**PROJECT**

**LARAVEL FORUM**

**SIR WASEEM ABBAS | WEB TECHNOLOGIES**

**FA19-BCS-013**

**Ali Hamza**

# **FUNCTIONALITES:**

* Login & registration with verification
* Database Migrations and Seeding with Factories
* Profile Management
* Auth for Creating threads
* Auth for Replying to Threads
* Sorting of Threads on basis of Tags
* Only Author can Delete and Update Threads
* Only Verified Users Can Reply to and Like Threads
* View Count for threads
* Thread Subscription
* Notification to User when new replies are created
* User Dashboard for notification and Editing profile
* Admin Dashboard for Creating Categories and Tags
* Admin Can Delete all replies and Threads
* Pagination for Showing Threads on Multiple Pages
* HTML purifier to clean inputs before storing to DB
* Livewire Components for Confirmation of thread deletion and editing replies
* Js Alerts and jQuery Animations
* Blade UI and Bootstrap icons and components for designing and responsiveness

# **Models:**

<?php

namespace App\Models;

use App\Traits\HasTimestamps;

use Illuminate\Database\Eloquent\Factories\HasFactory;

use Illuminate\Database\Eloquent\Model;

use Illuminate\Database\Eloquent\Relations\HasMany;

class Category extends Model

{

    use HasFactory;

    use HasTimestamps;

    protected $fillable = ['name','slug'];

    public function threads(): HasMany

    {

*return* *$this*->hasMany(Thread::class);

    }

    public function name(): string

    {

*return* *$this*->name;

    }

    public function id(): int

    {

*return* *$this*->id;

    }

    public function slug(): string

    {

*return* *$this*->slug;

    }

}

<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;

use Illuminate\Database\Eloquent\Model;

use Illuminate\Database\Eloquent\Relations\HasMany;

class Like extends Model

{

    use HasFactory;

    protected $fillable = ['user\_id'];

}

<?php

namespace App\Models;

use App\Models\ReplyAble;

use App\Traits\HasAuthor;

use App\Traits\HasLikes;

use Illuminate\Support\Str;

use App\Traits\ModelHelpers;

use App\Traits\HasTimestamps;

use Illuminate\Database\Eloquent\Model;

use Illuminate\Database\Eloquent\Relations\MorphTo;

use Illuminate\Database\Eloquent\Factories\HasFactory;

final class Reply extends Model

{

    use HasLikes;

    use HasAuthor;

    use HasFactory;

    use HasTimestamps;

    use ModelHelpers;

    const TABLE = 'replies';

*/\*\**

*\* {@inheritdoc}*

*\*/*

    protected $table = self::TABLE;

*/\*\**

*\* {@inheritdoc}*

*\*/*

    protected $fillable = [

        'body',

    ];

    protected $with = [

        'likesRelation',

    ];

    public function id(): int

    {

*return* *$this*->id;

    }

    public function body(): string

    {

*return* *$this*->body;

    }

    public function excerpt(int $limit = 200): string

    {

*return* Str::limit(strip\_tags(*$this*->body()), $limit);

    }

    public function to(ReplyAble $replyAble)

    {

*return* *$this*->replyAbleRelation()->associate($replyAble);

    }

    public function replyAble(): ReplyAble

    {

*return* *$this*->replyAbleRelation;

    }

    public function replyAbleRelation(): MorphTo

    {

*return* *$this*->morphTo('replyAbleRelation', 'replyable\_type', 'replyable\_id');

    }

}

<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Relations\MorphMany;

*/\*\**

*\* This interface allows models to receive replies.*

*\*/*

interface ReplyAble

{

    public function title(): string;

*/\*\**

*\* @return \App\Models\Reply[]*

*\*/*

    public function replies();

*/\*\**

*\* @return \App\Models\Reply[]*

*\*/*

    public function latestReplies(int $amount = 5);

    public function deleteReplies();

    public function repliesRelation(): MorphMany;

    public function isConversationOld(): bool;

    public function replyAbleSubject(): string;

