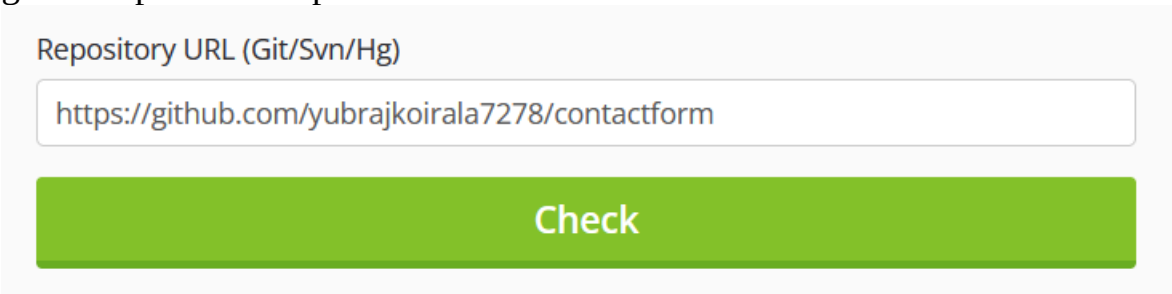
```
Hello this is my contact form package.
```

33. Go to your package directory:

```
cd .\packages\laraphant\contactform\
```

git init
git checkout –b main
git add .
git commit –m “contact form package has been completed”
git tag 1.0.0
Create new repository in github
git remote add origin https://github.com/yubrajkoirala7278/contactform.git
git push –u origin --all

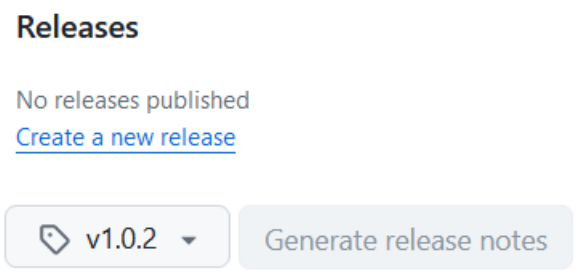
34. After pushing code to github go to https://packagist.org/ and create an account if not created and go to https://packagist.org/packages/submit . Then copy your github repo link and paste like below:



If there is an error then change the name in your composer.json file and push the code again: (optional if error occurs)

```
{
    "name": "new-vendor/contactform",
    "description": "laravel contact form package",
    "type": "package",
    "license": "MIT",
    "autoload": {
        "psr-4": {
            "Laraphant\\Contactform\\": "src/"
        }
    },
```

35. Go to your github package repo in github and click on **create a new release** like below:



And then publish.

36.Congratulations! Your package has been published successfully!

37.Now, using your package in another project. The steps are given below:

38.In your new project install your package:

```
composer require new-vendor/contactform
```

In .env file:

```
ADMIN_EMAIL="yubrajkoirala7278@gmail.com"
```

Write below command to publish your package:

```
> php artisan vendor:publish --tag=contactform-config
```

Now migrate the contacts table that we have used in our package.

```
> php artisan migrate
```

If you haven't setup then setup below code:

```
MAIL_MAILER=smtp
MAIL_HOST=smtp.gmail.com
MAIL_PORT=465
MAIL_USERNAME=nontoxicyubs@gmail.com
MAIL_PASSWORD=pttfjvkziasfqpnh
MAIL_ENCRYPTION=tls
MAIL_FROM_ADDRESS=nontoxicyubs@gmail.com
MAIL_FROM_NAME="${APP_NAME}"
```

Now go to /contact url and use the contact us form.

Use MongoDB(NoSQL) in Laravel:

https://www.mongodb.com/resources/products/compatibilities/mongodb-laravel-integration

1. composer require mongodb/laravel-mongodb
2. Inside database.php (inside connections array)

```
'mongodb' => [
    'driver' => 'mongodb',
    'dsn' => env('DB_URI', 'mongodb://localhost:27017/'),
    'database' => 'mysticode_grow-fin-management',
],
```

3. Inside app.php

```
'providers' => [

/*
* Laravel Framework Service Providers...
*/

MongoDB\Laravel\MongoDBServiceProvider::class,
```

4. Make model and controller:

```
php artisan make:model Post -mc
```

5. Inside Post Model:

```
<?php

namespace App\Models;

use MongoDB\Laravel\Eloquent\Model;

class Post extends Model
{
    protected $connection = 'mongodb';
}
```

6. Inside Post Controller to add post:

```
public function store()
{
    $post = new Post;

    $post->title = 'testing';
    $post->body = 'testing yubraj';
    $post->slug = 'testing ram';

    $post->save();
    return 'saved';
}
```

7. Inside web.php

```
Route::get('/post',[PostController::class,'store']);
```

8. php artisan migrate:fresh --database=mongodb

Unit Testing:

Unit Testing is a software development practice that helps developers to ensure that individual units or components of their code are working as intended. In fact, most unit tests probably focus on a single method.

