# FIBONACCO ADVERTISING, EMAIL & EMERGENCY SYSTEMS

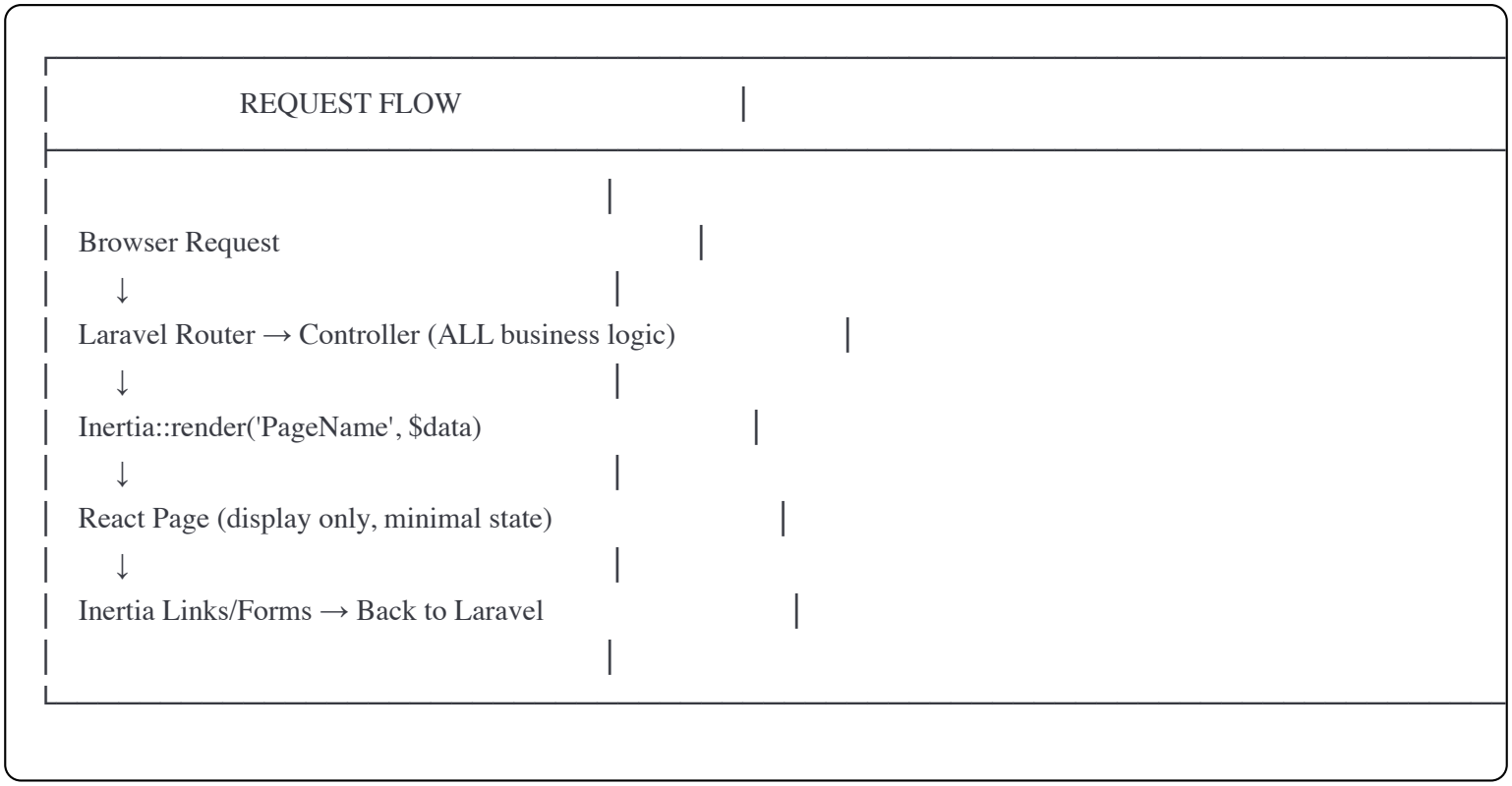
### Cursor Instructions - Laravel + Inertia.js (React Navigation)

**Stack:** Laravel 11 + Inertia.js + React (navigation only) + TypeScript + Tailwind CSS
**Database:** PostgreSQL + Valkey (Redis)
**Hosting:** AWS (ECS Fargate, RDS, ElastiCache)
**Date:** December 23, 2025

---

## ARCHITECTURE OVERVIEW

This follows your existing pattern: Laravel handles ALL business logic, Inertia passes data to React pages for display/navigation. No heavy React components or state management.




---

## PHASE 1: DATABASE MIGRATIONS

All migrations go in `database/migrations/`. Run with `php artisan migrate`.

### 1.1 Advertising Tables

php

```php
<?php
// database/migrations/2024_12_23_000001_create_ad_campaigns_table.php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    public function up(): void
    {
        Schema::create('ad_campaigns', function (Blueprint $table) {
            $table->id();
            $table->uuid('uuid')->unique();
            $table->foreignId('advertiser_id')->constrained('businesses')->cascadeOnDelete();
            $table->string('name');
            $table->text('description')->nullable();
            $table->enum('status', ['draft', 'pending', 'active', 'paused', 'completed', 'cancelled'])->default('draft');
            $table->enum('type', ['cpm', 'cpc', 'flat_rate', 'sponsored']);
            $table->decimal('budget', 12, 2);
            $table->decimal('spent', 12, 2)->default(0);
            $table->decimal('daily_budget', 10, 2)->nullable();
            $table->date('start_date');
            $table->date('end_date');
            $table->json('targeting')->nullable(); // communities, demographics, etc.
            $table->json('platforms')->nullable(); // day_news, goeventcity, etc.
            $table->timestamps();
            $table->softDeletes();

            $table->index(['status', 'start_date', 'end_date']);
            $table->index('advertiser_id');
        });
    }

    public function down(): void
    {
        Schema::dropIfExists('ad_campaigns');
    }
};
```