}

<?php

namespace App\Models;

use App\Models\User;

use App\Models\SubscriptionAble;

use Illuminate\Database\Eloquent\Model;

use Illuminate\Database\Eloquent\Relations\BelongsTo;

use Illuminate\Database\Eloquent\Factories\HasFactory;

use Illuminate\Database\Eloquent\Relations\MorphTo;

class Subscription extends Model

{

    use HasFactory;

    protected $table = 'subscriptions';

    public function user(): User

    {

*return* *$this*->userRelation;

    }

    public function userRelation(): BelongsTo

    {

*return* *$this*->BelongsTo(User::class, 'user\_id');

    }

    public function subscriptionAble(): SubscriptionAble

    {

*return* *$this*->subscriptionAbleRelation;

    }

    public function subscriptionAbleRelation(): MorphTo

    {

*return* *$this*->morphTo('subscriptionAbleRelation', 'subscriptionable\_type', 'subscriptionable\_id');

    }

}

<?php

namespace App\Models;

use App\Models\User;

use Illuminate\Database\Eloquent\Relations\MorphMany;

interface SubscriptionAble

{

    public function subscriptions();

    public function subscriptionsRelation(): MorphMany;

    public function hasSubscriber(User $user): bool;

}

<?php

namespace App\Models;

use App\Traits\HasTimestamps;

use Illuminate\Database\Eloquent\Factories\HasFactory;

use Illuminate\Database\Eloquent\Model;

use Illuminate\Database\Eloquent\Relations\MorphToMany;

class Tag extends Model

{

    use HasFactory;

    use HasTimestamps;

    protected $table = 'tags';

    protected $fillable =[

        'name',

        'slug',

    ];

    public function threads(): MorphToMany

    {

*return* *$this*->morphedByMany(Thread::class, 'taggable');

    }

    public function id(): int

    {

*return* *$this*->id;

    }

    public function name(): string

    {

*return* *$this*->name;

    }

    public function slug(): string

    {

*return* *$this*->slug;

    }

}

<?php

namespace App\Models;

use App\Traits\HasTags;

use App\Traits\HasLikes;

use App\Traits\HasAuthor;

use App\Traits\HasReplies;

use Illuminate\Support\Str;

use App\Traits\HasSubscriptions;

use Illuminate\Database\Eloquent\Model;

use Illuminate\Database\Eloquent\Builder;

use CyrildeWit\EloquentViewable\Contracts\Viewable;

use CyrildeWit\EloquentViewable\InteractsWithViews;

use Illuminate\Database\Eloquent\Relations\BelongsTo;

use Illuminate\Database\Eloquent\Factories\HasFactory;

class Thread extends Model implements ReplyAble,SubscriptionAble, Viewable

{

    use HasLikes;

    use HasTags;

    use HasFactory;

    use HasAuthor;

    use HasReplies;

    use HasSubscriptions;

    use InteractsWithViews;

    const TABLE = 'threads';

    protected $table = self::TABLE;

*//protected $table = 'threads';*

    protected $fillable =[

        'title',

        'body',

        'slug',

        'category\_id',

        'author\_id',

    ];

    protected $with =[

        'tagsRelation',

        'category',

        'likesRelation',

        'authorRelation',

    ];

    public function category(): BelongsTo

    {

*return* *$this*->belongsTo(Category::class);

    }

    public function except(int $limit = 250): string

    {

*return* Str::limit(strip\_tags(*$this*->body()),$limit);

    }

    public function replyAbleSubject(): string

    {

*return* *$this*->title();

    }

    public function title(): string

    {

*return* *$this*->title;

    }

    public function slug(): string

    {

*return* *$this*->slug;

    }

    public function id(): int

    {

*return* *$this*->id;

    }

    public function body(): string

    {

*return* *$this*->body;

    }

    public function delete()

    {

*$this*->removeTags();

        parent::delete();

    }

    public function scopeForTag(Builder $query,string $tag): Builder

    {

*return* $query->whereHas('tagsRelation',function ($query) use ($tag) {

            $query->where('tags.slug',$tag);

        });

