

Software design

EcoTrack: Software Design Document

1. Database Design

1.1 Core Tables Schema

Users Table

```
```sql
CREATE TABLE users (
 id BIGINT UNSIGNED PRIMARY KEY AUTO_INCREMENT,
 username VARCHAR(50) UNIQUE NOT NULL,
 email VARCHAR(255) UNIQUE NOT NULL,
 email_verified_at TIMESTAMP NULL,
 password VARCHAR(255) NOT NULL,
 role ENUM('user', 'moderator', 'admin') DEFAULT 'user',
 status ENUM('active', 'banned', 'suspended') DEFAULT 'active',
 profile_image VARCHAR(255) NULL,
 bio TEXT NULL,
 phone VARCHAR(20) NULL,
 eco_score INT DEFAULT 0,
 language ENUM('en', 'de') DEFAULT 'en',
 created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
 updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE
 CURRENT_TIMESTAMP,

 INDEX idx_role (role),
 INDEX idx_status (status),
 INDEX idx_eco_score (eco_score),
 INDEX idx_username (username),
 FULLTEXT INDEX ft_username_bio (username, bio)
);
```
```

Emissions Table

```
```sql
CREATE TABLE emissions (
 id BIGINT UNSIGNED PRIMARY KEY AUTO_INCREMENT,
 user_id BIGINT UNSIGNED NOT NULL,
 activity_type ENUM('transport', 'energy', 'food', 'waste', 'other') NOT NULL,
 activity_description VARCHAR(255) NOT NULL,
 carbon_amount DECIMAL(8,2) NOT NULL, -- kg CO2e
 activity_date DATE NOT NULL,
 metadata JSON NULL, -- Additional activity details
 created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
 updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE
 CURRENT_TIMESTAMP,

 FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE CASCADE,
 INDEX idx_user_date (user_id, activity_date),
 INDEX idx_activity_type (activity_type),
 INDEX idx_carbon_amount (carbon_amount),
 INDEX idx_activity_date (activity_date)
);
```
```

Posts Table

```
```sql
CREATE TABLE posts (
```

```

id BIGINT UNSIGNED PRIMARY KEY AUTO_INCREMENT,
user_id BIGINT UNSIGNED NOT NULL,
title VARCHAR(255) NOT NULL,
content TEXT NOT NULL,
image_path VARCHAR(255) NULL,
tags JSON NULL,
status ENUM('published', 'draft', 'removed') DEFAULT 'published',
views_count INT DEFAULT 0,
created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,

```

```

FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE CASCADE,
INDEX idx_user_status (user_id, status),
INDEX idx_created_at (created_at),
INDEX idx_status_created (status, created_at),
FULLTEXT INDEX ft_content (title, content)

```

```

);

```

#### #### Reactions Table

```

```sql

```

```

CREATE TABLE reactions (
  id BIGINT UNSIGNED PRIMARY KEY AUTO_INCREMENT,
  user_id BIGINT UNSIGNED NOT NULL,
  reactable_id BIGINT UNSIGNED NOT NULL,
  reactable_type VARCHAR(50) NOT NULL, -- 'post', 'comment'
  reaction_type ENUM('plant', 'recycle', 'fire', 'poop') NOT NULL,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

```

```

FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE CASCADE,
UNIQUE KEY unique_reaction (user_id, reactable_id, reactable_type),
INDEX idx_reactable (reactable_id, reactable_type),
INDEX idx_reaction_type (reaction_type),
INDEX idx_user_reactions (user_id, created_at)

```

```

);

```

Follows Table

```

```sql

```

```

CREATE TABLE follows (
 id BIGINT UNSIGNED PRIMARY KEY AUTO_INCREMENT,
 follower_id BIGINT UNSIGNED NOT NULL,
 following_id BIGINT UNSIGNED NOT NULL,
 created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

```

```

FOREIGN KEY (follower_id) REFERENCES users(id) ON DELETE CASCADE,
FOREIGN KEY (following_id) REFERENCES users(id) ON DELETE CASCADE,
UNIQUE KEY unique_follow (follower_id, following_id),
INDEX idx_follower (follower_id),
INDEX idx_following (following_id),
INDEX idx_created_at (created_at)

```

```

);

```

#### #### Announcements Table

```

```sql

```

```

CREATE TABLE announcements (
  id BIGINT UNSIGNED PRIMARY KEY AUTO_INCREMENT,
  user_id BIGINT UNSIGNED NOT NULL,

```

```

title VARCHAR(255) NOT NULL,
content TEXT NOT NULL,
status ENUM('pending', 'published', 'rejected') DEFAULT 'pending',
published_at TIMESTAMP NULL,
created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,

```

```

FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE CASCADE,
INDEX idx_status (status),
INDEX idx_published_at (published_at),
INDEX idx_status_published (status, published_at)
);

```

Reports Table

```

```sql

```

```

CREATE TABLE reports (
 id BIGINT UNSIGNED PRIMARY KEY AUTO_INCREMENT,
 reporter_id BIGINT UNSIGNED NOT NULL,
 reported_user_id BIGINT UNSIGNED NULL,
 reported_post_id BIGINT UNSIGNED NULL,
 reason ENUM('spam', 'harassment', 'inappropriate', 'other') NOT NULL,
 description TEXT NULL,
 status ENUM('pending', 'reviewed', 'resolved', 'dismissed') DEFAULT 'pending',
 reviewed_by BIGINT UNSIGNED NULL,
 reviewed_at TIMESTAMP NULL,
 action_taken VARCHAR(255) NULL,
 created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

```

```

FOREIGN KEY (reporter_id) REFERENCES users(id) ON DELETE CASCADE,
FOREIGN KEY (reported_user_id) REFERENCES users(id) ON DELETE SET NULL,
FOREIGN KEY (reported_post_id) REFERENCES posts(id) ON DELETE SET NULL,
FOREIGN KEY (reviewed_by) REFERENCES users(id) ON DELETE SET NULL,
INDEX idx_status (status),
INDEX idx_reported_user (reported_user_id),
INDEX idx_reported_post (reported_post_id),
INDEX idx_reviewer (reviewed_by)
);

```

#### #### Notifications Table

```

```sql

```

```

CREATE TABLE notifications (
  id BIGINT UNSIGNED PRIMARY KEY AUTO_INCREMENT,
  user_id BIGINT UNSIGNED NOT NULL,
  type VARCHAR(50) NOT NULL,
  title VARCHAR(255) NOT NULL,
  message TEXT NOT NULL,
  data JSON NULL,
  read_at TIMESTAMP NULL,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

```

```

FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE CASCADE,
INDEX idx_user_unread (user_id, read_at),
INDEX idx_type (type),
INDEX idx_created_at (created_at),
INDEX idx_user_created (user_id, created_at)
);

```

1.2 Database Relationships

...

Users (1) ———< (M) Emissions
Users (1) ———< (M) Posts
Users (1) ———< (M) Reactions
Users (1) ———< (M) Announcements
Users (1) ———< (M) Reports (as reporter)
Users (1) ———< (M) Reports (as reported)
Users (1) ———< (M) Notifications
Users (M) ———< (M) Follows (follower/following)
Posts (1) ———< (M) Reactions (polymorphic)
Posts (1) ———< (M) Reports

...

2. Laravel Backend Design

2.1 Project Structure

...

```
app/  
├── Http/  
│   ├── Controllers/  
│   │   ├── Api/  
│   │   │   ├── AuthController.php  
│   │   │   ├── UserController.php  
│   │   │   ├── EmissionController.php  
│   │   │   ├── PostController.php  
│   │   │   ├── SocialController.php  
│   │   │   ├── NotificationController.php  
│   │   │   └── Admin/  
│   │   │       ├── UserManagementController.php  
│   │   │       ├── ReportController.php  
│   │   │       ├── AnnouncementController.php  
│   │   │       └── AnalyticsController.php  
│   ├── Middleware/  
│   │   ├── RoleMiddleware.php  
│   │   ├── CheckUserStatus.php  
│   │   └── ApiRateLimit.php  
│   └── Requests/  
│       ├── Auth/  
│       │   ├── LoginRequest.php  
│       │   ├── RegisterRequest.php  
│       │   └── ResetPasswordRequest.php  
│       ├── EmissionRequest.php  
│       ├── PostRequest.php  
│       ├── UserUpdateRequest.php  
│       └── ReportRequest.php  
└── Resources/
```

- | |—— UserResource.php
- | |—— PostResource.php
- | |—— EmissionResource.php
- | |—— NotificationResource.php
- | | └ Collections/
 - | | |—— PostCollection.php
 - | | |—— UserCollection.php
- |—— Models/
 - | |—— User.php
 - | |—— Emission.php
 - | |—— Post.php
 - | |—— Reaction.php
 - | |—— Follow.php
 - | |—— Announcement.php
 - | |—— Report.php
 - | | └ Notification.php
- |—— Services/
 - | |—— AuthService.php
 - | |—— EmissionCalculatorService.php
 - | |—— NotificationService.php
 - | |—— FileUploadService.php
 - | |—— AnalyticsService.php
 - | |—— SocialService.php
 - | | └ ModerationService.php
- |—— Jobs/
 - | |—— SendNotification.php
 - | |—— ProcessImageUpload.php
 - | |—— CalculateEcoScore.php
 - | |—— GenerateAnalytics.php
 - | | └ SendEmailNotification.php
- |—— Events/
 - | |—— UserRegistered.php
 - | |—— UserFollowed.php
 - | |—— PostCreated.php
 - | |—— PostReacted.php
 - | | └ UserReported.php
- |—— Listeners/
 - | |—— SendWelcomeNotification.php
 - | |—— SendFollowNotification.php
 - | |—— BroadcastPostUpdate.php
 - | |—— SendReactionNotification.php
 - | | └ NotifyModerators.php
- |—— Policies/

```

|   |—— UserPolicy.php
|   |—— PostPolicy.php
|   |—— ReportPolicy.php
|   |—— AnnouncementPolicy.php
|   |—— Enums/
|       |—— UserRole.php
|       |—— UserStatus.php
|       |—— PostStatus.php
|       |—— ReactionType.php
|       |—— NotificationType.php
...