php

```php
<?php
// database/migrations/2024_12_23_000002_create_ad_creatives_table.php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    public function up(): void
    {
        Schema::create('ad_creatives', function (Blueprint $table) {
            $table->id();
            $table->uuid('uuid')->unique();
            $table->foreignId('campaign_id')->constrained('ad_campaigns')->cascadeOnDelete();
            $table->string('name');
            $table->enum('format', ['leaderboard', 'medium_rectangle', 'sidebar', 'native', 'sponsored_article', 'audio', 'video']);
            $table->string('headline')->nullable();
            $table->text('body')->nullable();
            $table->string('image_url')->nullable();
            $table->string('video_url')->nullable();
            $table->string('audio_url')->nullable();
            $table->string('click_url');
            $table->string('cta_text')->default('Learn More');
            $table->enum('status', ['draft', 'pending_review', 'approved', 'rejected', 'active', 'paused'])->default('draft');
            $table->integer('width')->nullable();
            $table->integer('height')->nullable();
            $table->timestamps();
            $table->softDeletes();

            $table->index(['campaign_id', 'status']);
            $table->index('format');
        });
    }

    public function down(): void
    {
        Schema::dropIfExists('ad_creatives');
    }
};
```

```
php
```

```php
<?php
// database/migrations/2024_12_23_000003_create_ad_placements_table.php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    public function up(): void
    {
        Schema::create('ad_placements', function (Blueprint $table) {
            $table->id();
            $table->string('platform'); // day_news, goeventcity, downtown_guide, alphasite_community, golocalvoices
            $table->string('slot'); // header_leaderboard, sidebar_top, in_article, footer, etc.
            $table->string('name');
            $table->text('description')->nullable();
            $table->string('format'); // leaderboard, medium_rectangle, etc.
            $table->integer('width');
            $table->integer('height');
            $table->decimal('base_cpm', 8, 2);
            $table->decimal('base_cpc', 8, 2)->nullable();
            $table->boolean('is_active')->default(true);
            $table->integer('priority')->default(0);
            $table->timestamps();

            $table->unique(['platform', 'slot']);
            $table->index('platform');
            $table->index('is_active');
        });
    }

    public function down(): void
    {
        Schema::dropIfExists('ad_placements');
    }
};
```

```php
php
```

php

```php
<?php
// database/migrations/2024_12_23_000004_create_ad_inventory_table.php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    public function up(): void
    {
        Schema::create('ad_inventory', function (Blueprint $table) {
            $table->id();
            $table->foreignId('placement_id')->constrained('ad_placements')->cascadeOnDelete();
            $table->foreignId('community_id')->constrained('communities')->cascadeOnDelete();
            $table->date('date');
            $table->integer('total_impressions')->default(0);
            $table->integer('sold_impressions')->default(0);
            $table->integer('delivered_impressions')->default(0);
            $table->decimal('revenue', 10, 2)->default(0);
            $table->timestamps();

            $table->unique(['placement_id', 'community_id', 'date']);
            $table->index(['date', 'placement_id']);
        });
    }

    public function down(): void
    {
        Schema::dropIfExists('ad_inventory');
    }
};
```

php

```php
<?php
// database/migrations/2024_12_23_000005_create_ad_impressions_table.php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    public function up(): void
    {
        Schema::create('ad_impressions', function (Blueprint $table) {
            $table->id();
            $table->foreignId('creative_id')->constrained('ad_creatives')->cascadeOnDelete();
            $table->foreignId('placement_id')->constrained('ad_placements')->cascadeOnDelete();
            $table->foreignId('community_id')->constrained('communities')->cascadeOnDelete();
            $table->string('session_id', 64)->nullable();
            $table->string('ip_hash', 64)->nullable();
            $table->string('user_agent')->nullable();
            $table->string('referrer')->nullable();
            $table->decimal('cost', 8, 4)->default(0);
            $table->timestamp('impressed_at');

            $table->index(['creative_id', 'impressed_at']);
            $table->index(['placement_id', 'impressed_at']);
```

```
            $table->index(['community_id', 'impressed_at']);
            $table->index('impressed_at');
        });
    }

    public function down(): void
    {
        Schema::dropIfExists('ad_impressions');
    }
};
```

php

```
<?php
// database/migrations/2024_12_23_000006_create_ad_clicks_table.php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;
```

```php
return new class extends Migration
{
    public function up(): void
    {
        Schema::create('ad_clicks', function (Blueprint $table) {
            $table->id();
            $table->foreignId('impression_id')->constrained('ad_impressions')->cascadeOnDelete();
            $table->foreignId('creative_id')->constrained('ad_creatives')->cascadeOnDelete();
            $table->string('ip_hash', 64)->nullable();
            $table->decimal('cost', 8, 4)->default(0);
            $table->timestamp('clicked_at');

            $table->index(['creative_id', 'clicked_at']);
            $table->index('clicked_at');
        });
    }

    public function down(): void
    {
        Schema::dropIfExists('ad_clicks');
    }
};
```

## 1.2 Email Tables

```php
```

```php
<?php
// database/migrations/2024_12_23_000010_create_email_subscribers_table.php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    public function up(): void
    {
        Schema::create('email_subscribers', function (Blueprint $table) {
            $table->id();
            $table->uuid('uuid')->unique();
            $table->string('email')->index();
            $table->string('first_name')->nullable();
            $table->string('last_name')->nullable();
            $table->foreignId('community_id')->constrained('communities')->cascadeOnDelete();
            $table->foreignId('business_id')->nullable()->constrained('businesses')->nullOnDelete();
            $table->enum('type', ['reader', 'smb'])->default('reader');
            $table->enum('status', ['pending', 'active', 'unsubscribed', 'bounced', 'complained'])->default('pending');
            $table->timestamp('confirmed_at')->nullable();
            $table->timestamp('unsubscribed_at')->nullable();
            $table->string('unsubscribe_reason')->nullable();
            $table->json('preferences')->nullable(); // daily_digest, breaking_news, weekly_newsletter
            $table->string('source')->nullable(); // signup_form, import, api_claim
```

```
            $table->string('source')->nullable(); // signup_form, import, api, claim
            $table->timestamps();
            $table->softDeletes();

            $table->unique(['email', 'community_id']);
            $table->index(['community_id', 'status', 'type']);
        });
    }

    public function down(): void
    {
        Schema::dropIfExists('email_subscribers');
    }
};
```