Inside phpunit.xml:

```
<env name="DB_CONNECTION" value="sqlite"/>
<env name="DB_DATABASE" value=":memory:"/>
```

1. Make test:

```
php artisan make:test PostTest --unit
```

2. Inside PostTest

```
class PostTest extends TestCase
{
    use RefreshDatabase;

    /** @test */
    public function it_can_create_a_post()
    {
        // Manually create a post
        $post = Post::create([
            'title' => 'Test Title',
            'content' => 'Test content of the post.',
        ]);

        // Assert that the post was created and exists in the database
        $this->assertDatabaseHas('posts', [
            'id' => $post->id,
            'title' => 'Test Title',
            'content' => 'Test content of the post.',
        ]);
    }
}
```

3. Run Test:

```
php artisan test --filter=PostTest::it_can_create_a_post
```

Or

```
php artisan test
```

Or

```
php artisan test --filter=PostTest
```

Check User Status (Online or Offline)

1. In Users table:

```php
Schema::create('users', function (Blueprint $table) {
    $table->id();
    $table->string('name');
    $table->string('email')->unique();
    $table->timestamp('email_verified_at')->nullable();
    $table->string('password');
    $table->string('profile_picture')->nullable();
    $table->timestamp('last_login_at')->nullable();
    $table->timestamp('last_seen_at')->nullable();
    $table->rememberToken();
    $table->timestamps();
});
```

2. Inside EventServiceProvider.php:

```php
protected $listen = [
    Registered::class => [
        SendEmailVerificationNotification::class,
    ],
    'Illuminate\Auth\Events\Login' => [
        'App\Listeners\LogSuccessfulLogin',
    ],
    'Illuminate\Auth\Events\Logout' => [
        'App\Listeners\LogSuccessfulLogout',
    ],
];
```

3. Create a Listener to check login user:

```
php artisan make:listener LogSuccessfulLogin
```

4. Inside app/Listeners/LogSuccessfulLogin.php:

```php
namespace App\Listeners;

use Illuminate\Auth\Events\Login;
use Illuminate\Support\Facades\Auth;

class LogSuccessfulLogin
{
    public function handle(Login $event)
    {
        $user = Auth::user();
        $user->last_login_at = now();
        $user->save();
    }
}
```

5. Create listener to check logout user:

```
php artisan make:listener LogSuccessfulLogout
```

6. Inside app/Listeners/LogSuccessfulLogout.php:

```php
namespace App\Listeners;

use Illuminate\Auth\Events\Logout;
use Illuminate\Support\Facades\Cache;
use Illuminate\Support\Facades\Auth;

class LogSuccessfulLogout
{
    public function handle(Logout $event)
    {
        $user = $event->user;

        if ($user) {
            // Remove the user's online status from the cache
            Cache::forget('user-is-online-' . $user->id);

            // Update the last seen time
            $user->last_seen_at = now();
            $user->save();
        }
    }
}
```

7. Inside app/User.php model:

```php
protected $casts = [
    'last_login_at' => 'datetime',
];

public function isOnline()
{
    return Cache::has('user-is-online-' . $this->id);
}

public function setOnlineStatus()
{
    Cache::put('user-is-online-' . $this->id, true, Carbon::now()->addMinutes(5));
}
```

```php
protected $casts = [
    'last_login_at' => 'datetime',
    'last_seen_at' => 'datetime',
];
```

8. Create a middleware:

```
php artisan make:middleware SetOnlineStatus
```

9. Inside app/Http/Middleware/SetOnlineStatus.php:

```php
namespace App\Http\Middleware;

use Closure;
use Illuminate\Support\Facades\Auth;

class SetOnlineStatus
{
    public function handle($request, Closure $next)
    {
        if (Auth::check()) {
            $user = Auth::user();
            $user->setOnlineStatus();
        }

        return $next($request);
    }
}
```

10. Inside app/Http/Kernel.php:

```php
protected $middlewareGroups = [
    'web' => [
        // other middleware
        \App\Http\Middleware\SetOnlineStatus::class,
    ],
];
```

11. Inside your controller:

```php
public function messenger()
{
    $users = User::where('id', '!=', Auth::id())->get();
    return view('messenger', compact('users'));
}
```

12. Inside your blade file:

```blade
<td>
    @if($user->isOnline())
        <span class="text-success">Online</span>
    @else
        <span class="text-danger">Offline</span>
        <br>
        <small>Last seen: {{ $user->last_seen_at ? $user->last_seen_at->diffForHumans() :
    @endif
</td>
```