```

2.2 Core Models

User Model

```

```php
<?php

namespace App\Models;

use Illuminate\Foundation\Auth\User as Authenticatable;
use Laravel\Sanctum\HasApiTokens;
use App\Enums\UserRole;
use App\Enums\UserStatus;

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

 protected $fillable = [
 'username',
 'email',
 'password',
 'role',
 'status',
 'profile_image',
 'bio',
 'phone',
 'eco_score',
 'language'
];

 protected $hidden = [
 'password',
 'remember_token',
];

 protected $casts = [
 'email_verified_at' => 'datetime',
 'role' => UserRole::class,
 'status' => UserStatus::class,
 'eco_score' => 'integer',
];

 // Relationships
 public function emissions()
 {

```

```

 return $this->hasMany(Emission::class);
}

public function posts()
{
 return $this->hasMany(Post::class);
}

public function reactions()
{
 return $this->hasMany(Reaction::class);
}

public function followers()
{
 return $this->belongsToMany(User::class, 'follows', 'following_id', 'follower_id')
 ->withTimestamps();
}

public function following()
{
 return $this->belongsToMany(User::class, 'follows', 'follower_id', 'following_id')
 ->withTimestamps();
}

public function notifications()
{
 return $this->hasMany(Notification::class);
}

public function reportsMade()
{
 return $this->hasMany(Report::class, 'reporter_id');
}

public function reportsAgainst()
{
 return $this->hasMany(Report::class, 'reported_user_id');
}

// Scopes
public function scopeByType($query, string $type)
{
 return $query->where('activity_type', $type);
}

public function scopeByDateRange($query, Carbon $from, Carbon $to)
{
 return $query->whereBetween('activity_date', [$from, $to]);
}

public function scopeThisMonth($query)
{
 return $query->whereMonth('activity_date', now()->month)
 ->whereYear('activity_date', now()->year);
}

public function scopeThisWeek($query)
{
 return $query->whereBetween('activity_date', [

```

```

 now()->startOfWeek(),
 now()->endOfWeek()
]);
}

// Methods
public function getCarbonImpactLevel(): string
{
 return match(true) {
 $this->carbon_amount >= 10 => 'high',
 $this->carbon_amount >= 5 => 'medium',
 $this->carbon_amount >= 1 => 'low',
 default => 'minimal'
 };
}
}
}

```

### ### 2.3 Services Layer

#### #### EmissionCalculatorService

```

`php
<?php

```

```

namespace App\Services;

```

```

class EmissionCalculatorService
{
 private const EMISSION_FACTORS = [
 'transport' => [
 'car_petrol' => 2.31, // kg CO2 per liter
 'car_diesel' => 2.68, // kg CO2 per liter
 'car_electric' => 0.053, // kg CO2 per km (grid average)
 'bus' => 0.089, // kg CO2 per km
 'train' => 0.041, // kg CO2 per km
 'bicycle' => 0.0, // kg CO2 per km
 'walking' => 0.0, // kg CO2 per km
 'flight_domestic' => 0.255, // kg CO2 per km
 'flight_international' => 0.195, // kg CO2 per km
],
 'energy' => [
 'electricity_germany' => 0.366, // kg CO2 per kWh (Germany 2024)
 'electricity_eu' => 0.233, // kg CO2 per kWh (EU average)
 'natural_gas' => 0.184, // kg CO2 per kWh
 'heating_oil' => 0.266, // kg CO2 per kWh
 'renewable' => 0.0, // kg CO2 per kWh
],
 'food' => [
 'beef' => 27.0, // kg CO2 per kg
 'lamb' => 24.5, // kg CO2 per kg
 'pork' => 7.6, // kg CO2 per kg
 'chicken' => 6.9, // kg CO2 per kg
 'fish' => 5.4, // kg CO2 per kg
 'dairy' => 3.2, // kg CO2 per kg
 'vegetables' => 2.0, // kg CO2 per kg
 'fruits' => 1.1, // kg CO2 per kg
 'grains' => 1.4, // kg CO2 per kg
],
 'waste' => [
 'general_waste' => 0.5, // kg CO2 per kg

```



```

 'recycling' => -0.1, // kg CO2 per kg (negative = offset)
 'composting' => -0.2, // kg CO2 per kg (negative = offset)
]
};

```

```

public function calculateEmission(string $activityType, array $data): float
{
 return match($activityType) {
 'transport' => $this->calculateTransportEmission($data),
 'energy' => $this->calculateEnergyEmission($data),
 'food' => $this->calculateFoodEmission($data),
 'waste' => $this->calculateWasteEmission($data),
 default => 0.0
 };
}

```

```

private function calculateTransportEmission(array $data): float
{
 $method = $data['method'];
 $distance = $data['distance'] ?? 0;
 $fuel = $data['fuel_amount'] ?? 0;

 if (in_array($method, ['car_petrol', 'car_diesel'])) {
 return $fuel * self::EMISSION_FACTORS['transport'][$method];
 }

 $factor = self::EMISSION_FACTORS['transport'][$method] ?? 0;
 return $distance * $factor;
}

```

```

private function calculateEnergyEmission(array $data): float
{
 $energyType = $data['energy_type'];
 $amount = $data['amount'] ?? 0; // kWh

 $factor = self::EMISSION_FACTORS['energy'][$energyType] ?? 0;
 return $amount * $factor;
}

```

```

private function calculateFoodEmission(array $data): float
{
 $foodType = $data['food_type'];
 $weight = $data['weight'] ?? 0; // kg

 $factor = self::EMISSION_FACTORS['food'][$foodType] ?? 0;
 return $weight * $factor;
}

```

```

private function calculateWasteEmission(array $data): float
{
 $wasteType = $data['waste_type'];
 $weight = $data['weight'] ?? 0; // kg

 $factor = self::EMISSION_FACTORS['waste'][$wasteType] ?? 0;
 return $weight * $factor;
}

```

```

public function getEcoScoreFromEmissions(float $totalEmissions, float $averageEmissions): int
{
 if ($averageEmissions <= 0) {

```

```

 return 50; // Default score if no comparison data
 }

 $ratio = $totalEmissions / $averageEmissions;
 $score = max(0, min(100, 100 - ($ratio * 50)));

 return (int) round($score);
}

public function getEmissionInsights(User $user, Carbon $from, Carbon $to): array
{
 $emissions = $user->emissions()
 ->whereBetween('activity_date', [$from, $to])
 ->get();

 $byType = $emissions->groupBy('activity_type')
 ->map(fn($group) => $group->sum('carbon_amount'));

 $dailyAverage = $emissions->sum('carbon_amount') / max(1, $from->diffInDays($to));

 $trend = $this->calculateTrend($user, $from, $to);

 return [
 'total_emissions' => $emissions->sum('carbon_amount'),
 'by_type' => $byType,
 'daily_average' => round($dailyAverage, 2),
 'trend' => $trend,
 'top_activities' => $emissions->sortByDesc('carbon_amount')->take(5),
];
}

private function calculateTrend(User $user, Carbon $from, Carbon $to): string
{
 $days = $from->diffInDays($to);
 $previousFrom = $from->copy()->subDays($days);
 $previousTo = $from->copy()->subDay();

 $currentTotal = $user->emissions()
 ->whereBetween('activity_date', [$from, $to])
 ->sum('carbon_amount');

 $previousTotal = $user->emissions()
 ->whereBetween('activity_date', [$previousFrom, $previousTo])
 ->sum('carbon_amount');

 if ($previousTotal == 0) {
 return 'no_data';
 }

 $change = (($currentTotal - $previousTotal) / $previousTotal) * 100;

 return match(true) {
 $change > 10 => 'increasing',
 $change < -10 => 'decreasing',
 default => 'stable'
 };
}
}

```

#### NotificationService

```php  
<?php

namespace App\Services;

use App\Models\User;
use App\Models\Notification;
use App\Events\NotificationSent;
use App\Jobs\SendEmailNotification;

class NotificationService

```
{  
    public function sendNotification(User $user, string $type, array $data): Notification  
    {  
        $notification = $user->notifications()->create([  
            'type' => $type,  
            'title' => $data['title'],  
            'message' => $data['message'],  
            'data' => $data['extra'] ?? null,  
        ]);  
  
        // Broadcast real-time notification  
        broadcast(new NotificationSent($user, $notification));  
  
        // Queue email notification if needed  
        if ($this->shouldSendEmail($type)) {  
            SendEmailNotification::dispatch($user, $notification);  
        }  
  
        return $notification;  
    }  
  
    public function sendFollowNotification(User $follower, User $following): void  
    {  
        $this->sendNotification($following, 'follow', [  
            'title' => 'New Follower',  
            'message' => "{$follower->username} started following you",  
            'extra' => ['follower_id' => $follower->id]  
        ]);  
    }  
  
    public function sendReactionNotification(User $reactor, $reactable, string $reactionType): void  
    {  
        if ($reactable->user_id === $reactor->id) {  
            return; // Don't notify users about their own reactions  
        }  
  
        $this->sendNotification($reactable->user, 'reaction', [  
            'title' => 'New Reaction',  
            'message' => "{$reactor->username} reacted to your post with {$reactionType}",  
            'extra' => [  
                'reactor_id' => $reactor->id,  
                'reactable_type' => get_class($reactable),  
                'reactable_id' => $reactable->id,  
                'reaction_type' => $reactionType  
            ]  
        ]);  
    }  
}
```

```

public function sendReportNotification(User $reporter, $reported): void
{
    $moderators = User::where('role', 'moderator')
        ->orWhere('role', 'admin')
        ->get();

    foreach ($moderators as $moderator) {
        $this->sendNotification($moderator, 'report', [
            'title' => 'New Report',
            'message' => "A new report has been submitted by {$reporter->username}",
            'extra' => [
                'reporter_id' => $reporter->id,
                'reported_type' => get_class($reported),
                'reported_id' => $reported->id
            ]
        ]);
    }
}

public function sendAchievementNotification(User $user, string $achievement, array $details):
void
{
    $this->sendNotification($user, 'achievement', [
        'title' => 'Achievement Unlocked!',
        'message' => "You've earned the '{$achievement}' achievement!",
        'extra' => $details
    ]);
}

public function sendAnnouncementNotification(array $userIds, string $title, string $message):
void
{
    foreach ($userIds as $userId) {
        $user = User::find($userId);
        if ($user) {
            $this->sendNotification($user, 'announcement', [
                'title' => $title,
                'message' => $message
            ]);
        }
    }
}

private function shouldSendEmail(string $type): bool
{
    return in_array($type, ['follow', 'report', 'achievement', 'announcement']);
}

public function markAsRead(User $user, $notificationId): bool
{
    return $user->notifications()
        ->where('id', $notificationId)
        ->whereNull('read_at')
        ->update(['read_at' => now()]) > 0;
}

public function markAllAsRead(User $user): int
{
    return $user->notifications()
        ->whereNull('read_at')

```

```

        ->update(['read_at' => now()]);
    }

    public function getUnreadCount(User $user): int
    {
        return $user->notifications()
            ->whereNull('read_at')
            ->count();
    }
}

