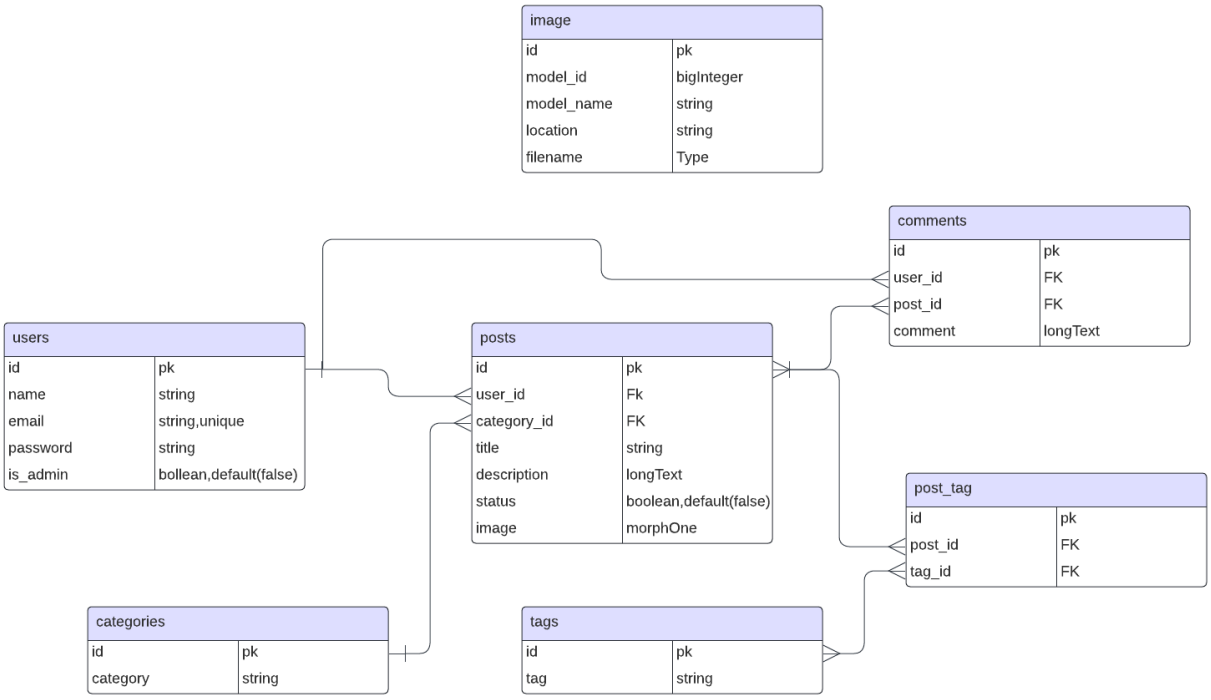
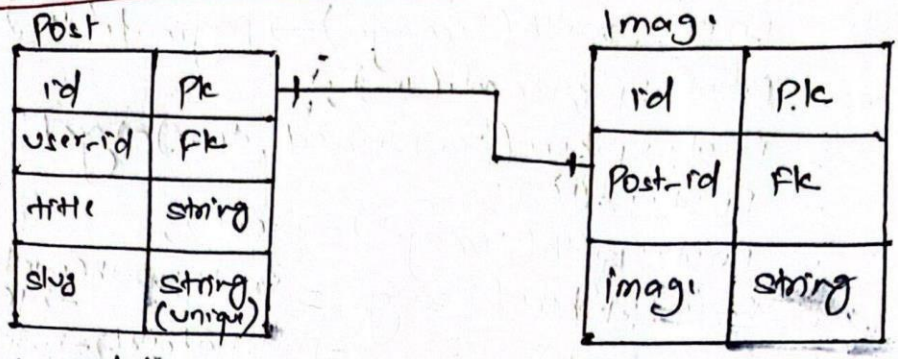


image
id pk
model_id bigInteger
model_name string
location string
filename Type
comments
id pk
user_id FK
post_id FK
comment longText
users
id pk
name string
email string,unique
password string
is_admin bollean,default(false)
posts
id pk
user_id Fk
category_id FK
title string
description longText
status boolean,default(false)
image morphOne
post_tag
id pk
post_id FK
tag_id FK
categories
id pk
category string
tags
id pk
tag string