    }

}

<?php

namespace App\Models;

use App\Models\Thread;

use App\Traits\HasPoints;

use App\Traits\HasFollows;

use App\Traits\ModelHelpers;

use App\Traits\HasTimestamps;

use Laravel\Sanctum\HasApiTokens;

use Laravel\Jetstream\HasProfilePhoto;

use Illuminate\Notifications\Notifiable;

use Laravel\Fortify\TwoFactorAuthenticatable;

use Illuminate\Contracts\Auth\MustVerifyEmail;

use Illuminate\Database\Eloquent\Relations\HasMany;

use Illuminate\Database\Eloquent\Factories\HasFactory;

use Illuminate\Foundation\Auth\User as Authenticatable;

class User extends Authenticatable implements MustVerifyEmail

{

    use Notifiable;

    use HasFactory;

*//use HasFollows;*

*//use HasPoints;*

    use HasApiTokens;

    use ModelHelpers;

    use HasTimestamps;

    use HasProfilePhoto;

    use TwoFactorAuthenticatable;

    const DEFAULT = 1;

    const MODERATOR = 2;

    const ADMIN = 3;

    const TABLE = 'users';

    protected $table = self::TABLE;

*/\*\**

*\* The attributes that are mass assignable.*

*\**

*\* @var array*

*\*/*

    protected $fillable = [

        'name',

        'email',

        'password',

        'bio',

        'type',

    ];

*/\*\**

*\* The attributes that should be hidden for arrays.*

*\**

*\* @var array*

*\*/*

    protected $hidden = [

        'password',

        'remember\_token',

        'two\_factor\_recovery\_codes',

        'two\_factor\_secret',

    ];

*/\*\**

*\* The attributes that should be cast to native types.*

*\**

*\* @var array*

*\*/*

    protected $casts = [

        'email\_verified\_at' => 'datetime',

    ];

*/\*\**

*\* The accessors to append to the model's array form.*

*\**

*\* @var array*

*\*/*

    protected $appends = [

        'profile\_photo\_url',

    ];

    public function id(): int

    {

*return* *$this*->id;

    }

    public function name(): string

    {

*return* *$this*->name;

    }

    public function userName(): string

    {

*return* *$this*->username;

    }

    public function emailAddress(): string

    {

*return* *$this*->email;

    }

    public function bio(): string

    {

*return* *$this*->bio;

    }

    public function type(): int

    {

*return* (int) *$this*->type;

    }

    public function isModerator(): bool

    {

*return* *$this*->type() === self::MODERATOR;

    }

    public function isAdmin(): bool

    {

*return* *$this*->type() === self::ADMIN;

    }

    public function threads()

    {

*return* *$this*->threadsRelation;

    }

    public function latestThreads(int $amount = 5)

    {

*return* *$this*->threadsRelation()->latest()->limit($amount)->get();

    }

    public function deleteThreads()

    {

*foreach* (*$this*->threads() as $thread) {

            $thread->delete();

        }

    }

    public function threadsRelation(): HasMany

    {

*return* *$this*->hasMany(Thread::class, 'author\_id');

    }

    public function countThreads(): int

    {

*return* *$this*->threadsRelation()->count();

    }

    public function replies()

    {

*return* *$this*->replyAble;

    }

    public function latestReplies(int $amount = 10)

    {

*return* *$this*->replyAble()->latest()->limit($amount)->get();

    }

    public function deleteReplies()

    {

*foreach* (*$this*->replyAble()->get() as $reply) {

            $reply->delete();

        }

    }

    public function countReplies(): int

    {

*return* *$this*->replyAble()->count();

    }

    public function replyAble(): HasMany

    {

*return* *$this*->hasMany(Reply::class, 'author\_id');

    }

}

# **VIEWS:**

Create Thread:

<x-guest-layout>

    <main *class*="grid grid-cols-4 gap-8 mt-8 wrapper">

        <x-partials.sidenav />

        <section *class*="flex flex-col col-span-3 gap-y-4">

            <small *class*="text-sm text-gray-400">discussion>create</small>

            <article *class*="p-5 bg-white shadow">

                <div *class*="grid grid-cols-8">

*{{-- Avatar --}}*

                    <div *class*="col-span-1">

*{{-- <x-user.avatar/> --}}*

                    </div>

*{{-- Create --}}*

                    <div *class*="col-span-7 space-y-6">

                        <x-form *action*="{{ route('threads.store') }}">

                            <div *class*="space-y-8">

*{{-- Title --}}*

                                <div>

                                    <x-form.label *for*="title" *value*="{{ \_\_('Title') }}" />

                                    <x-form.input *id*="title" *class*="block w-full mt-1" *type*="text" *name*="title" *:value*="old('title')" *autofocus* />

                                    <x-form.error *for*="title" />

                                </div>

*{{-- Category --}}*

                                <div>

                                    <x-form.label *for*="category" *value*="{{ \_\_('Category') }}" />

                                    <select *name*="category" *id*="category" *class*="w-full text-sm border-gray-300 rounded-md shadow-sm focus:border-blue-300 focus:ring focus:ring-blue-200 focus:ring-opacity-50">

                                        <option *value*="">Select Category</option>

                                    @foreach ($categories as $category)

                                        <option *value*="{{ $category->id() }}">{{ $category->name() }}</option>

                                    @endforeach

                                    </select>

                                    <x-form.error *for*="category" />

                                </div>

*{{-- Tag --}}*

                                <div>

                                    <x-form.label *for*="tags" *value*="{{ \_\_('Tags') }}" />

                                    <select *name*="tags[]" *id*="tags" *x-data*="{}" *x-init*="function () {choices($el)}"

*class*="w-full text-sm border-gray-300 rounded-md shadow-sm focus:border-blue-300 focus:ring focus:ring-blue-200

                                    focus:ring-opacity-50" *multiple* >

                                        <option *value*="">Select Tags</option>

                                    @foreach ($tags as $tag)

                                        <option *value*="{{ $tag->id() }}">{{ $tag->name() }}</option>

                                    @endforeach

                                    </select>

                                    <x-form.error *for*="tags" />

                                </div>

*{{-- Body --}}*

                                <div>

                                    <x-form.label *for*="body" *value*="{{ \_\_('Description') }}" />

                                    <x-trix *name*="body" *styling*="shadow-inner bg-gray-100" />

                                    <x-form.error *for*="body" />

                                </div>

*{{-- Button --}}*

                                <x-buttons.primary>

                                    {{ \_\_('Create') }}

                                </x-buttons.primary>

                        </x-form>

                    </div>

                </div>

            </article>

        </section>

    </main>

</x-guest-layout>

Edit Thread:

<x-guest-layout>

    <main *class*="grid grid-cols-4 gap-8 mt-8 wrapper">

        <x-partials.sidenav />

        <section *class*="flex flex-col col-span-3 gap-y-4">

            <small *class*="text-sm text-gray-400">threads>{{ $thread->title() }}>edit</small>

            <article *class*="p-5 bg-white shadow">

                <div *class*="w-full">

*{{-- Create --}}*

                    <div *class*="space-y-6">

                        <x-form *action*="{{ route('threads.update', $thread->slug()) }}">

                            <div *class*="space-y-8">

*{{-- Title --}}*

                                <div>

                                    <x-form.label *for*="title" *value*="{{ \_\_('Title') }}" />

                                    <x-form.input *id*="title" *class*="block w-full mt-1" *type*="text" *name*="title" *:value*="$thread->title()" *autofocus* />

                                    <x-form.error *for*="title" />

                                </div>

*{{-- Category --}}*

                                <div>

                                    <x-form.label *for*="category" *value*="{{ \_\_('Category') }}" />

                                    <select *name*="category" *id*="category" *class*="w-full text-sm border-gray-300 rounded-md shadow-sm focus:border-blue-300 focus:ring focus:ring-blue-200 focus:ring-opacity-50">

                                        @foreach ($categories as $category)

                                        <option *value*="{{ $category->id() }}" @if($category->id() == $selectedCategory->id) *selected* @endif>