```

2.4 API Controllers

AuthController

```

```php
<?php

```

```

namespace App\Http\Controllers\Api;

use App\Http\Controllers\Controller;
use App\Http\Requests\Auth\LoginRequest;
use App\Http\Requests\Auth\RegisterRequest;
use App\Services\AuthService;
use App\Http\Resources\UserResource;

class AuthController extends Controller
{
 public function __construct(
 private AuthService $authService
) {}

 public function register(RegisterRequest $request)
 {
 try {
 $user = $this->authService->register($request->validated());

 return response()->json([
 'success' => true,
 'message' => 'User registered successfully',
 'user' => new UserResource($user),
 'token' => $user->createToken('auth_token')->plainTextToken
], 201);
 } catch (\Exception $e) {
 return response()->json([
 'success' => false,
 'message' => 'Registration failed',
 'error' => $e->getMessage()
], 400);
 }
 }

 public function login(LoginRequest $request)
 {
 try {
 $result = $this->authService->login($request->validated());

 if (!$result['success']) {
 return response()->json([
 'success' => false,

```

```

 'message' => $result['message']
], 401);
}

return response()->json([
 'success' => true,
 'message' => 'Login successful',
 'user' => new UserResource($result['user']),
 'token' => $result['token']
]);
} catch (\Exception $e) {
 return response()->json([
 'success' => false,
 'message' => 'Login failed',
 'error' => $e->getMessage()
], 500);
}
}

public function logout(Request $request)
{
 $request->user()->currentAccessToken()->delete();

 return response()->json([
 'success' => true,
 'message' => 'Logged out successfully'
]);
}

public function me(Request $request)
{
 return response()->json([
 'success' => true,
 'user' => new UserResource($request->user())
]);
}
}
}

```

#### PostController

```

```php
<?php

```

```

namespace App\Http\Controllers\Api;

```

```

use App\Http\Controllers\Controller;
use App\Http\Requests\PostRequest;
use App\Models\Post;
use App\Http\Resources\PostResource;
use App\Http\Resources\Collections\PostCollection;
use App\Services\FileUploadService;
use Illuminate\Http\Request;

```

```

class PostController extends Controller
{
    public function __construct(
        private FileUploadService $fileUploadService
    ) {}

    public function index(Request $request)

```

```

{
    $query = Post::with(['user', 'reactions'])
        ->published()
        ->latest();

    // Apply filters
    if ($request->has('filter')) {
        $query = match($request->filter) {
            'following' => $query->followingFeed($request->user()),
            'popular' => $query->popular(),
            default => $query
        };
    }

    if ($request->has('tags')) {
        $tags = explode(' ', $request->tags);
        $query->byTags($tags);
    }

    $posts = $query->paginate(15);

    return new PostCollection($posts);
}

```

```

public function store(PostRequest $request)
{
    try {
        $data = $request->validated();

        // Handle image upload
        if ($request->hasFile('image')) {
            $data['image_path'] = $this->fileUploadService
                ->uploadImage($request->file('image'), 'posts');
        }

        $data['user_id'] = $request->user()->id;
        $post = Post::create($data);

        // Load relationships for response
        $post->load(['user', 'reactions']);

        return response()->json([
            'success' => true,
            'message' => 'Post created successfully',
            'post' => new PostResource($post)
        ], 201);
    } catch (\Exception $e) {
        return response()->json([
            'success' => false,
            'message' => 'Failed to create post',
            'error' => $e->getMessage()
        ], 500);
    }
}

```

```

public function show(Post $post)
{
    $this->authorize('view', $post);

    $post->incrementViews();
}

```

```

$post->load(['user', 'reactions.user']);

return response()->json([
    'success' => true,
    'post' => new PostResource($post)
]);
}

public function update(PostRequest $request, Post $post)
{
    $this->authorize('update', $post);

    try {
        $data = $request->validated();

        if ($request->hasFile('image')) {
            // Delete old image if exists
            if ($post->image_path) {
                $this->fileUploadService->deleteImage($post->image_path);
            }

            $data['image_path'] = $this->fileUploadService
                ->uploadImage($request->file('image'), 'posts');
        }

        $post->update($data);
        $post->load(['user', 'reactions']);

        return response()->json([
            'success' => true,
            'message' => 'Post updated successfully',
            'post' => new PostResource($post)
        ]);
    } catch (\Exception $e) {
        return response()->json([
            'success' => false,
            'message' => 'Failed to update post',
            'error' => $e->getMessage()
        ], 500);
    }
}

```

```

public function destroy(Post $post)
{
    $this->authorize('delete', $post);

    try {
        if ($post->image_path) {
            $this->fileUploadService->deleteImage($post->image_path);
        }

        $post->delete();

        return response()->json([
            'success' => true,
            'message' => 'Post deleted successfully'
        ]);
    } catch (\Exception $e) {
        return response()->json([
            'success' => false,

```



```

        'message' => 'Failed to delete post',
        'error' => $e->getMessage()
    ], 500);
    }
}

public function react(Request $request, Post $post)
{
    $request->validate([
        'reaction_type' => 'required|in:plant,recycle,fire,poop'
    ]);

    try {
        $user = $request->user();
        $reactionType = $request->reaction_type;

        // Check if user already reacted with this type
        $existingReaction = $post->reactions()
            ->where('user_id', $user->id)
            ->where('reaction_type', $reactionType)
            ->first();

        if ($existingReaction) {
            // Remove reaction
            $existingReaction->delete();
            $action = 'removed';
        } else {
            // Remove any other reaction from this user
            $post->reactions()
                ->where('user_id', $user->id)
                ->delete();

            // Add new reaction
            $post->reactions()->create([
                'user_id' => $user->id,
                'reaction_type' => $reactionType
            ]);
            $action = 'added';
        }

        return response()->json([
            'success' => true,
            'message' => "Reaction {$action} successfully",
            'reaction_counts' => $this->getReactionCounts($post)
        ]);
    } catch (\Exception $e) {
        return response()->json([
            'success' => false,
            'message' => 'Failed to react to post',
            'error' => $e->getMessage()
        ], 500);
    }
}

private function getReactionCounts(Post $post): array
{
    return $post->reactions()
        ->selectRaw('reaction_type, COUNT(*) as count')
        ->groupBy('reaction_type')
        ->pluck('count', 'reaction_type')
}

```

```

    }
    }
  }
}

```

3. Vue.js Frontend Design

3.1 Project Structure

...

```

src/
├── components/
│   ├── auth/
│   │   ├── LoginForm.vue
│   │   ├── RegisterForm.vue
│   │   └── PasswordReset.vue
│   ├── dashboard/
│   │   ├── UserDashboard.vue
│   │   ├── ModeratorDashboard.vue
│   │   └── AdminDashboard.vue
│   ├── emissions/
│   │   ├── EmissionForm.vue
│   │   ├── EmissionList.vue
│   │   ├── EmissionChart.vue
│   │   ├── EmissionAnalytics.vue
│   │   └── EmissionInsights.vue
│   ├── social/
│   │   ├── UserProfile.vue
│   │   ├── FollowButton.vue
│   │   ├── FollowersList.vue
│   │   ├── FollowingList.vue
│   │   └── UserCard.vue
│   ├── blog/
│   │   ├── PostForm.vue
│   │   ├── PostCard.vue
│   │   ├── PostFeed.vue
│   │   ├── ReactionButtons.vue
│   │   ├── PostFilters.vue
│   │   └── PostDetails.vue
│   └── notifications/
│       ├── NotificationDropdown.vue
│       ├── NotificationItem.vue
│       ├── NotificationCenter.vue
│       └── NotificationBadge.vue
│   └── admin/
│       └── UserManagement.vue

```

```
| | | |—— ReportManagement.vue
| | | |—— AnnouncementManagement.vue
| | | |—— SystemAnalytics.vue
| | | |—— ContentModeration.vue
| |—— shared/
| | | |—— BaseModal.vue
| | | |—— BaseButton.vue
| | | |—— BaseInput.vue
| | | |—— BaseTextarea.vue
| | | |—— ImageUpload.vue
| | | |—— LoadingSpinner.vue
| | | |—— ConfirmDialog.vue
| | | |—— Pagination.vue
| | | |—— Toast.vue
|—— layout/
| |—— AppHeader.vue
| |—— AppSidebar.vue
| |—— AppFooter.vue
| |—— AppLayout.vue
|—— composables/
| |—— useAuth.js
| |—— useApi.js
| |—— useNotifications.js
| |—— useWebSocket.js
| |—— usePermissions.js
| |—— useFileUpload.js
| |—— usePagination.js
| |—— useToast.js
|—— stores/
| |—— auth.js
| |—— user.js
| |—— posts.js
| |—— emissions.js
| |—— notifications.js
| |—— social.js
| |—— admin.js
|—— router/
| |—— index.js
| |—— guards.js
| |—— routes/
| | |—— auth.js
| | |—— dashboard.js
| | |—— admin.js
```

```

|   └── public.js
└── utils/
|   ├── api.js
|   ├── helpers.js
|   ├── constants.js
|   ├── validators.js
|   ├── formatters.js
|   └── permissions.js
└── plugins/
|   ├── axios.js
|   ├── i18n.js
|   └── chart.js
└── assets/
    ├── styles/
    |   ├── main.css
    |   ├── components.css
    |   └── utilities.css
    ├── images/
    |   ├── avatars/
    |   └── icons/
    └── locales/
        ├── en.json
        └── de.json
...

```

3.2 Key Vue Composables

useAuth Composable

```

````javascript
// composables/useAuth.js
import { ref, computed } from 'vue'
import { useRouter } from 'vue-router'
import { useAuthStore } from '@stores/auth'
import api from '@utils/api'

export function useAuth() {
 const router = useRouter()
 const authStore = useAuthStore()
 const isLoading = ref(false)
 const error = ref(null)

 const user = computed(() => authStore.user)
 const token = computed(() => authStore.token)
 const isAuthenticated = computed(() => authStore.isAuthenticated)

 const login = async (credentials) => {
 isLoading.value = true
 error.value = null

 try {
 const response = await api.post('/auth/login', credentials)
 const { user, token } = response.data
 }
 }
}

```

```

 authStore.setUser(user)
 authStore.setToken(token)

 // Set default authorization header
 api.defaults.headers.common['Authorization'] = `Bearer ${token}`

 router.push('/dashboard')

 return { success: true }
 } catch (err) {
 error.value = err.response?.data?.message || 'Login failed'
 return { success: false, error: error.value }
 } finally {
 isLoading.value = false
 }
}

const register = async (userData) => {
 isLoading.value = true
 error.value = null

 try {
 const response = await api.post('/auth/register', userData)
 const { user, token } = response.data

 authStore.setUser(user)
 authStore.setToken(token)

 api.defaults.headers.common['Authorization'] = `Bearer ${token}`

 router.push('/dashboard')

 return { success: true }
 } catch (err) {
 error.value = err.response?.data?.message || 'Registration failed'
 return { success: false, error: error.value }
 } finally {
 isLoading.value = false
 }
}

const logout = async () => {
 try {
 await api.post('/auth/logout')
 } catch (err) {
 console.error('Logout error:', err)
 } finally {
 authStore.clearAuth()
 delete api.defaults.headers.common['Authorization']
 router.push('/login')
 }
}

const hasRole = (role) => {
 return user.value?.role === role
}

const hasPermission = (permission) => {
 // Permission checking logic based on user role
 const permissions = {

```

```

 user: ['view_posts', 'create_posts', 'edit_own_posts'],
 moderator: ['view_posts', 'create_posts', 'edit_own_posts', 'moderate_posts', 'ban_users'],
 admin: ['*'] // All permissions
 }

 const userPermissions = permissions[user.value?.role] || []
 return userPermissions.includes('*') || userPermissions.includes(permission)
}

const canModerate = computed(() => {
 return ['admin', 'moderator'].includes(user.value?.role)
})

const isAdmin = computed(() => {
 return user.value?.role === 'admin'
})

return {
 user,
 token,
 isAuthenticated,
 isLoading,
 error,
 login,
 register,
 logout,
 hasRole,
 hasPermission,
 canModerate,
 isAdmin
}
}

```

#### useNotifications Composable