php

```
<?php
// database/migrations/2024_12_23_000011_create_email_templates_table.php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    public function up(): void
    {
        Schema::create('email_templates', function (Blueprint $table) {
            $table->id();
            $table->uuid('uuid')->unique();
            $table->string('name');
            $table->string('slug')->unique();
            $table->enum('type', ['daily_digest', 'breaking_news', 'weekly_newsletter', 'smb_report', 'emergency', 'transactional']);
            $table->string('subject_template');
            $table->string('preview_text')->nullable();
            $table->longText('html_template');
            $table->longText('text_template')->nullable();
            $table->json('variables')->nullable(); // List of available merge variables
            $table->boolean('is_active')->default(true);
            $table->integer('version')->default(1);
            $table->timestamps();

            $table->index(['type', 'is_active']);
        });
```

```
    }

    public function down(): void
    {
        Schema::dropIfExists('email_templates');
    }
};
```

php

```php
<?php
// database/migrations/2024_12_23_000012_create_email_campaigns_table.php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    public function up(): void
    {
        Schema::create('email_campaigns', function (Blueprint $table) {
            $table->id();
            $table->uuid('uuid')->unique();
            $table->foreignId('community_id')->constrained('communities')->cascadeOnDelete();
            $table->foreignId('template_id')->nullable()->constrained('email_templates')->nullOnDelete();
            $table->string('name');
            $table->enum('type', ['daily_digest', 'breaking_news', 'weekly_newsletter', 'smb_report', 'emergency', 'custom']);
            $table->enum('status', ['draft', 'scheduled', 'sending', 'sent', 'cancelled'])->default('draft');
            $table->string('subject');
            $table->string('preview_text')->nullable();
            $table->longText('html_content')->nullable();
            $table->longText('text_content')->nullable();
            $table->json('segment')->nullable(); // targeting criteria
            $table->timestamp('scheduled_at')->nullable();
            $table->timestamp('started_at')->nullable();
            $table->timestamp('completed_at')->nullable();
            $table->integer('total_recipients')->default(0);
            $table->integer('sent_count')->default(0);
            $table->integer('delivered_count')->default(0);
            $table->integer('opened_count')->default(0);
            $table->integer('clicked_count')->default(0);
            $table->integer('bounced_count')->default(0);
            $table->integer('complained_count')->default(0);
```

```
            $table->integer('unsubscribed_count')->default(0);
            $table->timestamps();

            $table->index(['community_id', 'status']);
            $table->index(['type', 'status']);
            $table->index('scheduled_at');
        });
    }

    public function down(): void
    {
        Schema::dropIfExists('email_campaigns');
    }
};
```

```
php
```

```php
<?php
// database/migrations/2024_12_23_000013_create_email_sends_table.php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    public function up(): void
    {
        Schema::create('email_sends', function (Blueprint $table) {
            $table->id();
            $table->foreignId('campaign_id')->constrained('email_campaigns')->cascadeOnDelete();
            $table->foreignId('subscriber_id')->constrained('email_subscribers')->cascadeOnDelete();
            $table->string('message_id')->nullable()->index();
            $table->enum('status', ['queued', 'sent', 'delivered', 'bounced', 'complained', 'failed'])->default('queued');
            $table->timestamp('sent_at')->nullable();
            $table->timestamp('delivered_at')->nullable();
            $table->timestamp('opened_at')->nullable();
            $table->integer('open_count')->default(0);
            $table->timestamp('clicked_at')->nullable();
            $table->integer('click_count')->default(0);
            $table->string('bounce_type')->nullable();
            $table->text('error_message')->nullable();
            $table->timestamps();

            $table->unique(['campaign_id', 'subscriber_id']);
            $table->index(['campaign_id', 'status']);
            $table->index('sent_at');
        });
    }

    public function down(): void
```

```
    {
        Schema::dropIfExists('email_sends');
    }
};
```

php

```php
<?php
// database/migrations/2024_12_23_000014_create_newsletter_subscriptions_table.php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    public function up(): void
    {
        Schema::create('newsletter_subscriptions', function (Blueprint $table) {
            $table->id();
            $table->foreignId('subscriber_id')->constrained('email_subscribers')->cascadeOnDelete();
            $table->enum('tier', ['free', 'paid'])->default('free');
            $table->decimal('price', 6, 2)->default(1.00);
            $table->string('stripe_subscription_id')->nullable();
            $table->enum('status', ['active', 'cancelled', 'past_due', 'paused'])->default('active');
            $table->timestamp('started_at');
            $table->timestamp('cancelled_at')->nullable();
            $table->timestamp('current_period_end')->nullable();
            $table->timestamps();

            $table->index(['subscriber_id', 'status']);
            $table->index('stripe_subscription_id');
        });
    }

    public function down(): void
    {
        Schema::dropIfExists('newsletter_subscriptions');
    }
};
```