                                            {{ $category->name() }}

                                        </option>

                                        @endforeach

                                    </select>

                                    <x-form.error *for*="category" />

                                </div>

*{{-- Tags --}}*

                                <div>

                                    <x-form.label *for*="tags" *value*="{{ \_\_('Tags') }}" />

                                    <select *name*="tags[]" *id*="tags" *x-data*="{}" *x-init*="function () { choices($el) }" *class*="w-full text-sm border-gray-300 rounded-md shadow-sm focus:border-blue-300 focus:ring focus:ring-blue-200 focus:ring-opacity-50" *multiple*>

                                        @foreach ($tags as $tag)

                                        <option *value*="{{ $tag->id() }}" @if(in\_array($tag->id(), $oldTags)) *selected* @endif

                                            >

                                            {{ $tag->name() }}

                                        </option>

                                        @endforeach

                                    </select>

                                    <x-form.error *for*="tags" />

                                </div>

*{{-- Body --}}*

                                <div>

                                    <x-form.label *for*="body" *value*="{{ \_\_('Description') }}" />

                                    <x-trix *name*="body" *styling*="shadow-inner bg-gray-100 h-56">

                                        {{ $thread->body() }}

                                    </x-trix>

                                    <x-form.error *for*="body" />

                                </div>

*{{-- Button --}}*

                                <x-buttons.primary>

                                    {{ \_\_('Update') }}

                                </x-buttons.primary>

                        </x-form>

                    </div>

                </div>

            </article>

        </section>

    </main>

</x-guest-layout>

Show Thread:

<x-guest-layout>

    <main *class*="wrapper">

        <section *class*="grid grid-cols-4 gap-8 mt-8">

*{{-- SIDE NAV --}}*

            <x-partials.sidenav />

            <div *class*="flex flex-col col-span-3 gap-y-4">

*{{-- Alert --}}*

            <x-alerts.main />

            @foreach ($threads as $thread)

                <x-thread *:thread*="$thread" />

            @endforeach

*{{-- PAGINATION --}}*

                <div *class*="mt-5">

                    {{ $threads->render() }}

                </div>

            </div>

        </section>

    </main>

</x-guest-layout>

<x-guest-layout>

    <main *class*="grid grid-cols-4 gap-8 mt-8 wrapper">

        <x-partials.sidenav *:thread*="$thread" />

        <section *class*="flex flex-col col-span-3 gap-y-4">

            <x-alerts.main />

            <small *class*="text-sm text-gray-400">Threads>{{ $category->name() }}>{{ $thread->title() }}</small>

            <article *class*="p-5 bg-white shadow">

                <div *class*="relative grid grid-cols-8">

*{{-- Avatar --}}*

                    <div *class*="col-span-1">

                        <x-user.avatar *:user*="$thread->author()" />

                    </div>

*{{-- Thread --}}*

                    <div *class*="relative col-span-7 space-y-6">

                        <div *class*="space-y-3">

                            <h2 *class*="text-xl tracking-wide hover:text-blue-400">

                                {{ $thread->title() }}

                            </h2>

                            <div *class*="text-gray-500">

                                {!! $thread->body() !!}

                            </div>

                        </div>

                        <div *class*="flex justify-between">

                            <div *class*="flex space-x-5 text-gray-500">

*{{-- Likes --}}*

                                <livewire:like-thread *:thread*="$thread" />

*{{-- Comments Count --}}*

                <div *class*="flex items-center space-x-2">

                    <x-heroicon-o-chat-alt *class*="w-4 h-4 text-green-300" />

                    <span *class*="text-xs text-gray-500">{{ count($thread->replies()) }}</span>

                </div>

*{{-- View Count --}}*

                                <div *class*="flex items-center space-x-2">

                                    <x-heroicon-o-eye *class*="w-4 h-4 text-blue-300" />

                                    <span *class*="text-xs text-gray-500">{{ views($thread)->count() }}</span>

                                </div>

                            </div>

*{{-- Date Posted --}}*

                            <div *class*="flex items-center text-xs text-gray-500">

                                <x-heroicon-o-clock *class*="w-4 h-4 mr-1" />

                                Posted: {{ $thread->created\_at->diffForHumans() }}

                            </div>

                        </div>

                    </div>

*{{-- Edit Button --}}*

                    <div *class*="absolute top-0 right-2">

                        <div *class*="flex space-x-2">

                            @can(App\Policies\ThreadPolicy::UPDATE, $thread)