```

````javascript
// composables/useNotifications.js
import { ref, computed, onMounted, onUnmounted } from 'vue'
import { useNotificationStore } from '@stores/notifications'
import { useWebSocket } from './useWebSocket'
import api from '@utils/api'

export function useNotifications() {
  const notificationStore = useNotificationStore()
  const { connectToChannel, disconnect } = useWebSocket()

  const notifications = computed(() => notificationStore.notifications)
  const unreadCount = computed(() => notificationStore.unreadCount)
  const isLoading = ref(false)

  const fetchNotifications = async () => {
    isLoading.value = true
    try {
      const response = await api.get('/notifications')
      notificationStore.setNotifications(response.data.notifications)
    } catch (error) {
      console.error('Failed to fetch notifications:', error)
    } finally {
      isLoading.value = false
    }
  }
}

```

```

}

const markAsRead = async (notificationId) => {
  try {
    await api.put(`/notifications/${notificationId}/read`)
    notificationStore.markAsRead(notificationId)
  } catch (error) {
    console.error('Failed to mark notification as read:', error)
  }
}

const markAllAsRead = async () => {
  try {
    await api.put('/notifications/read-all')
    notificationStore.markAllAsRead()
  } catch (error) {
    console.error('Failed to mark all notifications as read:', error)
  }
}

const deleteNotification = async (notificationId) => {
  try {
    await api.delete(`/notifications/${notificationId}`)
    notificationStore.removeNotification(notificationId)
  } catch (error) {
    console.error('Failed to delete notification:', error)
  }
}

const connectRealTime = (userId) => {
  connectToChannel(`notifications.${userId}`, (data) => {
    notificationStore.addNotification(data.notification)
  })
}

onMounted(() => {
  fetchNotifications()
})

onUnmounted(() => {
  disconnect()
})

return {
  notifications,
  unreadCount,
  isLoading,
  fetchNotifications,
  markAsRead,
  markAllAsRead,
  deleteNotification,
  connectRealTime
}
}

```

3.3 Pinia Stores

Auth Store

```
```javascript
```

```

// stores/auth.js
import { defineStore } from 'pinia'

export const useAuthStore = defineStore('auth', {
 state: () => ({
 user: null,
 token: localStorage.getItem('token'),
 isLoading: false,
 error: null
 }),

 getters: {
 isAuthenticated: (state) => !!state.token,
 isAdmin: (state) => state.user?.role === 'admin',
 isModerator: (state) => ['admin', 'moderator'].includes(state.user?.role),
 userName: (state) => state.user?.username || '',
 userRole: (state) => state.user?.role || 'user'
 },

 actions: {
 setUser(user) {
 this.user = user
 },

 setToken(token) {
 this.token = token
 localStorage.setItem('token', token)
 },

 clearAuth() {
 this.user = null
 this.token = null
 localStorage.removeItem('token')
 },

 updateUser(userData) {
 if (this.user) {
 this.user = { ...this.user, ...userData }
 }
 },

 setLoading(loading) {
 this.isLoading = loading
 },

 setError(error) {
 this.error = error
 }
 }
})

```

#### Posts Store

```

````javascript
// stores/posts.js
import { defineStore } from 'pinia'
import api from '@/utils/api'

export const usePostsStore = defineStore('posts', {
  state: () => ({

```



```

posts: [],
currentPost: null,
filters: {
  type: 'latest', // latest, popular, following
  tags: []
},
pagination: {
  currentPage: 1,
  lastPage: 1,
  perPage: 15,
  total: 0
},
isLoading: false,
error: null
}),

getters: {
  filteredPosts: (state) => {
    let filtered = [...state.posts]

    if (state.filters.tags.length > 0) {
      filtered = filtered.filter(post =>
        post.tags?.some(tag => state.filters.tags.includes(tag))
      )
    }

    return filtered
  },

  hasMorePages: (state) => {
    return state.pagination.currentPage < state.pagination.lastPage
  }
},

actions: {
  async fetchPosts(page = 1, filters = {}) {
    this.isLoading = true
    this.error = null

    try {
      const params = {
        page,
        filter: filters.type || this.filters.type,
        tags: filters.tags?.join(',') || this.filters.tags.join(',')
      }

      const response = await api.get('/posts', { params })
      const { data, meta } = response.data

      if (page === 1) {
        this.posts = data
      } else {
        this.posts.push(...data)
      }

      this.pagination = {
        currentPage: meta.current_page,
        lastPage: meta.last_page,
        perPage: meta.per_page,
        total: meta.total
      }
    }
  }
}

```

```

    }

    this.filters = { ...this.filters, ...filters }

    } catch (error) {
      this.error = error.response?.data?.message || 'Failed to fetch posts'
    } finally {
      this.isLoading = false
    }
  },

```

```

async createPost(postData) {
  try {
    const response = await api.post('/posts', postData, {
      headers: {
        'Content-Type': 'multipart/form-data'
      }
    })

    const newPost = response.data.post
    this.posts.unshift(newPost)

    return { success: true, post: newPost }
  } catch (error) {
    return {
      success: false,
      error: error.response?.data?.message || 'Failed to create post'
    }
  }
},

```

```

async updatePost(postId, postData) {
  try {
    const response = await api.put(`/posts/${postId}`, postData, {
      headers: {
        'Content-Type': 'multipart/form-data'
      }
    })

    const updatedPost = response.data.post
    const index = this.posts.findIndex(post => post.id === postId)
    if (index !== -1) {
      this.posts[index] = updatedPost
    }

    return { success: true, post: updatedPost }
  } catch (error) {
    return {
      success: false,
      error: error.response?.data?.message || 'Failed to update post'
    }
  }
},

```

```

async deletePost(postId) {
  try {
    await api.delete(`/posts/${postId}`)
    this.posts = this.posts.filter(post => post.id !== postId)
    return { success: true }
  } catch (error) {
  }
}

```

```

    return {
      success: false,
      error: error.response?.data?.message || 'Failed to delete post'
    }
  }
},

async reactToPost(postId, reactionType) {
  try {
    const response = await api.post(`/posts/${postId}/react`, {
      reaction_type: reactionType
    })

    // Update post reactions in local state
    const post = this.posts.find(p => p.id === postId)
    if (post) {
      post.reaction_counts = response.data.reaction_counts
      post.user_reaction = reactionType
    }

    return { success: true }
  } catch (error) {
    return {
      success: false,
      error: error.response?.data?.message || 'Failed to react to post'
    }
  }
},

setFilters(filters) {
  this.filters = { ...this.filters, ...filters }
},

resetPosts() {
  this.posts = []
  this.pagination.currentPage = 1
},

addPost(post) {
  this.posts.unshift(post)
},

updatePostInList(postId, updates) {
  const index = this.posts.findIndex(post => post.id === postId)
  if (index !== -1) {
    this.posts[index] = { ...this.posts[index], ...updates }
  }
}
})
```

```

#### Notifications Store

```javascript

// stores/notifications.js

import { defineStore } from 'pinia'

```

export const useNotificationStore = defineStore('notifications', {
  state: () => ({
    notifications: [],
  })
})

```

```

    isLoading: false,
    error: null
  }},

  getters: {
    unreadCount: (state) => {
      return state.notifications.filter(n => !n.read_at).length
    },

    unreadNotifications: (state) => {
      return state.notifications.filter(n => !n.read_at)
    },

    recentNotifications: (state) => {
      return state.notifications.slice(0, 10)
    }
  },

  actions: {
    setNotifications(notifications) {
      this.notifications = notifications
    },

    addNotification(notification) {
      this.notifications.unshift(notification)
    },

    markAsRead(notificationId) {
      const notification = this.notifications.find(n => n.id === notificationId)
      if (notification) {
        notification.read_at = new Date().toISOString()
      }
    },

    markAllAsRead() {
      const now = new Date().toISOString()
      this.notifications.forEach(notification => {
        if (!notification.read_at) {
          notification.read_at = now
        }
      })
    },

    removeNotification(notificationId) {
      this.notifications = this.notifications.filter(n => n.id !== notificationId)
    },

    clearAll() {
      this.notifications = []
    }
  }
})

```

3.4 Key Vue Components

PostCard Component

```

``vue
<!-- components/blog/PostCard.vue -->
<template>

```

```

<div class="bg-white dark:bg-gray-800 rounded-lg shadow-md p-6 mb-4">
  <!-- Post Header -->
  <div class="flex items-center justify-between mb-4">
    <div class="flex items-center space-x-3">
      
    </div>
    <div>
      <h4 class="font-semibold text-gray-900 dark:text-white">
        {{ post.user.username }}
      </h4>
      <p class="text-sm text-gray-500 dark:text-gray-400">
        {{ formatDate(post.created_at) }}
      </p>
    </div>
  </div>

  <!-- Post Actions Menu -->
  <div class="relative" v-if="canEditPost">
    <button @click="showMenu = !showMenu" class="text-gray-400 hover:text-gray-600">
      <svg class="w-5 h-5" fill="currentColor" viewBox="0 0 20 20">
        <path d="M10 6a2 2 0 110-4 2 2 0 010 4zM10 12a2 2 0 110-4 2 2 0 010 4zM10 18a2 2 0
110-4 2 2 0 010 4z"></path>
      </svg>
    </button>

    <div v-if="showMenu" class="absolute right-0 mt-2 w-48 bg-white rounded-md shadow-lg
z-10">
      <button @click="editPost" class="block px-4 py-2 text-sm text-gray-700 hover:bg-
gray-100 w-full text-left">
        Edit Post
      </button>
      <button @click="deletePost" class="block px-4 py-2 text-sm text-red-600 hover:bg-
gray-100 w-full text-left">
        Delete Post
      </button>
    </div>
  </div>

  <!-- Post Content -->
  <div class="mb-4">
    <h3 class="text-lg font-semibold text-gray-900 dark:text-white mb-2">
      {{ post.title }}
    </h3>
    <p class="text-gray-700 dark:text-gray-300 whitespace-pre-wrap">
      {{ post.content }}
    </p>
  </div>

  <!-- Post Image -->
  <div v-if="post.image_path" class="mb-4">
    
  </div>

```

```

<!-- Post Tags -->
<div v-if="post.tags && post.tags.length" class="mb-4">
  <span
    v-for="tag in post.tags"
    :key="tag"
    class="inline-block bg-green-100 text-green-800 text-xs px-2 py-1 rounded-full mr-2 mb-1"
  >
    #{{ tag }}
  </span>
</div>

```

```

<!-- Reaction Buttons -->
<ReactionButtons
  :post="post"
  @reaction-changed="handleReactionChanged"
/>