Filters with AJAX:

❖ Filter with Searching Keyword:

```
{{-- ========search field========== --}}
<input class="form-control" type="search" name="" id="search"
    placeholder="Search products..">
{{-- ========end of search field===== --}}
```

```
// ============Search products when typing in the search field(input text tag)======
let timer;
$('#search').on('keyup', () => {
    clearTimeout(timer);
    timer = setTimeout(searchProducts, 1000);
});
// ==========end of search products when typing in the search field=====
```

❖ Filter with Select Field:

```
<div class="btn-group">
    <button type="button" class="btn btn-sm btn-light dropdown-toggle"
        data-bs-toggle="dropdown">Sorting</button>
    <div class="dropdown-menu dropdown-menu-right sorting-products-filters">
        <a class="dropdown-item" href="javascript:void(0)">Latest</a>
        <a class="dropdown-item" href="javascript:void(0)">Price High</a>
        <a class="dropdown-item" href="javascript:void(0)">Price Low</a>
    </div>
</div>
```

```
// ======Filter products when clicking on sorting options(select tag)========
$('.sorting-products-filters a').on('click', function() {
    const sortBy = $(this).text().trim();
    searchProducts(sortBy);
});
// ======end of filter products when clicking on sorting options===========
```

❖ Filter with Checkbox:

```
@if (count($requiredShopData['brands']) > 0)
    @foreach ($requiredShopData['brands'] as $key => $brand)
        <div class="form-check mb-2">
            <input class="form-check-input brand-check-input" type="checkbox"
                value="{{ $brand->id }}" id="{{ $brand->name }}-{{ $key }}">
            <label class="form-check-label" for="{{ $brand->name }}-{{ $key }}">
                {{ $brand->name }}
            </label>
        </div>
    @endforeach
@endif
```

```
//=========Filter with brand names(checkbox)=========
const getSelectedBrands = () => {
    let selectedBrands = [];
    $('.brand-check-input:checked').each(function() {
        selectedBrands.push($(this).val());
    });
    console.log(selectedBrands);
    return selectedBrands;
};
$('.brand-check-input').on('change', function() {
    searchProducts();
});
// ==========end of filter with brand names==============
```

❖ Filter with radio button:

```
@if (count($requiredShopData['categories']) > 0)
    @foreach ($requiredShopData['categories'] as $key => $category)
        <div class="form-check mb-2">
            <input class="form-check-input category-radio-input" type="radio"
                name="category" value="{{ $category->id }}"
                id="{{ $category->name }}-{{ $key }}">
            <label class="form-check-label" for="{{ $category->name }}-{{ $key }}">
                {{ $category->name }}
            </label>
        </div>
    @endforeach
@endif
```

```
// =========Filter with category(radio button)=======
const getSelectedCategory = () => {
    return $('input[name="category"]:checked').val(); // Get the value of the selected radio button
};
$('.category-radio-input').on('change', function() {
    searchProducts();
});
// ==========end of filter with category(radion button)=======
```

❖ Filter with price range:

```
{{-- price range slider range --}}
    <link rel="stylesheet"
href="https://cdnjs.cloudflare.com/ajax/libs/noUiSlider/14.6.4/nouislider.min.css"
/>
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/noUiSlider/14.6.4/nouislider.min.js"></
script>
```

```
<div id="price-slider" style="margin: 20px;"></div>
<p>Price: <span id="min-price"></span> - <span id="max-price"></span></p>
```

```
// ==========Filter with price slider(range tag)==================
const priceSlider = document.getElementById('price-slider');
noUiSlider.create(priceSlider, {
    start: [0, 1000],
    connect: true,
    range: {
        'min': 0,
        'max': 1000
    },
    step: 10,
    tooltips: [true, true],
    format: {
        to: function(value) {
            return parseInt(value);
        },
        from: function(value) {
            return parseInt(value);
        }
    }
});

priceSlider.noUiSlider.on('update', function(values) {
    $('#min-price').text(values[0]);
    $('#max-price').text(values[1]);
});

priceSlider.noUiSlider.on('change', function() {
    searchProducts();
});
// ===========end of filter with price slider range============
```