                            <x-links.secondary *href*="{{ route('threads.edit', $thread->slug()) }}">

                                Edit

                            </x-links.secondary>

                            @endcan

                            @can(App\Policies\ThreadPolicy::DELETE, $thread)

                            <livewire:thread.delete *:thread*="$thread" *:key*="$thread->id()" />

                            @endcan

                        </div>

                    </div>

                </div>

            </article>

*{{-- Replies --}}*

            <div *class*="mt-6 space-y-5">

                <h2 *class*="mb-0 text-sm font-bold uppercase">Replies</h2>

                <hr>

                @foreach($thread->replies() as $reply)

                <livewire:reply.update *:reply*="$reply" *:wire:key*="$reply->id()" />

                @endforeach

            </div>

            @auth

            <div *class*="p-5 space-y-4 bg-white shadow">

                <h2 *class*="text-gray-500">Post a reply</h2>

                <x-form *action*="{{ route('replies.store') }}">

                    <div>

                        <input *type*="text" *name*="body" *class*="w-full bg-gray-200 border-none shadow-inner focus:ring-blue-400" />

                        <x-form.error *for*="body" />

                        <input *type*="hidden" *name*="replyable\_id" *value*="{{ $thread->id() }}">

                        <x-form.error *for*="replyable\_id" />

                        <input *type*="hidden" *name*="replyable\_type" *value*="threads">

                        <x-form.error *for*="replyable\_type" />

                    </div>

                    <div *class*="grid mt-4">

*{{-- Button --}}*

                        <x-buttons.primary *class*="justify-self-end">

                            {{ \_\_('Post') }}

                        </x-buttons.primary>

                    </div>

                </x-form>

            </div>

            @else

            <div *class*="flex justify-between p-4 text-gray-700 bg-blue-200 rounded">

                <h2>Please login to leave a comment</h2>

                <a *href*="{{ route('login') }}">Login</a>

            </div>

            @endauth

        </section>

    </main>

</x-guest-layout>

# Controllers:

<?php

namespace App\Http\Controllers\Pages;

use App\Models\Tag;

use App\Models\Thread;

use App\Models\Category;

use App\Jobs\CreateThread;

use App\Jobs\UpdateThread;

use Illuminate\Support\Str;

use Illuminate\Http\Request;

use App\Policies\ThreadPolicy;

use Mews\Purifier\Facades\Purifier;

use App\Http\Controllers\Controller;

use Illuminate\Support\Facades\Auth;

use App\Http\Middleware\Authenticate;

use App\Http\Requests\ThreadStoreRequest;

use App\Jobs\SubscribeToSubscriptionAble;

use function PHPUnit\Framework\returnSelf;

use App\Jobs\UnsubscribeFromSubscriptionAble;

use Illuminate\Auth\Middleware\EnsureEmailIsVerified;

class ThreadController extends Controller

{

*/\*\**

*\* Display a listing of the resource.*

*\**

*\* @return \Illuminate\Http\Response*

*\*/*

    public function \_\_construct()

    {

*return* *$this*->middleware([

            Authenticate::class,

            EnsureEmailIsVerified::class

            ])->except(['index','show']);

    }

    public function index()

    {

*return* view('pages.threads.index',[

            'threads' => Thread::orderBy('id', 'desc')->paginate(10),

        ]);

    }

*/\*\**

*\* Show the form for creating a new resource.*

*\**

*\* @return \Illuminate\Http\Response*

*\*/*

    public function create()

    {

*return* view('pages.threads.create',[

            'categories' => Category::all(),

            'tags' => Tag::all(),

        ]);

    }

*/\*\**

*\* Store a newly created resource in storage.*

*\**

*\* @param  \Illuminate\Http\Request  $request*

*\* @return \Illuminate\Http\Response*

*\*/*

    public function store(ThreadStoreRequest $request)

    {

*$this*->dispatchSync(CreateThread::fromRequest($request));