```

```

<!-- Post Stats -->
<div class="flex items-center justify-between mt-4 text-sm text-gray-500">
  <span>{{ post.views_count }} views</span>
  <span>{{ totalReactions }} reactions</span>
</div>
</div>
</template>

```

```

<script setup>
import { ref, computed } from 'vue'
import { useAuth } from '@composables/useAuth'
import { usePostsStore } from '@stores/posts'
import ReactionButtons from './ReactionButtons.vue'
import { formatDate } from '@utils/formatters'

```

```

const props = defineProps({
  post: {
    type: Object,
    required: true
  }
})

```

```

const emit = defineEmits(['edit', 'delete'])

```

```

const { user, hasPermission } = useAuth()
const postsStore = usePostsStore()

```

```

const showMenu = ref(false)

```

```

const canEditPost = computed(() => {
  return user.value?.id === props.post.user.id || hasPermission('moderate_posts')
})

```

```

const totalReactions = computed(() => {
  if (!props.post.reaction_counts) return 0
  return Object.values(props.post.reaction_counts).reduce((sum, count) => sum + count, 0)
})

```

```

const editPost = () => {
  showMenu.value = false
  emit('edit', props.post)
}

```

```

const deletePost = async () => {
  if (confirm('Are you sure you want to delete this post?')) {
    showMenu.value = false
    const result = await postsStore.deletePost(props.post.id)
    if (result.success) {
      emit('delete', props.post.id)
    }
  }
}

const handleReactionChanged = (reactionData) => {
  // Update local post data
  if (props.post.reaction_counts) {
    props.post.reaction_counts = reactionData.reaction_counts
  }
}
</script>

```

ReactionButtons Component

```

``vue
<!-- components/blog/ReactionButtons.vue -->
<template>
  <div class="flex items-center space-x-4">
    <button
      v-for="reaction in reactions"
      :key="reaction.type"
      @click="toggleReaction(reaction.type)"
      :class="[
        'flex items-center space-x-1 px-3 py-1 rounded-full text-sm transition-all duration-200',
        isUserReaction(reaction.type)
          ? 'bg-green-100 text-green-700 border border-green-300'
          : 'bg-gray-100 text-gray-600 hover:bg-gray-200'
      ]"
      :disabled="isLoading"
    >
      <span class="text-lg">{{ reaction.emoji }}</span>
      <span>{{ getReactionCount(reaction.type) }}</span>
    </button>
  </div>
</template>

<script setup>
import { ref, computed } from 'vue'
import { useAuth } from '@composables/useAuth'
import { usePostsStore } from '@stores/posts'
import { useToast } from '@composables/useToast'

const props = defineProps({
  post: {
    type: Object,
    required: true
  }
})

const emit = defineEmits(['reaction-changed'])

const { user, isAuthenticated } = useAuth()
const postsStore = usePostsStore()

```

```

const { showToast } = useToast()

const isLoading = ref(false)

const reactions = [
  { type: 'plant', emoji: '🌱', label: 'Plant' },
  { type: 'recycle', emoji: '♻️', label: 'Recycle' },
  { type: 'fire', emoji: '🔥', label: 'Fire' },
  { type: 'poop', emoji: '💩', label: 'Poop' }
]

const getReactionCount = (type) => {
  return props.post.reaction_counts?.[type] || 0
}

const isUserReaction = (type) => {
  return props.post.user_reaction === type
}

const toggleReaction = async (reactionType) => {
  if (!isAuthenticated.value) {
    showToast('Please login to react to posts', 'warning')
    return
  }

  if (isLoading.value) return

  isLoading.value = true

  try {
    const result = await postsStore.reactToPost(props.post.id, reactionType)

    if (result.success) {
      emit('reaction-changed', result)
      showToast('Reaction updated!', 'success')
    } else {
      showToast(result.error, 'error')
    }
  } catch (error) {
    showToast('Failed to update reaction', 'error')
  } finally {
    isLoading.value = false
  }
}
</script>

```

EmissionForm Component

```

<!-- vue
components/emissions/EmissionForm.vue -->
<template>
  <form @submit.prevent="submitForm" class="space-y-6">
    <div class="bg-white dark:bg-gray-800 rounded-lg shadow-md p-6">
      <h3 class="text-lg font-semibold text-gray-900 dark:text-white mb-4">
        Log Carbon Emission
      </h3>

      <!-- Activity Type Selection -->

```



```

<div class="mb-4">
  <label class="block text-sm font-medium text-gray-700 dark:text-gray-300 mb-2">
    Activity Type
  </label>
  <select
    v-model="form.activity_type"
    @change="resetActivityData"
    class="w-full px-3 py-2 border border-gray-300 rounded-md focus:outline-none
focus:ring-2 focus:ring-green-500"
    required
  >
    <option value="">Select activity type</option>
    <option value="transport">Transportation</option>
    <option value="energy">Energy Usage</option>
    <option value="food">Food Consumption</option>
    <option value="waste">Waste Management</option>
    <option value="other">Other</option>
  </select>
</div>

<!-- Dynamic Form Fields Based on Activity Type -->
<div v-if="form.activity_type === 'transport'" class="space-y-4">
  <div>
    <label class="block text-sm font-medium text-gray-700 dark:text-gray-300 mb-2">
      Transportation Method
    </label>
    <select v-model="form.transport_method" class="form-select" required>
      <option value="">Select method</option>
      <option value="car_petrol">Car (Petrol)</option>
      <option value="car_diesel">Car (Diesel)</option>
      <option value="car_electric">Car (Electric)</option>
      <option value="bus">Bus</option>
      <option value="train">Train</option>
      <option value="bicycle">Bicycle</option>
      <option value="walking">Walking</option>
      <option value="flight_domestic">Flight (Domestic)</option>
      <option value="flight_international">Flight (International)</option>
    </select>
  </div>

  <div v-if="needsFuelInput">
    <label class="block text-sm font-medium text-gray-700 dark:text-gray-300 mb-2">
      Fuel Amount (Liters)
    </label>
    <input
      v-model.number="form.fuel_amount"
      type="number"
      step="0.1"
      min="0"
      class="form-input"
      required
    >
  </div>

  <div v-if="needsDistanceInput">
    <label class="block text-sm font-medium text-gray-700 dark:text-gray-300 mb-2">
      Distance (Kilometers)
    </label>
    <input
      v-model.number="form.distance"

```

```

        type="number"
        step="0.1"
        min="0"
        class="form-input"
        required
    >
</div>
</div>

<div v-else-if="form.activity_type === 'energy'" class="space-y-4">
    <div>
        <label class="block text-sm font-medium text-gray-700 dark:text-gray-300 mb-2">
            Energy Type
        </label>
        <select v-model="form.energy_type" class="form-select" required>
            <option value="">Select energy type</option>
            <option value="electricity_germany">Electricity (Germany)</option>
            <option value="electricity_eu">Electricity (EU Average)</option>
            <option value="natural_gas">Natural Gas</option>
            <option value="heating_oil">Heating Oil</option>
            <option value="renewable">Renewable Energy</option>
        </select>
    </div>

    <div>
        <label class="block text-sm font-medium text-gray-700 dark:text-gray-300 mb-2">
            Amount (kWh)
        </label>
        <input
            v-model.number="form.amount"
            type="number"
            step="0.1"
            min="0"
            class="form-input"
            required
        >
    </div>
</div>

<div v-else-if="form.activity_type === 'food'" class="space-y-4">
    <div>
        <label class="block text-sm font-medium text-gray-700 dark:text-gray-300 mb-2">
            Food Type
        </label>
        <select v-model="form.food_type" class="form-select" required>
            <option value="">Select food type</option>
            <option value="beef">Beef</option>
            <option value="lamb">Lamb</option>
            <option value="pork">Pork</option>
            <option value="chicken">Chicken</option>
            <option value="fish">Fish</option>
            <option value="dairy">Dairy Products</option>
            <option value="vegetables">Vegetables</option>
            <option value="fruits">Fruits</option>
            <option value="grains">Grains</option>
        </select>
    </div>

    <div>
        <label class="block text-sm font-medium text-gray-700 dark:text-gray-300 mb-2">

```

```

    Weight (kg)
  </label>
  <input
    v-model.number="form.weight"
    type="number"
    step="0.1"
    min="0"
    class="form-input"
    required
  >
</div>
</div>

```

```

<!-- Common Fields -->
<div class="space-y-4">
  <div>
    <label class="block text-sm font-medium text-gray-700 dark:text-gray-300 mb-2">
      Activity Description
    </label>
    <input
      v-model="form.activity_description"
      type="text"
      class="form-input"
      placeholder="Describe your activity..."
      required
    >
  </div>

```

```

  <div>
    <label class="block text-sm font-medium text-gray-700 dark:text-gray-300 mb-2">
      Date
    </label>
    <input
      v-model="form.activity_date"
      type="date"
      class="form-input"
      :max="today"
      required
    >
  </div>
</div>

```

```

<!-- Calculated Carbon Amount Display -->
<div v-if="calculatedCarbon > 0" class="mt-4 p-4 bg-green-50 border border-green-200
rounded-md">
  <p class="text-sm text-green-800">
    <strong>Estimated Carbon Footprint:</strong>
    {{ calculatedCarbon.toFixed(2) }} kg CO2e
  </p>
</div>

```

```

<!-- Submit Button -->
<div class="mt-6">
  <button
    type="submit"
    :disabled="isLoading || !isFormValid"
    class="w-full bg-green-600 text-white py-2 px-4 rounded-md hover:bg-green-700
focus:outline-none focus:ring-2 focus:ring-green-500 disabled:opacity-50 disabled:cursor-not-
allowed"
  >

```

```

        <span v-if="isLoading">Saving...</span>
        <span v-else>Log Emission</span>
      </button>
    </div>
  </div>
</form>
</template>

```

```

<script setup>
import { ref, computed, watch } from 'vue'
import { useEmissionsStore } from '@stores/emissions'
import { useToast } from '@composables/useToast'

const emit = defineEmits(['emission-logged'])

const emissionsStore = useEmissionsStore()
const { showToast } = useToast()

const isLoading = ref(false)
const today = new Date().toISOString().split('T')[0]

const form = ref({
  activity_type: '',
  activity_description: '',
  activity_date: today,
  // Transport specific
  transport_method: '',
  distance: 0,
  fuel_amount: 0,
  // Energy specific
  energy_type: '',
  amount: 0,
  // Food specific
  food_type: '',
  weight: 0,
  // Waste specific
  waste_type: '',
})

const needsFuelInput = computed(() => {
  return ['car_petrol', 'car_diesel'].includes(form.value.transport_method)
})

const needsDistanceInput = computed(() => {
  return !needsFuelInput.value && form.value.transport_method && form.value.activity_type ===
  'transport'
})

const calculatedCarbon = ref(0)

const isValidForm = computed(() => {
  return form.value.activity_type &&
    form.value.activity_description &&
    form.value.activity_date &&
    calculatedCarbon.value >= 0
})

// Watch for form changes to calculate carbon footprint
watch(form, async (newForm) => {
  if (newForm.activity_type && isValidForm(newForm)) {