(*) 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);
```

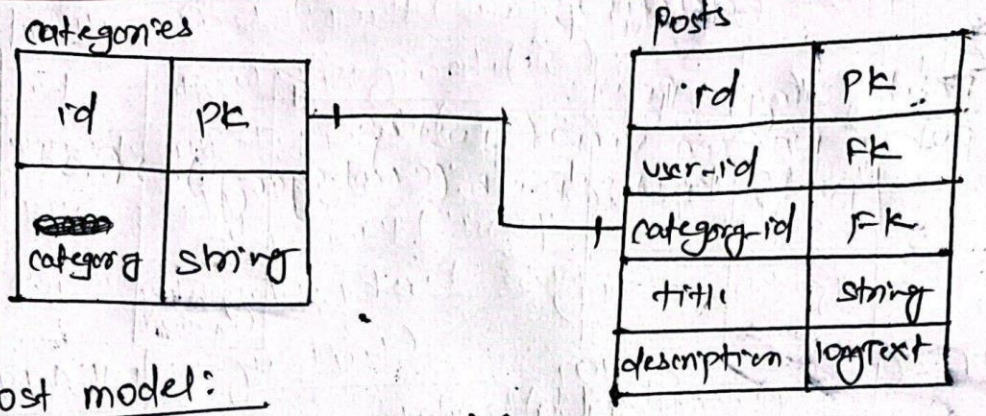
//parent:

```
    $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']
    ]); }
```

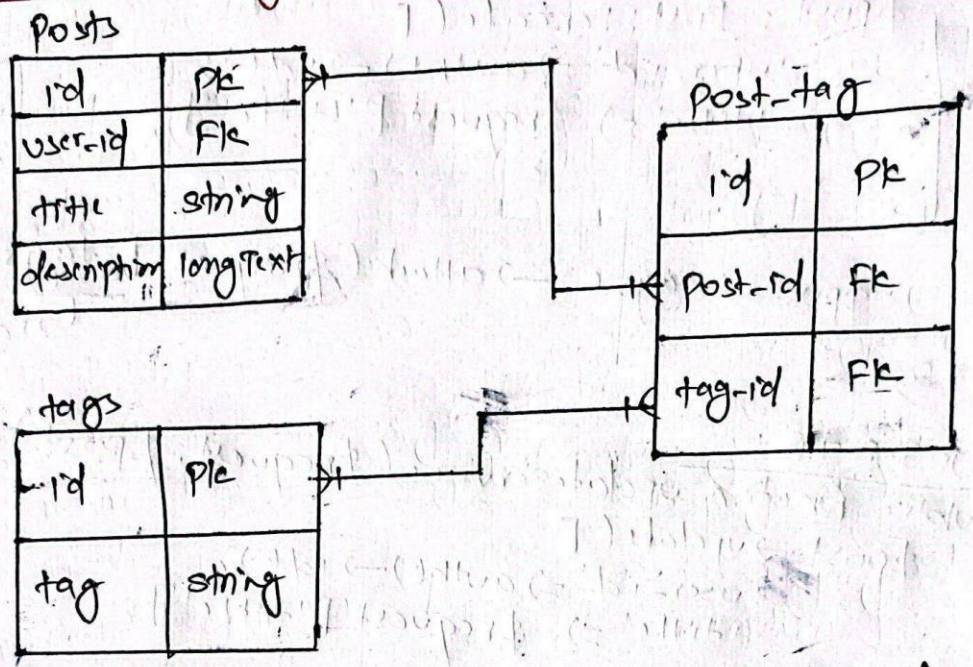
- DELETE:

```
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']);
}
```

(tags → id)

- **UPDATE:**

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

(tags → id)

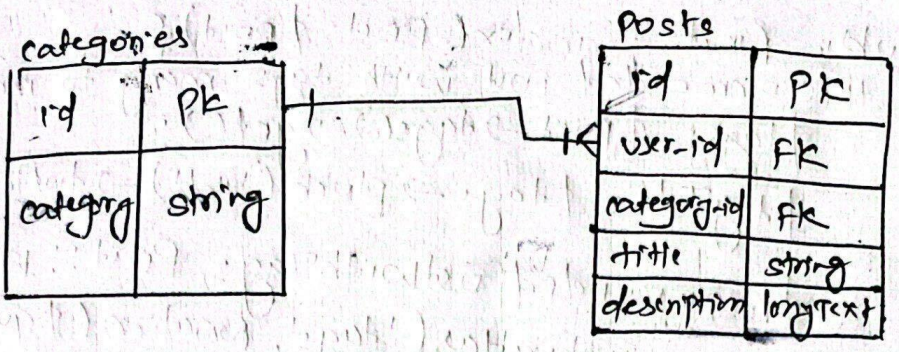
- **Single Post:**

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

(*) Task :

(7)

(1)

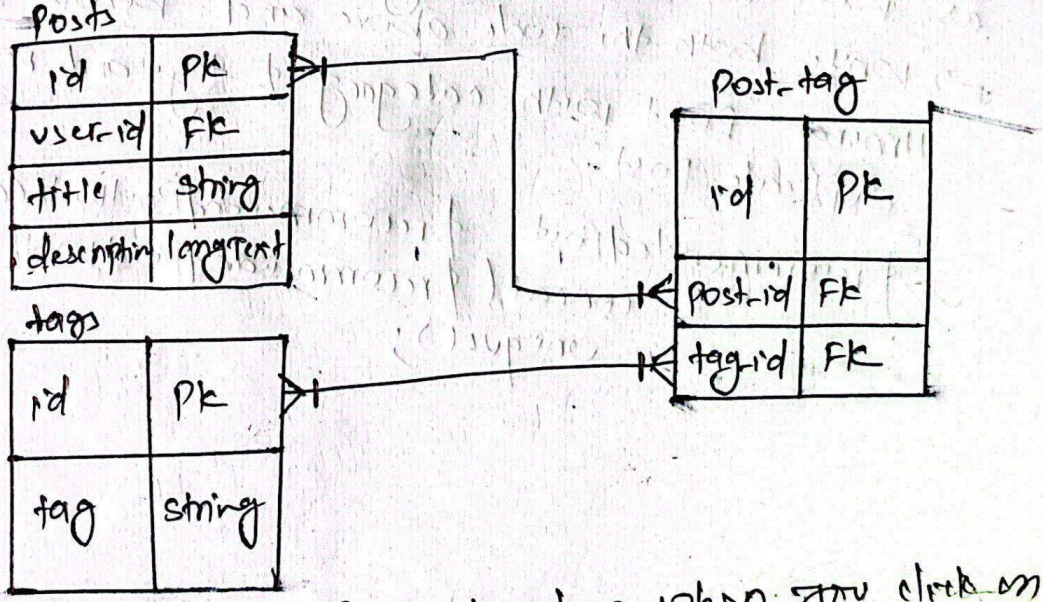


There is a list of posts. Now, when you click on individual post then view that post and also with all the similar posts as recommendation. all similar categories.

```
public function index(Post $post)
    //recommended post with category in one to many relation
    $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'])) {
```

```php
            // 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:

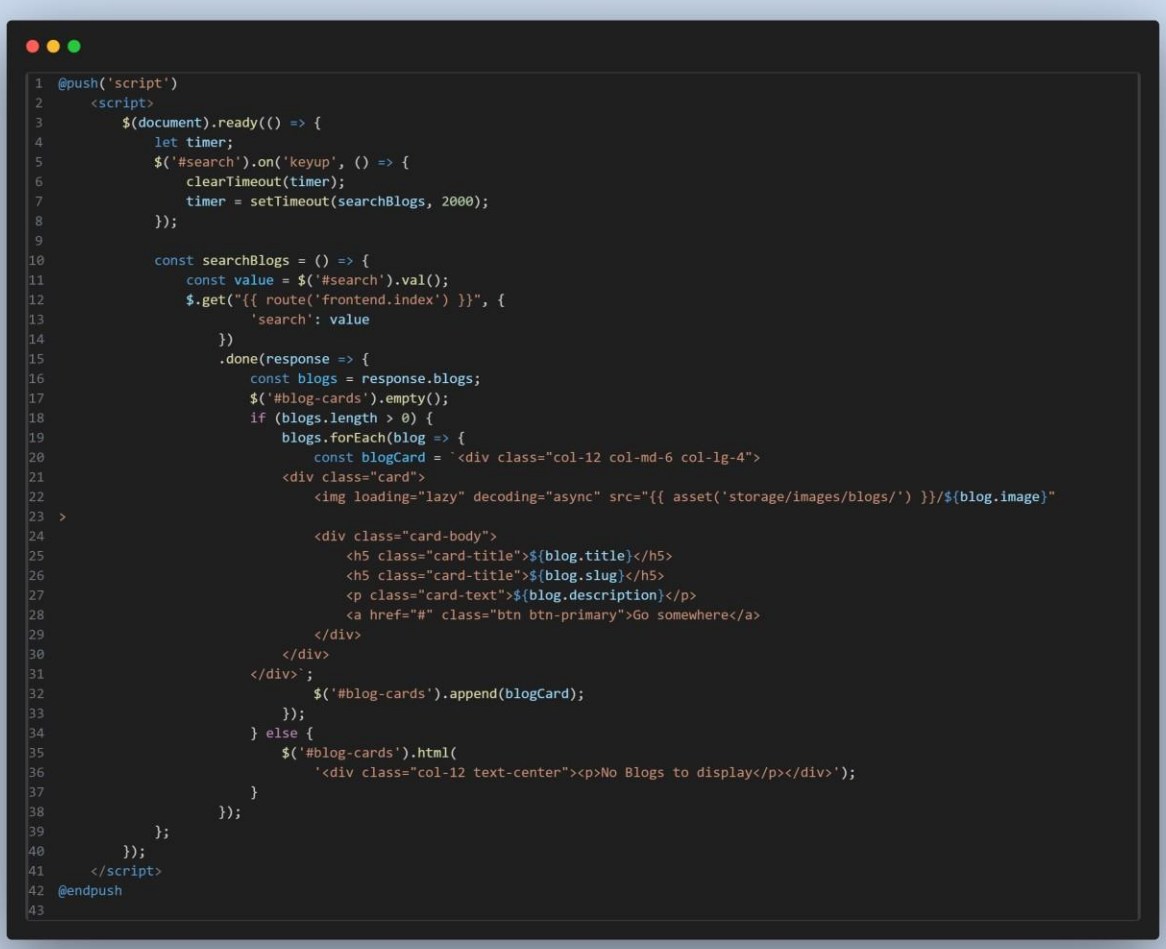
```php
// ====frontend=========
require __DIR__ . '/public.php';

// ====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

```

❖ 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"
```

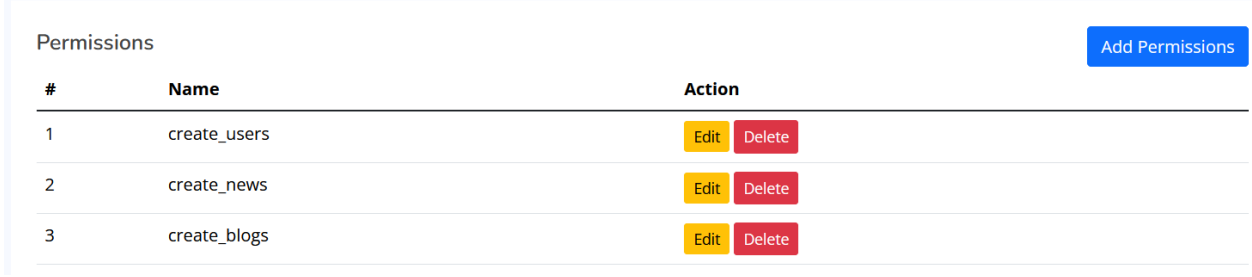
```
php artisan optimize:clear
# or
php artisan config:clear
```

```
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']);
}
```

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>
```

```
<!-- ==========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"
```

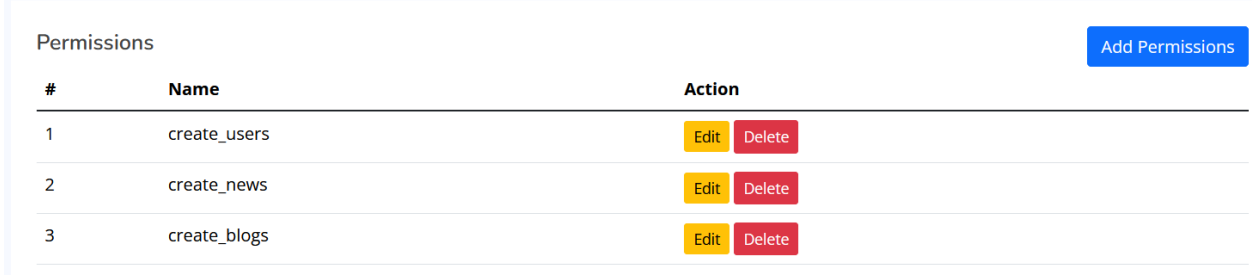
```
php artisan optimize:clear
# or
php artisan config:clear
```

```
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']);
}
```