*return* redirect()->route('threads.index')->with('success','Post Uploaded!');

    }

*/\*\**

*\* Display the specified resource.*

*\**

*\* @param  \App\Models\Thread  $thread*

*\* @return \Illuminate\Http\Response*

*\*/*

    public function show(Category $category, Thread $thread)

    {

        $expireAt = now()->addHours(5);

        views($thread)

        ->cooldown($expireAt)

        ->record();

*return* view('pages.threads.show',compact('thread', 'category'));

    }

*/\*\**

*\* Show the form for editing the specified resource.*

*\**

*\* @param  \App\Models\Thread  $thread*

*\* @return \Illuminate\Http\Response*

*\*/*

    public function edit(Thread $thread)

    {

*$this*->authorize(ThreadPolicy::UPDATE, $thread);

        $oldTags = $thread->tags()->pluck('id')->toArray();

        $selectedCategory = $thread->category;

*return* view('pages.threads.edit', [

            'thread'            => $thread,

            'tags'              => Tag::all(),

            'oldTags'           => $oldTags,

            'categories'        => Category::all(),

            'selectedCategory'  => $selectedCategory,

        ]);

    }

*/\*\**

*\* Update the specified resource in storage.*

*\**

*\* @param  \Illuminate\Http\Request  $request*

*\* @param  \App\Models\Thread  $thread*

*\* @return \Illuminate\Http\Response*

*\*/*

    public function update(ThreadStoreRequest $request, Thread $thread)

    {

*$this*->authorize(ThreadPolicy::UPDATE, $thread);

*$this*->dispatchSync(UpdateThread::fromRequest($thread, $request));

*return* redirect()->route('threads.index')->with('success', 'Thread Updated!');

    }

    public function subscribe(Request $request, Category $category, Thread $thread)

    {

*$this*->authorize(ThreadPolicy::SUBSCRIBE, $thread);

*$this*->dispatchSync(new SubscribeToSubscriptionAble($request->user(), $thread));

*return* redirect()->route('threads.show', [$thread->category->slug(), $thread->slug()])

            ->with('success', 'You have been subscribed to this thread');

    }

    public function unsubscribe(Request $request, Category $category, Thread $thread)

    {

*$this*->authorize(ThreadPolicy::UNSUBSCRIBE, $thread);

*$this*->dispatchSync(new UnsubscribeFromSubscriptionAble($request->user(), $thread));

*return* redirect()->route('threads.show', [$thread->category->slug(), $thread->slug()])

            ->with('success', 'You have been unsubscribed from this thread');

    }

}

<?php

namespace App\Http\Controllers\Pages;

use App\Models\Reply;

use App\Jobs\CreateReply;

use Illuminate\Http\Request;

use App\Policies\ReplyPolicy;

use App\Http\Controllers\Controller;

use App\Http\Middleware\Authenticate;

use App\Http\Requests\CreateReplyRequest;

use Illuminate\Auth\Middleware\EnsureEmailIsVerified;

class ReplyController extends Controller

{

    public function \_\_construct()

    {

*return* *$this*->middleware([Authenticate::class, EnsureEmailIsVerified::class]);

    }

    public function store(CreateReplyRequest $request)

    {

*$this*->authorize(ReplyPolicy::CREATE, Reply::class);

*$this*->dispatchSync(CreateReply::fromRequest($request));

*return* back()->with('success', 'Reply Created');

    }

    public function redirect($id, $type)

    {

        $reply = Reply::where('replyable\_id', $id)->where('replyable\_type', $type)->firstOrFail();

*return* redirect()->route('threads.show', [$reply->replyAble()->category->slug(), $reply->replyAble()->slug()]);

    }

}

# ROUTES:

<?php

use Illuminate\Support\Facades\Route;

use App\Http\Controllers\PageController;

use App\Http\Controllers\Pages\TagController;

use App\Http\Controllers\Pages\HomeController;

use App\Http\Controllers\Pages\ReplyController;

use App\Http\Controllers\Pages\ThreadController;

use App\Http\Controllers\Pages\ProfileController;

use App\Http\Controllers\Dashboard\NotificationController;