❖ Ajax Call for filter and sorting and display products:

```
<div class="row" id="product-card">
    <!-- Products will be displayed here dynamically -->
</div>
```

```
//========== AJAX call to filter and sort products============
            const searchProducts = (sortBy = 'Latest', pageUrl = null) => {
                const keyword = $('#search').val(); // Get searching keyword
                const brands = getSelectedBrands(); // Get the selected brands
                const category = getSelectedCategory(); // Get the selected category
                const minPrice = priceSlider.noUiSlider.get()[0]; // Get min price from slider
                const maxPrice = priceSlider.noUiSlider.get()[1]; // Get max price from slider

                // Determine the URL for pagination or for sorting/filtering
                const url = pageUrl ? pageUrl : "{{ route('frontend.products') }}";

                // Make AJAX GET request with sorting, brand filters, price range, and category
                $.get(url, {
                        'search': keyword,
                        'sort_by': sortBy,
                        'brands': brands,
                        'min_price': minPrice,
                        'max_price': maxPrice,
                        'category_id': category
                    })
                    .done(response => {
                        const products = response.products;
                        $('#product-card').empty();

                        if (products.length > 0) {
                            products.forEach(product => {
                                let comparePrice = '';
                                if (product.compare_price) {
                                    comparePrice =
                                        `<span class="h6 text-underline"><del>${product.compare_price}</del></span>`;
                                }

                                const productCard = `
                                    <div class="col-4">
                                        <div class="card product-card">
                                            <div class="product-image position-relative">
                                                <a href="" class="product-img">
                                                    <img class="card-img-top"
                                                    src="{{ asset('storage/images/products/') }}/${product.image}" alt="" style="height:300px">
                                                </a>
                                                <a class="whishlist" href="222"><i class="far fa-heart"></i></a>

                                                <div class="product-action">
                                                    <a class="btn btn-dark" href="#">
                                                        <i class="fa fa-shopping-cart"></i> Add To Cart
                                                    </a>
                                                </div>
                                            </div>
                                            <div class="card-body text-center mt-3">
                                                <a class="h6 link" href="product.php">${product.title}</a>
                                                <div class="price mt-2">
                                                    <span class="h5"><strong>${product.price}</strong></span>
                                                    ${comparePrice}
                                                </div>
                                            </div>
                                        </div>
                                    </div>
                                `;

                                $('#product-card').append(productCard);
                            });
                        } else {
                            $('#product-card').append('<p>No products found</p>');
                        }
                        // Update pagination links
                        $('#pagination-links').html(response.pagination);

                        // Bind click event for pagination links
                        $('#pagination-links a').on('click', function(e) {
                            e.preventDefault();
                            const pageUrl = $(this).attr('href');
                            searchProducts(sortBy, pageUrl);
                        });
                    });
            };
            // ======end of AJAX call to filter and sort products==========

            //======== Initial product load===============
            searchProducts();
            // ======end of Initial product load==========
```

❖ Inside Controller:

```php
public function products(Request $request)
    {
        $query = Product::with('brand', 'category', 'subCategory');

        // =======Apply search filter=======
        if ($request->search) {
            $query->where('title', 'LIKE', '%' . $request->search . '%')
                ->orWhere('price', 'LIKE', '%' . $request->search . '%');
        }

        // =====Apply brand filter from checkbox=========
        if ($request->brands) {
            $query->whereIn('brand_id', $request->brands);
        }

        // ====Apply category filter from radio button======
        if ($request->category_id) {
            $query->where('category_id', $request->category_id);
        }

        // ======Apply price range filter from slider range======
        if ((int)$request->min_price >= 0 && $request->max_price) {
            $query->whereBetween('price', [(int)$request->min_price, (int)$request->max_price]);
        }

        // ========Apply sorting from select field=========
        switch ($request->sort_by) {
            case 'Price High':
                $query->orderBy('price', 'desc');
                break;
            case 'Price Low':
                $query->orderBy('price', 'asc');
                break;
            case 'Latest':
            default:
                $query->orderBy('created_at', 'desc');
                break;
        }

        // =======Paginate the results===========
        $products = $query->paginate(4);

        // =========return products with pagination=======
        return response()->json([
            'products' => $products->items(),
            'pagination' => (string) $products->links('pagination::bootstrap-5')
        ]);
    }
```

In web.php

```php
// ====Auth===============
Auth::routes([
    'register'=>false,
]);
// ====End of Auth=======


// ========Frontend==============
require __DIR__.'/public.php';
// =======End of Frontend========

// ===========Backend============
Route::middleware(['auth.admin'])->group(function(){
    Route::prefix('admin')->group(function(){
        require __DIR__.'/admin.php';
    });
});
// =======end of Backend====


// =====handle wrong url======
Route::redirect('/{any}', '/', 301);
//=======end of handling wrong url===
```

Inside public.php

```php
Route::get('/products',[ShopController::class,'products'])->name('frontend.products');
```

Inside admin.php

```php
use Illuminate\Support\Facades\Route;

Route::get('/home', [HomeController::class, 'index'])->name('home');

Route::resources([
    'category'=>CategoryController::class,
    'sub_category'=>SubCategoryController::class,
    'brands'=>BrandController::class,
    'products'=>ProductController::class,
]);
```