## 1.3 Emergency Tables

php

```php
<?php
// database/migrations/2024_12_23_000020_create_emergency_alerts_table.php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    public function up(): void
    {
        Schema::create('emergency_alerts', function (Blueprint $table) {
            $table->id();
            $table->uuid('uuid')->unique();
            $table->foreignId('community_id')->constrained('communities')->cascadeOnDelete();
            $table->foreignId('created_by')->nullable()->constrained('users')->nullOnDelete();
            $table->foreignId('municipal_partner_id')->nullable();
            $table->enum('priority', ['critical', 'urgent', 'advisory', 'info'])->default('advisory');
            $table->string('category'); // weather, crime, health, utility, traffic, government, school, amber
            $table->string('title');
            $table->text('message');
            $table->text('instructions')->nullable();
            $table->string('source')->nullable();
            $table->string('source_url')->nullable();
            $table->enum('status', ['draft', 'active', 'expired', 'cancelled'])->default('draft');
            $table->timestamp('published_at')->nullable();
            $table->timestamp('expires_at')->nullable();
            $table->json('delivery_channels')->nullable(); // email, sms, push
            $table->integer('email_sent')->default(0);
            $table->integer('sms_sent')->default(0);
            $table->timestamps();
            $table->softDeletes();

            $table->index(['community_id', 'status', 'priority']);
            $table->index(['status', 'published_at']);
            $table->index('expires_at');
        });
    }

    public function down(): void
    {
        Schema::dropIfExists('emergency_alerts');
    }
};
```

```
};
```

php

```php
<?php
// database/migrations/2024_12_23_000021_create_emergency_subscriptions_table.php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    public function up(): void
    {
        Schema::create('emergency_subscriptions', function (Blueprint $table) {
            $table->id();
            $table->foreignId('subscriber_id')->constrained('email_subscribers')->cascadeOnDelete();
            $table->boolean('email_enabled')->default(true);
            $table->boolean('sms_enabled')->default(false);
            $table->string('phone_number')->nullable();
            $table->boolean('phone_verified')->default(false);
            $table->string('phone_verification_code')->nullable();
            $table->timestamp('phone_verified_at')->nullable();
            $table->json('priority_levels')->nullable(); // which priorities to receive
            $table->json('categories')->nullable(); // which categories to receive
            $table->string('stripe_subscription_id')->nullable(); // for SMS tier
            $table->enum('sms_tier', ['none', 'basic'])->default('none');
            $table->timestamps();

            $table->unique('subscriber_id');
            $table->index(['sms_enabled', 'phone_verified']);
        });
    }

    public function down(): void
    {
        Schema::dropIfExists('emergency_subscriptions');
    }
};
```

php

```php
<?php
// database/migrations/2024_12_23_000022_create_emergency_deliveries_table.php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    public function up(): void
    {
        Schema::create('emergency_deliveries', function (Blueprint $table) {
            $table->id();
            $table->foreignId('alert_id')->constrained('emergency_alerts')->cascadeOnDelete();
            $table->foreignId('subscription_id')->constrained('emergency_subscriptions')->cascadeOnDelete();
            $table->enum('channel', ['email', 'sms']);
            $table->enum('status', ['queued', 'sent', 'delivered', 'failed'])->default('queued');
            $table->string('external_id')->nullable(); // SES message ID or Twilio SID
            $table->timestamp('sent_at')->nullable();
            $table->timestamp('delivered_at')->nullable();
            $table->text('error_message')->nullable();
            $table->timestamps();

            $table->unique(['alert_id', 'subscription_id', 'channel']);
            $table->index(['alert_id', 'status']);
            $table->index('sent_at');
        });
    }

    public function down(): void
    {
        Schema::dropIfExists('emergency_deliveries');
    }
};
```

```php
<?php
// database/migrations/2024_12_23_000023_create_municipal_partners_table.php
```

```php
use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    public function up(): void
    {
        Schema::create('municipal_partners', function (Blueprint $table) {

            $table->id();
            $table->uuid('uuid')->unique();
            $table->string('name');
            $table->enum('type', ['municipality', 'law_enforcement', 'school_district', 'utility', 'other']);
            $table->json('community_ids'); // communities they can broadcast to
            $table->foreignId('primary_contact_id')->nullable()->constrained('users')->nullOnDelete();
            $table->string('api_key_hash')->nullable();
            $table->boolean('is_verified')->default(false);
            $table->boolean('is_active')->default(true);
            $table->json('allowed_categories')->nullable();
            $table->json('allowed_priorities')->nullable();
            $table->boolean('requires_approval')->default(true);
            $table->timestamps();
            $table->softDeletes();

            $table->index(['is_active', 'is_verified']);
        });

        // Add foreign key to emergency_alerts
        Schema::table('emergency_alerts', function (Blueprint $table) {
            $table->foreign('municipal_partner_id')
                ->references('id')
                ->on('municipal_partners')
                ->nullOnDelete();
        });
    }

    public function down(): void
    {
        Schema::table('emergency_alerts', function (Blueprint $table) {
            $table->dropForeign(['municipal_partner_id']);
        });
        Schema::dropIfExists('municipal_partners');
    }
};
```

php

```php
<?php
// database/migrations/2024_12_23_000024_create_emergency_audit_log_table.php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    public function up(): void
    {
        Schema::create('emergency_audit_log', function (Blueprint $table) {
            $table->id();
            $table->foreignId('alert_id')->nullable()->constrained('emergency_alerts')->nullOnDelete();
            $table->foreignId('user_id')->nullable()->constrained('users')->nullOnDelete();
            $table->foreignId('municipal_partner_id')->nullable()->constrained('municipal_partners')->nullOnDelete();
            $table->string('action'); // created, published, updated, cancelled, expired
            $table->json('changes')->nullable();
            $table->string('ip_address')->nullable();
            $table->string('user_agent')->nullable();
            $table->timestamps();

            $table->index(['alert_id', 'created_at']);
            $table->index(['user_id', 'created_at']);
            $table->index('created_at');
        });
    }

    public function down(): void
    {
        Schema::dropIfExists('emergency_audit_log');
    }
};
```