*require* 'admin.php';

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

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

Route::group(['prefix' => 'threads', 'as' => 'threads.'],function(){

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

    Route::get('create', [ThreadController::class, 'create'])->name('create');

    Route::post('/', [ThreadController::class, 'store'])->name('store');

    Route::get('/{thread:slug}/edit', [ThreadController::class, 'edit'])->name('edit');

    Route::post('/{thread:slug}', [ThreadController::class, 'update'])->name('update');

    Route::get('/{category:slug}/{thread:slug}', [ThreadController::class, 'show'])->name('show');

    Route::get('/{category:slug}/{thread:slug}/subscribe', [ThreadController::class, 'subscribe'])->name('subscribe');

    Route::get('/{category:slug}/{thread:slug}/unsubscribe', [ThreadController::class, 'unsubscribe'])->name('unsubscribe');

    Route::group(['as' => 'tags.'], function () {

        Route::get('/{tag:slug}', [TagController::class, 'index'])->name('index');

    });

});

Route::group(['prefix' => 'replies', 'as' => 'replies.'], function () {

*/\* Name: Replies*

*\* Url: /replies/\**

*\* Route: replies.\**

*\*/*

    Route::post('/', [ReplyController::class, 'store'])->name('store');

    Route::get('reply/{id}/{type}', [ReplyController::class, 'redirect'])->name('replyAble');

});

Route::group(['prefix' => 'dashboard', 'as' => 'dashboard.'], function () {

*/\* Name: Notifications*

*\* Url: /dashboard/notifications\**

*\* Route: dashboard.notifications\**

*\*/*

    Route::group(['prefix' => 'notifications', 'as' => 'notifications.'], function () {

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

    });

});

Route::get('dashboard/users', [PageController::class, 'users'])->name('users');

*//Route::get('profile/user/{user:username}', [ProfileController::class, 'show'])->name('profile');*

Route::get('/dashboard/categories/index', [PageController::class, 'categoriesIndex'])->name('categories.index');

Route::get('/dashboard/categories/create', [PageController::class, 'categoriesCreate'])->name('categories.create');

Route::middleware(['auth:sanctum', 'verified'])->get('/dashboard', function () {

*return* view('dashboard');

})->name('dashboard');

<?php

use App\Http\Controllers\Admin\CategoryController;

use App\Http\Controllers\Admin\DashboardController;

use App\Http\Controllers\Admin\TagController;

use Illuminate\Support\Facades\Route;

use Symfony\Component\Routing\Loader\Configurator\Traits\PrefixTrait;

Route::group(['prefix' => 'admin', 'as' => 'admin.'],function(){

*//dashboard*

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

*//Categories*

    Route::group(['prefix' => 'categories', 'as' => 'categories.'],function(){

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

        Route::get('/create',[CategoryController::class,'create'])->name('create');

        Route::post('/',[CategoryController::class,'store'])->name('store');

        Route::get('/edit/{category:slug}',[CategoryController::class,'edit'])->name('edit');

        Route::put('/{category:slug}',[CategoryController::class,'update'])->name('update');

        Route::delete('/{category:slug}',[CategoryController::class,'delete'])->name('delete');

    });

*//Tags*

    Route::group(['prefix' => 'tags', 'as' => 'tags.'],function(){

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

        Route::get('/create',[TagController::class,'create'])->name('create');

        Route::post('/',[TagController::class,'store'])->name('store');

        Route::get('/edit/{tag:slug}',[TagController::class,'edit'])->name('edit');

        Route::put('/{tag:slug}',[TagController::class,'update'])->name('update');

        Route::delete('/{tag:slug}',[TagController::class,'delete'])->name('delete');

    });

});

# OUTPUT:

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application

Description automatically generatedGraphical user interface, text, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Graphical user interface, application

Description automatically generated

Graphical user interface, application

Description automatically generated

Graphical user interface, application

Description automatically generated

Graphical user interface, text, application, Teams

Description automatically generated

A screenshot of a computer

Description automatically generated

Graphical user interface, application, Teams

Description automatically generated