```

```

    try {
      calculatedCarbon.value = await calculateCarbon(newForm)
    } catch (error) {
      calculatedCarbon.value = 0
    }
  } else {
    calculatedCarbon.value = 0
  }
}, { deep: true })

```

```

const isFormDataComplete = (formData) => {
  switch (formData.activity_type) {
    case 'transport':
      return formData.transport_method &&
        (needsFuelInput.value ? formData.fuel_amount > 0 : formData.distance > 0)
    case 'energy':
      return formData.energy_type && formData.amount > 0
    case 'food':
      return formData.food_type && formData.weight > 0
    case 'waste':
      return formData.waste_type && formData.weight > 0
    default:
      return true
  }
}

```

```

const calculateCarbon = async (formData) => {
  // This would typically call a service or API to calculate carbon footprint
  // For now, we'll use a simplified calculation
  const factors = {
    transport: {
      car_petrol: 2.31,
      car_diesel: 2.68,
      car_electric: 0.053,
      bus: 0.089,
      train: 0.041,
      bicycle: 0,
      walking: 0,
      flight_domestic: 0.255,
      flight_international: 0.195
    },
    energy: {
      electricity_germany: 0.366,
      electricity_eu: 0.233,
      natural_gas: 0.184,
      heating_oil: 0.266,
      renewable: 0
    },
    food: {
      beef: 27.0,
      lamb: 24.5,
      pork: 7.6,
      chicken: 6.9,
      fish: 5.4,
      dairy: 3.2,
      vegetables: 2.0,
      fruits: 1.1,
      grains: 1.4
    }
  }
}

```

```

switch (formData.activity_type) {
  case 'transport':
    if (needsFuelInput.value) {
      return formData.fuel_amount * (factors.transport[formData.transport_method] || 0)
    } else {
      return formData.distance * (factors.transport[formData.transport_method] || 0)
    }
  case 'energy':
    return formData.amount * (factors.energy[formData.energy_type] || 0)
  case 'food':
    return formData.weight * (factors.food[formData.food_type] || 0)
  default:
    return 0
}
}

```

```

const resetActivityData = () => {
  form.value.transport_method = ''
  form.value.distance = 0
  form.value.fuel_amount = 0
  form.value.energy_type = ''
  form.value.amount = 0
  form.value.food_type = ''
  form.value.weight = 0
  form.value.waste_type = ''
  calculatedCarbon.value = 0
}

```

```

const submitForm = async () => {
  if (!isFormValid.value) return

```

```

  isLoading.value = true

```

```

  try {
    const emissionData = {
      ...form.value,
      carbon_amount: calculatedCarbon.value,
      metadata: getMetadata()
    }
  }

```

```

  const result = await emissionsStore.createEmission(emissionData)

```

```

  if (result.success) {
    showToast('Emission logged successfully!', 'success')
    resetForm()
    emit('emission-logged', result.emission)
  } else {
    showToast(result.error, 'error')
  }

```

```

} catch (error) {
  showToast('Failed to log emission', 'error')
} finally {
  isLoading.value = false
}
}

```

```

const getMetadata = () => {
  const metadata = {}

```

```

if (form.value.activity_type === 'transport') {
  metadata.method = form.value.transport_method
  if (needsFuelInput.value) {
    metadata.fuel_amount = form.value.fuel_amount
  } else {
    metadata.distance = form.value.distance
  }
} else if (form.value.activity_type === 'energy') {
  metadata.energy_type = form.value.energy_type
  metadata.amount = form.value.amount
} else if (form.value.activity_type === 'food') {
  metadata.food_type = form.value.food_type
  metadata.weight = form.value.weight
}

return metadata
}

```

```

const resetForm = () => {
  form.value = {
    activity_type: '',
    activity_description: '',
    activity_date: today,
    transport_method: '',
    distance: 0,
    fuel_amount: 0,
    energy_type: '',
    amount: 0,
    food_type: '',
    weight: 0,
    waste_type: '',
  }
  calculatedCarbon.value = 0
}
</script>

```

```

<style scoped>
.form-input {
  @apply w-full px-3 py-2 border border-gray-300 rounded-md focus:outline-none focus:ring-2
  focus:ring-green-500 dark:bg-gray-700 dark:border-gray-600 dark:text-white;
}

.form-select {
  @apply w-full px-3 py-2 border border-gray-300 rounded-md focus:outline-none focus:ring-2
  focus:ring-green-500 dark:bg-gray-700 dark:border-gray-600 dark:text-white;
}
</style>

```

4. API Design & Routing

4.1 Laravel API Routes

```

```php
// routes/api.php
<?php

```

```

use Illuminate\Support\Facades\Route;
use App\Http\Controllers\Api\AuthController;
use App\Http\Controllers\ApiUserController;

```

```

use App\Http\Controllers\Api\PostController;
use App\Http\Controllers\Api\EmissionController;
use App\Http\Controllers\Api\SocialController;
use App\Http\Controllers\Api\NotificationController;
use App\Http\Controllers\Api\Admin\UserManagementController;
use App\Http\Controllers\Api\Admin\ReportController;
use App\Http\Controllers\Api\Admin\AnnouncementController;

/*
|-----
| Authentication Routes
|-----
*/
Route::prefix('auth')->group(function () {
 Route::post('register', [AuthController::class, 'register']);
 Route::post('login', [AuthController::class, 'login']);
 Route::post('forgot-password', [AuthController::class, 'forgotPassword']);
 Route::post('reset-password', [AuthController::class, 'resetPassword']);

 Route::middleware('auth:sanctum')->group(function () {
 Route::post('logout', [AuthController::class, 'logout']);
 Route::get('me', [AuthController::class, 'me']);
 Route::post('refresh', [AuthController::class, 'refresh']);
 });
});

/*
|-----
| Protected Routes
|-----
*/
Route::middleware(['auth:sanctum', 'check.user.status'])->group(function () {

 /*
 |-----
 | User Management Routes
 |-----
 */
 Route::prefix('user')->group(function () {
 Route::get('/', [UserController::class, 'show']);
 Route::put('/', [UserController::class, 'update']);
 Route::post('avatar', [UserController::class, 'uploadAvatar']);
 Route::delete('avatar', [UserController::class, 'removeAvatar']);
 Route::get('analytics', [UserController::class, 'getAnalytics']);
 });

 Route::prefix('users')->group(function () {
 Route::get('/', [UserController::class, 'index']);
 Route::get('{user}', [UserController::class, 'profile']);
 Route::get('{user}/followers', [SocialController::class, 'getFollowers']);
 Route::get('{user}/following', [SocialController::class, 'getFollowing']);
 Route::post('{user}/follow', [SocialController::class, 'follow']);
 Route::delete('{user}/follow', [SocialController::class, 'unfollow']);
 Route::post('{user}/report', [SocialController::class, 'reportUser']);
 });

 /*
 |-----
 | Emissions Routes
 |-----
 */

```



```

*/
Route::prefix('emissions')->group(function () {
 Route::get('/', [EmissionController::class, 'index']);
 Route::post('/', [EmissionController::class, 'store']);
 Route::get('{emission}', [EmissionController::class, 'show']);
 Route::put('{emission}', [EmissionController::class, 'update']);
 Route::delete('{emission}', [EmissionController::class, 'destroy']);
 Route::get('analytics/summary', [EmissionController::class, 'getSummary']);
 Route::get('analytics/insights', [EmissionController::class, 'getInsights']);
 Route::get('analytics/trends', [EmissionController::class, 'getTrends']);
 Route::post('calculate', [EmissionController::class, 'calculateEmission']);
});

/*
|-----
| Posts & Blog Routes
|-----
*/
Route::prefix('posts')->group(function () {
 Route::get('/', [PostController::class, 'index']);
 Route::post('/', [PostController::class, 'store']);
 Route::get('{post}', [PostController::class, 'show']);
 Route::put('{post}', [PostController::class, 'update']);
 Route::delete('{post}', [PostController::class, 'destroy']);
 Route::post('{post}/react', [PostController::class, 'react']);
 Route::delete('{post}/react', [PostController::class, 'unreact']);
 Route::post('{post}/report', [PostController::class, 'report']);
 Route::get('feed/following', [PostController::class, 'followingFeed']);
 Route::get('feed/popular', [PostController::class, 'popularFeed']);
});

/*
|-----
| Notifications Routes
|-----
*/
Route::prefix('notifications')->group(function () {
 Route::get('/', [NotificationController::class, 'index']);
 Route::put('{notification}/read', [NotificationController::class, 'markAsRead']);
 Route::put('read-all', [NotificationController::class, 'markAllAsRead']);
 Route::delete('{notification}', [NotificationController::class, 'destroy']);
 Route::get('unread-count', [NotificationController::class, 'getUnreadCount']);
});

/*
|-----
| Announcements Routes
|-----
*/
Route::prefix('announcements')->group(function () {
 Route::get('/', [AnnouncementController::class, 'index']);
 Route::get('{announcement}', [AnnouncementController::class, 'show']);

 // Moderator and Admin routes
 Route::middleware('role:moderator,admin')->group(function () {
 Route::post('/', [AnnouncementController::class, 'store']);
 Route::put('{announcement}', [AnnouncementController::class, 'update']);
 });
});

```

```

/*
|-----
| Moderator Routes
|-----
*/
Route::middleware('role:moderator,admin')->prefix('moderator')->group(function () {
 Route::get('reports', [ReportController::class, 'index']);
 Route::put('reports/{report}', [ReportController::class, 'update']);
 Route::get('posts/reported', [PostController::class, 'getReportedPosts']);
 Route::put('posts/{post}/moderate', [PostController::class, 'moderate']);
 Route::get('users/reported', [UserController::class, 'getReportedUsers']);
 Route::put('users/{user}/moderate', [UserController::class, 'moderate']);
});

/*
|-----
| Admin Routes
|-----
*/
Route::middleware('role:admin')->prefix('admin')->group(function () {
 // User Management
 Route::get('users', [UserManagementController::class, 'index']);
 Route::get('users/{user}', [UserManagementController::class, 'show']);
 Route::put('users/{user}/role', [UserManagementController::class, 'updateRole']);
 Route::put('users/{user}/status', [UserManagementController::class, 'updateStatus']);
 Route::delete('users/{user}', [UserManagementController::class, 'destroy']);

 // Announcement Management
 Route::get('announcements/pending', [AnnouncementController::class, 'pending']);
 Route::put('announcements/{announcement}/approve', [AnnouncementController::class,
'approve']);
 Route::put('announcements/{announcement}/reject', [AnnouncementController::class,
'reject']);

 // System Analytics
 Route::get('analytics/overview', [UserManagementController::class, 'getSystemAnalytics']);
 Route::get('analytics/users', [UserManagementController::class, 'getUserAnalytics']);
 Route::get('analytics/content', [UserManagementController::class, 'getContentAnalytics']);
 Route::get('analytics/emissions', [UserManagementController::class,
'getEmissionAnalytics']);