# PHASE 2: ELOQUENT MODELS

## 2.1 Advertising Models

php

```php
<?php
// app/Models/AdCampaign.php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\SoftDeletes;
use Illuminate\Database\Eloquent\Relations\BelongsTo;
use Illuminate\Database\Eloquent\Relations\HasMany;
use Illuminate\Support\Str;

class AdCampaign extends Model
{
    use HasFactory, SoftDeletes;

    protected $fillable = [
        'uuid',
        'advertiser_id',
        'name',
        'description',
        'status',
        'type',
        'budget',
        'spent',
        'daily_budget',
        'start_date',
        'end_date',
        'targeting',
        'platforms',
    ];

    protected $casts = [
        'budget' => 'decimal:2',
        'spent' => 'decimal:2',
```

```
        'daily_budget' => 'decimal:2',
        'start_date' => 'date',
        'end_date' => 'date',
        'targeting' => 'array',
        'platforms' => 'array',
    ];

    protected static function boot()
    {
        parent::boot();
        static::creating(function ($model) {
            $model->uuid = $model->uuid ?? Str::uuid();
        });
    }

    public function advertiser(): BelongsTo
    {
        return $this->belongsTo(Business::class, 'advertiser_id');
    }

    public function creatives(): HasMany
    {
        return $this->hasMany(AdCreative::class, 'campaign_id');
    }

    public function isActive(): bool
    {
        return $this->status === 'active'
            && $this->start_date <= now()
            && $this->end_date >= now()
            && $this->spent < $this->budget;
    }

    public function getRemainingBudgetAttribute(): float
    {
        return max(0, $this->budget - $this->spent);
    }
}
```

php

```
<?php
// app/Models/AdCreative.php

namespace App\Models;
```

```php
use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\SoftDeletes;
use Illuminate\Database\Eloquent\Relations\BelongsTo;
use Illuminate\Database\Eloquent\Relations\HasMany;
use Illuminate\Support\Str;

class AdCreative extends Model
{
    use HasFactory, SoftDeletes;

    protected $fillable = [
        'uuid',
        'campaign_id',
        'name',
        'format',
        'headline',

        'body',
        'image_url',
        'video_url',
        'audio_url',
        'click_url',
        'cta_text',
        'status',
        'width',
        'height',
    ];

    protected static function boot()
    {
        parent::boot();
        static::creating(function ($model) {
            $model->uuid = $model->uuid ?? Str::uuid();
        });
    }

    public function campaign(): BelongsTo
    {
        return $this->belongsTo(AdCampaign::class, 'campaign_id');
    }

    public function impressions(): HasMany
    {
```

```
        return $this->hasMany(AdImpression::class, 'creative_id');
    }

    public function clicks(): HasMany
    {
        return $this->hasMany(AdClick::class, 'creative_id');
    }

    public function getCtrAttribute(): float
    {
        $impressions = $this->impressions()->count();
        if ($impressions === 0) return 0;
        return round(($this->clicks()->count() / $impressions) * 100, 2);
    }
}
```

php

```
<?php
// app/Models/AdPlacement.php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\Relations\HasMany;

class AdPlacement extends Model
{
    use HasFactory;

    protected $fillable = [
        'platform',
        'slot',
        'name',
        'description',
        'format',
        'width',
        'height',
        'base_cpm',
        'base_cpc',
        'is_active',
        'priority',
```

```
    ];

    protected $casts = [
        'base_cpm' => 'decimal:2',
        'base_cpc' => 'decimal:2',
        'is_active' => 'boolean',
    ];

    public function inventory(): HasMany
    {
        return $this->hasMany(AdInventory::class, 'placement_id');
    }

    public function impressions(): HasMany
    {
        return $this->hasMany(AdImpression::class, 'placement_id');
    }
}
```

php

```php
<?php

// app/Models/AdImpression.php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\Relations\BelongsTo;
use Illuminate\Database\Eloquent\Relations\HasOne;

class AdImpression extends Model
{
    public $timestamps = false;

    protected $fillable = [
        'creative_id',
        'placement_id',
        'community_id',
        'session_id',
        'ip_hash',
        'user_agent',
        'referrer',
        'cost',
        'impressed_at',
```

```
    ];

    protected $casts = [
        'cost' => 'decimal:4',
        'impressed_at' => 'datetime',
    ];

    public function creative(): BelongsTo
    {
        return $this->belongsTo(AdCreative::class, 'creative_id');
    }

    public function placement(): BelongsTo
    {
        return $this->belongsTo(AdPlacement::class, 'placement_id');
    }

    public function community(): BelongsTo
    {
        return $this->belongsTo(Community::class);
    }

    public function click(): HasOne
    {
        return $this->hasOne(AdClick::class, 'impression_id');
    }
}
```

## 2.2 Email Models

php

```
<?php
// app/Models/EmailSubscriber.php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\SoftDeletes;
use Illuminate\Database\Eloquent\Relations\BelongsTo;
use Illuminate\Database\Eloquent\Relations\HasMany;
use Illuminate\Database\Eloquent\Relations\HasOne;
use Illuminate\Support\Str;
```

```php
class EmailSubscriber extends Model
{
    use HasFactory, SoftDeletes;

    protected $fillable = [
        'uuid',
        'email',
        'first_name',
        'last_name',
        'community_id',
        'business_id',
        'type',
        'status',
        'confirmed_at',
        'unsubscribed_at',
        'unsubscribe_reason',
        'preferences',
        'source',
    ];

    protected $casts = [
        'confirmed_at' => 'datetime',

        'unsubscribed_at' => 'datetime',
        'preferences' => 'array',
    ];

    protected static function boot()
    {
        parent::boot();
        static::creating(function ($model) {
            $model->uuid = $model->uuid ?? Str::uuid();
        });
    }

    public function community(): BelongsTo
    {
        return $this->belongsTo(Community::class);
    }

    public function business(): BelongsTo
    {
        return $this->belongsTo(Business::class);
    }
```

```
    public function sends(): HasMany
    {
        return $this->hasMany(EmailSend::class, 'subscriber_id');
    }

    public function newsletterSubscription(): HasOne
    {
        return $this->hasOne(NewsletterSubscription::class, 'subscriber_id');
    }

    public function emergencySubscription(): HasOne
    {
        return $this->hasOne(EmergencySubscription::class, 'subscriber_id');
    }

    public function getFullNameAttribute(): string
    {
        return trim("{$this->first_name} {$this->last_name}") ?: $this->email;
    }

    public function wantsDigest(): bool
    {
        return ($this->preferences['daily_digest'] ?? true) && $this->status === 'active';
    }

    public function wantsBreakingNews(): bool
    {
        return ($this->preferences['breaking_news'] ?? true) && $this->status === 'active';
    }

    public function wantsNewsletter(): bool
    {
        return ($this->preferences['weekly_newsletter'] ?? false) && $this->status === 'active';
    }
}
```

php

```
<?php
// app/Models/EmailCampaign.php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
```

```php
use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\Relations\BelongsTo;
use Illuminate\Database\Eloquent\Relations\HasMany;
use Illuminate\Support\Str;

class EmailCampaign extends Model
{
    use HasFactory;

    protected $fillable = [
        'uuid',
        'community_id',
        'template_id',
        'name',
        'type',
        'status',
        'subject',
        'preview_text',
        'html_content',
        'text_content',
        'segment',
        'scheduled_at',
        'started_at',
        'completed_at',
        'total_recipients',
        'sent_count',
        'delivered_count',
        'opened_count',
        'clicked_count',
        'bounced_count',
        'complained_count',
        'unsubscribed_count',
    ];

    protected $casts = [
        'segment' => 'array',
        'scheduled_at' => 'datetime',
        'started_at' => 'datetime',
        'completed_at' => 'datetime',
    ];

    protected static function boot()
    {
        parent::boot();
        static::creating(function ($model) {
```

```
            $model->uuid = $model->uuid ?? Str::uuid();
        });
    }

    public function community(): BelongsTo
    {
        return $this->belongsTo(Community::class);
    }

    public function template(): BelongsTo
    {
        return $this->belongsTo(EmailTemplate::class);
    }

    public function sends(): HasMany
    {
        return $this->hasMany(EmailSend::class, 'campaign_id');
    }

    public function getOpenRateAttribute(): float
    {
        if ($this->delivered_count === 0) return 0;
        return round(($this->opened_count / $this->delivered_count) * 100, 2);
    }

    public function getClickRateAttribute(): float
    {
        if ($this->opened_count === 0) return 0;
        return round(($this->clicked_count / $this->opened_count) * 100, 2);
    }
}
```

## 2.3 Emergency Models

php

```
<?php
// app/Models/EmergencyAlert.php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\SoftDeletes;
use Illuminate\Database\Eloquent\Relations\BelongsTo;
```

```php
use Illuminate\Database\Eloquent\Relations\BelongsTo;
use Illuminate\Database\Eloquent\Relations\HasMany;
use Illuminate\Support\Str;

class EmergencyAlert extends Model
{
    use HasFactory, SoftDeletes;

    protected $fillable = [
        'uuid',
        'community_id',
        'created_by',
        'municipal_partner_id',
        'priority',
        'category',
        'title',
        'message',
        'instructions',
        'source',
        'source_url',
        'status',
        'published_at',
        'expires_at',
        'delivery_channels',
        'email_sent',
        'sms_sent',
    ];

    protected $casts = [
        'published_at' => 'datetime',
        'expires_at' => 'datetime',
        'delivery_channels' => 'array',
    ];

    protected static function boot()
    {
        parent::boot();
        static::creating(function ($model) {
            $model->uuid = $model->uuid ?? Str::uuid();
        });
    }

    public function community(): BelongsTo
    {
        return $this->belongsTo(Community::class);
    }
```

```php
    }

    public function creator(): BelongsTo
    {
        return $this->belongsTo(User::class, 'created_by');
    }

    public function municipalPartner(): BelongsTo
    {
        return $this->belongsTo(MunicipalPartner::class);
    }

    public function deliveries(): HasMany
    {
        return $this->hasMany(EmergencyDelivery::class, 'alert_id');
    }

    public function auditLogs(): HasMany
    {
        return $this->hasMany(EmergencyAuditLog::class, 'alert_id');
    }

    public function isActive(): bool
    {
        return $this->status === 'active'
            && ($this->expires_at === null || $this->expires_at > now());
    }

    public function getPriorityColorAttribute(): string
    {
        return match($this->priority) {
            'critical' => 'red',
            'urgent' => 'orange',
            'advisory' => 'yellow',
            'info' => 'blue',
            default => 'gray',
        };
    }
}
```

php

```php
<?php
// app/Models/EmergencySubscription.php
```

```php
namespace App\Models;

use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\Relations\BelongsTo;
use Illuminate\Database\Eloquent\Relations\HasMany;

class EmergencySubscription extends Model
{
    protected $fillable = [
        'subscriber_id',
        'email_enabled',
        'sms_enabled',
        'phone_number',
        'phone_verified',
        'phone_verification_code',
        'phone_verified_at',
        'priority_levels',
        'categories',
        'stripe_subscription_id',
        'sms_tier',
    ];

    protected $casts = [
        'email_enabled' => 'boolean',
        'sms_enabled' => 'boolean',
        'phone_verified' => 'boolean',
        'phone_verified_at' => 'datetime',
        'priority_levels' => 'array',
        'categories' => 'array',
    ];

    public function subscriber(): BelongsTo
    {
        return $this->belongsTo(EmailSubscriber::class, 'subscriber_id');
    }

    public function deliveries(): HasMany
    {
        return $this->hasMany(EmergencyDelivery::class, 'subscription_id');
    }

    public function shouldReceiveAlert(EmergencyAlert $alert): bool
    {
        // Check priority level
        $priorities = $this->priority_levels ?? ['critical', 'urgent', 'advisory', 'info'];
```

```
        if (!in_array($alert->priority, $priorities)) {
            return false;
        }

        // Check category
        $categories = $this->categories ?? [];
        if (!empty($categories) && !in_array($alert->category, $categories)) {
            return false;
        }

        return true;
    }

    public function canReceiveSms(): bool
    {
        return $this->sms_enabled
            && $this->phone_verified
            && $this->sms_tier !== 'none'
            && !empty($this->phone_number);
    }
}
```

---

# PHASE 3: LARAVEL SERVICES (Business Logic)

All business logic lives in Services. Controllers call services, services do the work.