 // Reports Management
 Route::get('reports/all', [ReportController::class, 'getAllReports']);
 Route::get('reports/statistics', [ReportController::class, 'getStatistics']);
});

/*
|-----
| Public Routes
|-----
*/
Route::prefix('public')->group(function () {
 Route::get('posts', [PostController::class, 'publicFeed']);
 Route::get('announcements', [AnnouncementController::class, 'publicAnnouncements']);
 Route::get('statistics', [UserController::class, 'getPublicStatistics']);
});

```

### ### 4.2 Request Validation Classes

#### PostRequest

```
```php
<?php
```

```
namespace App\Http\Requests;
```

```
use Illuminate\Foundation\Http\FormRequest;
```

```
class PostRequest extends FormRequest
```

```
{
```

```
    public function authorize(): bool
```

```
    {
```

```
        return auth()->check();
```

```
    }
```

```
    public function rules(): array
```

```
    {
```

```
        $rules = [
```

```
            'title' => 'required|string|max:255',
```

```
            'content' => 'required|string|max:5000',
```

```
            'tags' => 'nullable|array|max:10',
```

```
            'tags.*' => 'string|max:50|regex:/^[a-zA-Z0-9_]+$/',
```

```
            'status' => 'nullable|in:published,draft'
```

```
        ];
```

```
        if ($this->hasFile('image')) {
```

```
            $rules['image'] = 'image|mimes:jpeg,png,jpg,gif|max:5120'; // 5MB max
```

```
        }
```

```
        return $rules;
```

```
    }
```

```
    public function messages(): array
```

```
    {
```

```
        return [
```

```
            'title.required' => 'Post title is required',
```

```
            'title.max' => 'Post title cannot exceed 255 characters',
```

```
            'content.required' => 'Post content is required',
```

```
            'content.max' => 'Post content cannot exceed 5000 characters',
```

```
            'tags.max' => 'You can add a maximum of 10 tags',
```

```
            'tags.*.regex' => 'Tags can only contain letters, numbers, and underscores',
```

```
            'image.image' => 'File must be an image',
```

```
            'image.mimes' => 'Image must be a JPEG, PNG, JPG, or GIF file',
```

```
            'image.max' => 'Image size cannot exceed 5MB',
```

```
        ];
```

```
    }
```

```
    protected function prepareForValidation(): void
```

```
    {
```

```
        if ($this->has('tags') && is_string($this->tags)) {
```

```
            $this->merge([
```

```
                'tags' => array_filter(array_map('trim', explode(',', $this->tags)))
```

```
            ]);
```

```
        }
```

```
    }
```

```
}```
```

EmissionRequest

```

`php
<?php

namespace App\Http\Requests;

use Illuminate\Foundation\Http\FormRequest;

class EmissionRequest extends FormRequest
{
    public function authorize(): bool
    {
        return auth()->check();
    }

    public function rules(): array
    {
        return [
            'activity_type' => 'required|in:transport,energy,food,waste,other',
            'activity_description' => 'required|string|max:255',
            'carbon_amount' => 'required|numeric|min:0|max:1000',
            'activity_date' => 'required|date|before_or_equal:today',
            'metadata' => 'nullable|array',
            'metadata.transport_method' => 'required_if:activity_type,transport|string',
            'metadata.distance' => 'nullable|numeric|min:0',
            'metadata.fuel_amount' => 'nullable|numeric|min:0',
            'metadata.energy_type' => 'required_if:activity_type,energy|string',
            'metadata.amount' => 'nullable|numeric|min:0',
            'metadata.food_type' => 'required_if:activity_type,food|string',
            'metadata.weight' => 'nullable|numeric|min:0',
        ];
    }

    public function messages(): array
    {
        return [
            'activity_type.required' => 'Activity type is required',
            'activity_type.in' => 'Invalid activity type selected',
            'activity_description.required' => 'Activity description is required',
            'carbon_amount.required' => 'Carbon amount is required',
            'carbon_amount.numeric' => 'Carbon amount must be a number',
            'carbon_amount.min' => 'Carbon amount cannot be negative',
            'carbon_amount.max' => 'Carbon amount seems unrealistic (max: 1000 kg)',
            'activity_date.required' => 'Activity date is required',
            'activity_date.before_or_equal' => 'Activity date cannot be in the future',
            'metadata.transport_method.required_if' => 'Transport method is required for
transportation activities',
            'metadata.energy_type.required_if' => 'Energy type is required for energy activities',
            'metadata.food_type.required_if' => 'Food type is required for food activities',
        ];
    }
}
`

```

4.3 API Resources

UserResource

```

`php
<?php

namespace App\Http\Resources;

```

```

use Illuminate\Http\Request;
use Illuminate\Http\Resources\Json\JsonResource;

class UserResource extends JsonResource
{
    public function toArray(Request $request): array
    {
        return [
            'id' => $this->id,
            'username' => $this->username,
            'email' => $this->when($this->isOwner($request), $this->email),
            'role' => $this->role,
            'status' => $this->status,
            'profile_image' => $this->profile_image ? asset('storage/' . $this->profile_image) : null,
            'bio' => $this->bio,
            'eco_score' => $this->eco_score,
            'language' => $this->language,
            'followers_count' => $this->when($this->relationLoaded('followers'), $this->followers()-
>count()),
            'following_count' => $this->when($this->relationLoaded('following'), $this->following()-
>count()),
            'posts_count' => $this->when($this->relationLoaded('posts'), $this->posts()->count()),
            'total_emissions' => $this->when($this->relationLoaded('emissions'), $this->emissions()-
>sum('carbon_amount')),
            'is_following' => $this->when(
                $request->user() && $request->user()->id !== $this->id,
                fn() => $request->user()->isFollowing($this->resource)
            ),
            'created_at' => $this->created_at,
            'updated_at' => $this->updated_at,
        ];
    }

    private function isOwner(Request $request): bool
    {
        return $request->user() && $request->user()->id === $this->id;
    }
}

```

PostResource

```

`php
<?php

```

```

namespace App\Http\Resources;

```

```

use Illuminate\Http\Request;
use Illuminate\Http\Resources\Json\JsonResource;

```

```

class PostResource extends JsonResource
{
    public function toArray(Request $request): array
    {
        return [
            'id' => $this->id,
            'title' => $this->title,
            'content' => $this->content,
            'image_path' => $this->image_path ? asset('storage/' . $this->image_path) : null,
            'tags' => $this->tags ?? [],
        ];
    }
}

```

```

        'status' => $this->status,
        'views_count' => $this->views_count,
        'user' => new UserResource($this->whenLoaded('user')),
        'reaction_counts' => $this->getReactionCounts(),
        'user_reaction' => $this->getUserReaction($request),
        'can_edit' => $this->canUserEdit($request),
        'can_delete' => $this->canUserDelete($request),
        'created_at' => $this->created_at,
        'updated_at' => $this->updated_at,
    ];
}

private function getReactionCounts(): array
{
    if (!$this->relationLoaded('reactions')) {
        return [];
    }

    return $this->reactions
        ->groupBy('reaction_type')
        ->map(fn($group) => $group->count())
        ->toArray();
}

private function getUserReaction(Request $request): ?string
{
    if (!$request->user() || !$this->relationLoaded('reactions')) {
        return null;
    }

    $userReaction = $this->reactions
        ->where('user_id', $request->user()->id)
        ->first();

    return $userReaction?->reaction_type;
}

private function canUserEdit(Request $request): bool
{
    if (!$request->user()) {
        return false;
    }

    return $request->user()->id === $this->user_id ||
        $request->user()->canModerate();
}

private function canUserDelete(Request $request): bool
{
    if (!$request->user()) {
        return false;
    }

    return $request->user()->id === $this->user_id ||
        $request->user()->canModerate();
}
}

```

5. Background Jobs & Queue System

5.1 Job Classes

CalculateEcoScore Job

```
```php
<?php
```

```
namespace App\Jobs;
```

```
use App\Models\User;
use App\Services\EmissionCalculatorService;
use Carbon\Carbon;
use Illuminate\Bus\Queueable;
use Illuminate\Contracts\Queue\ShouldQueue;
use Illuminate\Foundation\Bus\Dispatchable;
use Illuminate\Queue\InteractsWithQueue;
use Illuminate\Queue\SerializesModels;
```

```
class CalculateEcoScore implements ShouldQueue
```

```
{
 use Dispatchable, InteractsWithQueue, Queueable, SerializesModels;
```

```
 public $timeout = 300; // 5 minutes
 public $tries = 3;
```

```
 public function __construct(
 private User $user,
 private ?Carbon $fromDate = null,
 private ?Carbon $toDate = null
) {
 $this->fromDate = $fromDate ?? now()->subDays(30);
 $this->toDate = $toDate ?? now();
 }
```

```
 public function handle(EmissionCalculatorService $calculator): void
 {
```

```
 try {
 // Calculate user's total emissions for the period
 $userEmissions = $this->user->emissions()
 ->whereBetween('activity_date', [$this->fromDate, $this->toDate])
 ->sum('carbon_amount');

 // Calculate average emissions for all users in the same period
 $averageEmissions = User::whereHas('emissions', function ($query) {
 $query->whereBetween('activity_date', [$this->fromDate, $this->toDate]);
 })->withSum(['emissions' => function ($query) {
 $query->whereBetween('activity_date', [$this->fromDate, $this->toDate]);
 }], 'carbon_amount')
 ->avg('emissions_sum_carbon_amount') ?? 0;
```

```
 // Calculate eco score
 $ecoScore = $calculator->getEcoScoreFromEmissions($userEmissions,
 $averageEmissions);
```

```
 // Update user's eco score
 $this->user->update(['eco_score' => $ecoScore]);
```

```
 // Check for achievements
 CheckAchievements::dispatch($this->user, $ecoScore);
 }
```

```

 // Log the calculation
 \Log::info("Eco score calculated for user {$this->user->id}: {$ecoScore}");

 } catch (\Exception $e) {
 \Log::error("Failed to calculate eco score for user {$this->user->id}: " . $e->getMessage());
 throw $e;
 }
}

public function failed(\Throwable $exception): void
{
 \Log::error("Eco score calculation failed for user {$this->user->id}: " . $exception-
>getMessage());
}
}
}

```

#### ProcessImageUpload Job

```

```php
<?php

```

```

namespace App\Jobs;

```

```

use Illuminate\Bus\Queueable;
use Illuminate\Contracts\Queue\ShouldQueue;
use Illuminate\Foundation\Bus\Dispatchable;
use Illuminate\Queue\InteractsWithQueue;
use Illuminate\Queue\SerializesModels;
use Illuminate\Support\Facades\Storage;
use Intervention\Image\Laravel\Facades\Image;

```

```

class ProcessImageUpload implements ShouldQueue

```

```

{
    use Dispatchable, InteractsWithQueue, Queueable, SerializesModels;

```

```

    public $timeout = 120; // 2 minutes
    public $tries = 3;

```

```

    public function __construct(
        private string $tempPath,
        private string $finalPath,
        private string $type = 'post', // 'post', 'avatar'
        private array $sizes = []
    ) {
        $this->sizes = $sizes ?: $this->getDefaultSizes($type);
    }

```

```

    public function handle(): void
    {
        try {
            if (!Storage::exists($this->tempPath)) {
                throw new \Exception("Temporary file not found: {$this->tempPath}");
            }

```

```

            $image = Image::read(Storage::path($this->tempPath));

```

```

            // Process different sizes
            foreach ($this->sizes as $sizeName => $dimensions) {
                $processedImage = clone $image;

```



```

        if ($sizeName === 'original') {
            // For original, just optimize without resizing
            $processedImage->optimize();
        } else {
            // Resize and optimize
            $processedImage->resize(
                $dimensions['width'],
                $dimensions['height'],
                function ($constraint) {
                    $constraint->aspectRatio();
                    $constraint->upscale();
                }
            );
        }

        // Generate file path
        $fileName = $this->generateFileName($sizeName);
        $savePath = $this->getFinalPath($fileName);

        // Save the processed image
        $processedImage->save(Storage::path($savePath), 85); // 85% quality

        \Log::info("Processed image saved: {$savePath}");
    }

    // Clean up temporary file
    Storage::delete($this->tempPath);

} catch (\Exception $e) {
    \Log::error("Image processing failed: " . $e->getMessage());
    Storage::delete($this->tempPath); // Clean up even on failure
    throw $e;
}

}

private function getDefaultSizes(string $type): array
{
    return match($type) {
        'avatar' => [
            'thumbnail' => ['width' => 150, 'height' => 150],
            'medium' => ['width' => 300, 'height' => 300],
            'original' => []
        ],
        'post' => [
            'thumbnail' => ['width' => 300, 'height' => 200],
            'medium' => ['width' => 800, 'height' => 600],
            'original' => []
        ],
        default => [
            'medium' => ['width' => 800, 'height' => 600]
        ]
    };
}

private function generateFileName(string $size): string
{
    $pathInfo = pathinfo($this->finalPath);
    $name = $pathInfo['filename'];
    $extension = $pathInfo['extension'];

```

```

        if ($size === 'original') {
            return "{$name}.{$extension}";
        }

        return "{$name}_{$size}.{$extension}";
    }

    private function getFinalPath(string $fileName): string
    {
        $directory = dirname($this->finalPath);
        return "{$directory}/{$fileName}";
    }

    public function failed(\Throwable $exception): void
    {
        \Log::error("Image processing job failed: " . $exception->getMessage());
        Storage::delete($this->tempPath);
    }
}

```

5.2 Event-Listener System

UserFollowed Event

```

```php
<?php

```

```

namespace App\Events;

```

```

use App\Models\User;
use Illuminate\Broadcasting\InteractsWithSockets;
use Illuminate\Broadcasting\PrivateChannel;
use Illuminate\Contracts\Broadcasting\ShouldBroadcast;
use Illuminate\Foundation\Events\Dispatchable;
use Illuminate\Queue\SerializesModels;

```

```

class UserFollowed implements ShouldBroadcast

```

```

{
 use Dispatchable, InteractsWithSockets, SerializesModels;

```

```

 public function __construct(
 public User $follower,
 public User $following
) {}

```

```

 public function broadcastOn(): array
 {
 return [
 new PrivateChannel("user.{$this->following->id}")
];
 }

```

```

 public function broadcastWith(): array
 {
 return [
 'follower' => [
 'id' => $this->follower->id,
 'username' => $this->follower->username,
 'profile_image' => $this->follower->profile_image
],

```

```

 'message' => "{$this->follower->username} started following you"
];
}

public function broadcastAs(): string
{
 return 'user.followed';
}
}

```

#### SendFollowNotification Listener

```

```php
<?php

```

```

namespace App\Listeners;

```

```

use App\Events\UserFollowed;
use App\Services\NotificationService;

```

```

class SendFollowNotification
{
    public function __construct(
        private NotificationService $notificationService
    ) {}

    public function handle(UserFollowed $event): void
    {
        $this->notificationService->sendFollowNotification(
            $event->follower,
            $event->following
        );
    }
}

```

6. WebSocket Integration (Laravel Reverb)

6.1 Broadcasting Configuration

```

```php
// config/broadcasting.php
'reverb' => [
 'driver' => 'reverb',
 'key' => env('REVERB_APP_KEY'),
 'secret' => env('REVERB_APP_SECRET'),
 'app_id' => env('REVERB_APP_ID'),
 'options' => [
 'host' => env('REVERB_HOST', '0.0.0.0'),
 'port' => env('REVERB_PORT', 8080),
 'scheme' => env('REVERB_SCHEME', 'http'),
 'useTLS' => env('REVERB_SCHEME', 'http') === 'https',
],
],

```

### 6.2 Frontend WebSocket Integration

#### useWebSocket Composable

```

```javascript

```

```

// composables/useWebSocket.js
import { ref, onMounted, onUnmounted } from 'vue'
import Echo from 'laravel-echo'
import Pusher from 'pusher-js'

export function useWebSocket() {
  const isConnected = ref(false)
  const echo = ref(null)
  const channels = ref(new Map())

  const connect = (token) => {
    if (echo.value) {
      disconnect()
    }

    window.Pusher = Pusher

    echo.value = new Echo({
      broadcaster: 'reverb',
      key: import.meta.env.VITE_REVERB_APP_KEY,
      wsHost: import.meta.env.VITE_REVERB_HOST,
      wsPort: import.meta.env.VITE_REVERB_PORT,
      wssPort: import.meta.env.VITE_REVERB_PORT,
      forceTLS: import.meta.env.VITE_REVERB_SCHEME === 'https',
      enabledTransports: ['ws', 'wss'],
      auth: {
        headers: {
          Authorization: `Bearer ${token}`
        }
      }
    })

    echo.value.connector.pusher.connection.bind('connected', () => {
      isConnected.value = true
      console.log('WebSocket connected')
    })

    echo.value.connector.pusher.connection.bind('disconnected', () => {
      isConnected.value = false
      console.log('WebSocket disconnected')
    })

    echo.value.connector.pusher.connection.bind('error', (error) => {
      console.error('WebSocket error:', error)
    })
  }

  const disconnect = () => {
    if (echo.value) {
      channels.value.forEach((channel, channelName) => {
        echo.value.leave(channelName)
      })
      channels.value.clear()
      echo.value.disconnect()
      echo.value = null
      isConnected.value = false
    }
  }

  const joinChannel = (channelName, callback) => {

```

```

if (!echo.value) {
  console.warn('Echo not initialized')
  return
}

const channel = echo.value.private(channelName)
channels.value.set(channelName, channel)

// Listen for all events on this channel
channel.listen('.notification.sent', callback)
channel.listen('.user.followed', callback)
channel.listen('.post.created', callback)
channel.listen('.post.reacted', callback)

return channel
}

const leaveChannel = (channelName) => {
  if (channels.value.has(channelName)) {
    echo.value.leave(channelName)
    channels.value.delete(channelName)
  }
}

const joinNotificationChannel = (userId, callback) => {
  return joinChannel(`notifications.${userId}`, (event) => {
    callback(event)
  })
}

const joinPostFeedChannel = (callback) => {
  return joinChannel('posts.feed', (event) => {
    callback(event)
  })
}

onUnmounted(() => {
  disconnect()
})

return {

  public function scopeActive($query)
  {
    return $query->where('status', UserStatus::ACTIVE);
  }

  public function scopeByRole($query, UserRole $role)
  {
    return $query->where('role', $role);
  }

  public function scopeWithEcoScore($query, $minScore = 0)
  {
    return $query->where('eco_score', '>=', $minScore);
  }

  // Methods
  public function isFollowing(User $user): bool
  {

```

```

        return $this->following()->where('following_id', $user->id)->exists();
    }

    public function hasRole(UserRole $role): bool
    {
        return $this->role === $role;
    }

    public function canModerate(): bool
    {
        return in_array($this->role, [UserRole::ADMIN, UserRole::MODERATOR]);
    }

    public function isBanned(): bool
    {
        return $this->status === UserStatus::BANNED;
    }

    public function getFollowersCount(): int
    {
        return $this->followers()->count();
    }

    public function getFollowingCount(): int
    {
        return $this->following()->count();
    }

    public function getTotalEmissions(?Carbon $from = null, ?Carbon $to = null): float
    {
        $query = $this->emissions();

        if ($from) {
            $query->where('activity_date', '>=', $from);
        }

        if ($to) {
            $query->where('activity_date', '<=', $to);
        }

        return $query->sum('carbon_amount');
    }
}

```

Post Model

```

```php
<?php

```

```

namespace App\Models;

```

```

use Illuminate\Database\Eloquent\Model;
use App\Enums\PostStatus;

```

```

class Post extends Model
{
 protected $fillable = [
 'user_id',
 'title',
 'content',
];
}

```

```

 'image_path',
 'tags',
 'status'
];

 protected $casts = [
 'tags' => 'array',
 'status' => PostStatus::class,
 'views_count' => 'integer',
];

 // Relationships
 public function user()
 {
 return $this->belongsTo(User::class);
 }

 public function reactions()
 {
 return $this->morphMany(Reaction::class, 'reactable');
 }

 public function reports()
 {
 return $this->hasMany(Report::class, 'reported_post_id');
 }

 // Scopes
 public function scopePublished($query)
 {
 return $query->where('status', PostStatus::PUBLISHED);
 }

 public function scopeByTags($query, array $tags)
 {
 foreach ($tags as $tag) {
 $query->whereJsonContains('tags', $tag);
 }
 return $query;
 }

 public function scopeFollowingFeed($query, User $user)
 {
 $followingIds = $user->following()->pluck('following_id');
 return $query->whereIn('user_id', $followingIds);
 }

 public function scopePopular($query, $days = 7)
 {
 return $query->withCount('reactions')
 ->where('created_at', '>=', now()->subDays($days))
 ->orderBy('reactions_count', 'desc');
 }

 // Methods
 public function getReactionCount($type = null): int
 {
 $query = $this->reactions();

 if ($type) {

```

```

 $query->where('reaction_type', $type);
 }

 return $query->count();
}

public function hasUserReacted(User $user, $type = null): bool
{
 $query = $this->reactions()->where('user_id', $user->id);

 if ($type) {
 $query->where('reaction_type', $type);
 }

 return $query->exists();
}

public function incrementViews(): void
{
 $this->increment('views_count');
}
}

```

#### Emission Model

```

```php
<?php

```

```

namespace App\Models;

```

```

use Illuminate\Database\Eloquent\Model;
use Carbon\Carbon;

```

```

class Emission extends Model
{
    protected $fillable = [
        'user_id',
        'activity_type',
        'activity_description',
        'carbon_amount',
        'activity_date',
        'metadata'
    ];

    protected $casts = [
        'carbon_amount' => 'decimal:2',
        'activity_date' => 'date',
        'metadata' => 'array',
    ];

    // Relationships
    public function user()
    {
        return $this->belongsTo(User::class);
    }

    // Scopes

```