## 3.1 Ad Server Service

```
php

<?php
// app/Services/AdServerService.php

namespace App\Services;

use App\Models\AdCampaign;
use App\Models\AdCreative;
use App\Models\AdPlacement;
use App\Models\AdImpression;
use App\Models\AdClick;
use App\Models\AdInventory;
use App\Models\Community;
```

```php
use Illuminate\Support\Facades\Cache;
use Illuminate\Support\Facades\DB;

class AdServerService
{
    /**
     * Get an ad to display for a given placement and community
     */
    public function getAd(string $platform, string $slot, int $communityId, ?string $sessionId = null): ?array
    {
        $placement = $this->getPlacement($platform, $slot);
        if (!$placement || !$placement->is_active) {
            return null;
        }

        // Get eligible campaigns targeting this community/platform
        $creative = $this->selectCreative($placement, $communityId, $sessionId);
        if (!$creative) {
            return null;
        }

        // Record impression
        $impression = $this->recordImpression($creative, $placement, $communityId, $sessionId);


        return [
            'creative_id' => $creative->uuid,
            'impression_id' => $impression->id,
            'format' => $creative->format,
            'headline' => $creative->headline,
            'body' => $creative->body,
            'image_url' => $creative->image_url,
            'click_url' => route('ads.click', ['impression' => $impression->id]),
            'cta_text' => $creative->cta_text,
            'width' => $placement->width,
            'height' => $placement->height,
        ];
    }

    /**
     * Select best creative based on targeting, budget, and rotation
     */
    protected function selectCreative(AdPlacement $placement, int $communityId, ?string $sessionId): ?AdCreative
    {
        $cacheKey = "ad_eligible_{$placement->id}_{$communityId}";
```

```php
$eligibleCreatives = Cache::remember($cacheKey, 60, function () use ($placement, $communityId) {
    return AdCreative::query()
        ->where('status', 'active')
        ->where('format', $placement->format)
        ->whereHas('campaign', function ($query) use ($communityId, $placement) {
            $query->where('status', 'active')
                ->where('start_date', '<=', now())
                ->where('end_date', '>=', now())
                ->whereRaw('spent < budget')
                ->where(function ($q) use ($communityId) {
                    $q->whereNull('targeting')
                      ->orWhereJsonContains('targeting->communities', $communityId);
                })
                ->where(function ($q) use ($placement) {
                    $q->whereNull('platforms')
                      ->orWhereJsonContains('platforms', $placement->platform);
                });
        })
        ->with('campaign')
        ->get();
});

if ($eligibleCreatives->isEmpty()) {
    return null;
}

// Apply frequency capping if session provided
if ($sessionId) {
    $recentImpressions = AdImpression::where('session_id', $sessionId)
        ->where('impressed_at', '>', now()->subHour())
        ->pluck('creative_id')
        ->toArray();

    $eligibleCreatives = $eligibleCreatives->filter(function ($creative) use ($recentImpressions) {
        // Allow max 3 impressions per hour per session
        return collect($recentImpressions)
            ->filter(fn($id) => $id === $creative->id)
            ->count() < 3;
    });
}

if ($eligibleCreatives->isEmpty()) {
    return null;
}
```

```php
    // Weighted random selection based on remaining budget
    $totalWeight = $eligibleCreatives->sum(fn($c) => $c->campaign->remaining_budget);
    $random = mt_rand(0, (int)($totalWeight * 100)) / 100;

    $cumulative = 0;
    foreach ($eligibleCreatives as $creative) {
        $cumulative += $creative->campaign->remaining_budget;
        if ($random <= $cumulative) {
            return $creative;
        }
    }

    return $eligibleCreatives->first();
}

/**
 * Record an impression
 */
protected function recordImpression(
    AdCreative $creative,
    AdPlacement $placement,
    int $communityId,
    ?string $sessionId
): AdImpression {
    // Calculate cost based on campaign type
    $cost = 0;
    if ($creative->campaign->type === 'cpm') {
        $cost = $placement->base_cpm / 1000;
    }

    $impression = AdImpression::create([
        'creative_id' => $creative->id,
        'placement_id' => $placement->id,
        'community_id' => $communityId,
        'session_id' => $sessionId,
        'ip_hash' => request()->ip() ? hash('sha256', request()->ip()) : null,
        'user_agent' => request()->userAgent(),
        'referrer' => request()->header('referer'),
        'cost' => $cost,
        'impressed_at' => now(),
    ]);

    // Update campaign spent (async via queue for performance)
    if ($cost > 0) {
        DB::table('ad_campaigns')
```

```
            ->where('id', $creative->campaign_id)
            ->increment('spent', $cost);
    }

    // Update inventory stats
    AdInventory::updateOrCreate(
        [
            'placement_id' => $placement->id,
            'community_id' => $communityId,
            'date' => now()->toDateString(),
        ],
        []
    )->increment('delivered_impressions');

    return $impression;
}

/**
 * Record a click
 */
public function recordClick(int $impressionId): ?string
{
    $impression = AdImpression::with('creative.campaign')->find($impressionId);
    if (!$impression) {
        return null;
    }

    // Check for click fraud (same IP clicking multiple times)
    $existingClick = AdClick::where('impression_id', $impressionId)->exists();
    if ($existingClick) {
        return $impression->creative->click_url;
    }

    // Calculate click cost
    $cost = 0;
    if ($impression->creative->campaign->type === 'cpc') {
        $cost = $impression->placement->base_cpc ?? 0;
    }

    AdClick::create([
        'impression_id' => $impressionId,
        'creative_id' => $impression->creative_id,
        'ip_hash' => request()->ip() ? hash('sha256', request()->ip()) : null,
        'cost' => $cost,
        'clicked_at' => now(),
```

```
        'clicked_at' => now(),
    ]);

    // Update campaign spent for CPC
    if ($cost > 0) {
        DB::table('ad_campaigns')
            ->where('id', $impression->creative->campaign_id)
            ->increment('spent', $cost);
    }

    return $impression->creative->click_url;
}

/**
 * Get placement by platform and slot
 */
protected function getPlacement(string $platform, string $slot): ?AdPlacement
{
    return Cache::remember(
        "placement_{$platform}_{$slot}",
        3600,
        fn() => AdPlacement::where('platform', $platform)
                    ->where('slot', $slot)
                    ->first()
    );
}

/**
 * Get campaign performance stats
 */
public function getCampaignStats(AdCampaign $campaign): array
{
    $impressions = AdImpression::whereIn('creative_id', $campaign->creatives->pluck('id'))
        ->count();

    $clicks = AdClick::whereIn('creative_id', $campaign->creatives->pluck('id'))
        ->count();

    return [
        'impressions' => $impressions,
        'clicks' => $clicks,
        'ctr' => $impressions > 0 ? round(($clicks / $impressions) * 100, 2) : 0,
        'spent' => $campaign->spent,
        'remaining' => $campaign->remaining_budget,
        'budget_utilization' => $campaign->budget > 0
            ? round(($campaign->spent / $campaign->budget) * 100, 1)
```

```
            : 0,
        ];
    }
}
```

## 3.2 Email Generator Service

php

```php
<?php
// app/Services/EmailGeneratorService.php

namespace App\Services;

use App\Models\Community;
use App\Models\EmailCampaign;
use App\Models\EmailSubscriber;
use App\Models\EmailTemplate;
use App\Models\EmailSend;
use App\Services\AIContentService;
use Illuminate\Support\Facades\DB;
use Illuminate\Support\Collection;


class EmailGeneratorService
{
    public function __construct(
        protected AIContentService $aiService,
        protected EmailDeliveryService $deliveryService
    ) {}

    /**
     * Generate daily digest for a community
     */
    public function generateDailyDigest(Community $community): EmailCampaign
    {
        // Gather content for digest
        $content = $this->gatherDigestContent($community);

        // Generate AI-written summaries
        $aiContent = $this->aiService->generateDigestContent($community, $content);

        // Get template
        $template = EmailTemplate::where('slug', 'daily-digest')
```

```
            ->where('is_active', true)
            ->first();

        // Create campaign
        $campaign = EmailCampaign::create([
            'community_id' => $community->id,
            'template_id' => $template?->id,
            'name' => "Daily Digest - {$community->name} - " . now()->format('M j, Y'),
            'type' => 'daily_digest',
            'status' => 'scheduled',
            'subject' => $aiContent['subject'],
            'preview_text' => $aiContent['preview'],
            'html_content' => $this->renderDigestHtml($community, $content, $aiContent, $template),
            'text_content' => $this->renderDigestText($community, $content, $aiContent),
            'segment' => ['type' => 'reader', 'preferences.daily_digest' => true],
            'scheduled_at' => $this->getOptimalSendTime($community, 'daily_digest'),
        ]);

        // Queue recipients
        $this->queueRecipients($campaign);

        return $campaign;
    }

    /**
     * Generate weekly newsletter for a community
     */
    public function generateWeeklyNewsletter(Community $community): EmailCampaign
    {
        // Gather week's content
        $content = $this->gatherWeeklyContent($community);

        // Generate AI editorial
        $aiContent = $this->aiService->generateNewsletterContent($community, $content);

        $template = EmailTemplate::where('slug', 'weekly-newsletter')
            ->where('is_active', true)
            ->first();

        $campaign = EmailCampaign::create([
            'community_id' => $community->id,
            'template_id' => $template?->id,
            'name' => "Weekly Newsletter - {$community->name} - Week of " . now()->format('M j'),
            'type' => 'weekly_newsletter',
            'status' => 'scheduled',
```

```
            'subject' => $aiContent['subject'],
            'preview_text' => $aiContent['preview'],
            'html_content' => $this->renderNewsletterHtml($community, $content, $aiContent, $template),
            'text_content' => $this->renderNewsletterText($community, $content, $aiContent),
            'segment' => ['preferences.weekly_newsletter' => true],
            'scheduled_at' => $this->getOptimalSendTime($community, 'weekly_newsletter'),
        ]);

        $this->queueRecipients($campaign);

        return $campaign;
    }

    /**
     * Generate breaking news alert
     */
    public function generateBreakingNews(Community $community, array $newsData): EmailCampaign
    {
        $campaign = EmailCampaign::create([
            'community_id' => $community->id,
            'name' => "Breaking: {$newsData['headline']}",
            'type' => 'breaking_news',
            'status' => 'sending', // Send immediately
            'subject' => "🔴 Breaking: {$newsData['headline']}",
            'preview_text' => substr($newsData['summary'], 0, 150),
            'html_content' => $this->renderBreakingNewsHtml($community, $newsData),
            'text_content' => $this->renderBreakingNewsText($community, $newsData),
            'segment' => ['type' => 'reader', 'preferences.breaking_news' => true],
            'started_at' => now(),
        ]);

        $this->queueRecipients($campaign, true); // Priority queue

        return $campaign;
    }

    /**
     * Generate SMB performance report
     */
    public function generateSmbReport(Community $community): EmailCampaign
    {
        $template = EmailTemplate::where('slug', 'smb-weekly-report')
            ->where('is_active', true)
            ->first();
```

```
    $campaign = EmailCampaign::create([
        'community_id' => $community->id,
        'template_id' => $template?->id,
        'name' => "SMB Weekly Report - {$community->name} - " . now()->format('M j'),
        'type' => 'smb_report',
        'status' => 'scheduled',
        'subject' => "Your Weekly Performance Report - " . now()->format('M j'),
        'preview_text' => "See how your business performed this week",
        'segment' => ['type' => 'smb'],
        'scheduled_at' => now()->next('Monday')->setTime(7, 0),
    ]);

    // SMB reports are personalized per business, handled differently
    $this->queueSmbReportRecipients($campaign);

    return $campaign;
}

/**
 * Gather content for daily digest
 */
protected function gatherDigestContent(Community $community): array
{
    return [
        'top_stories' => DB::table('articles')
            ->where('community_id', $community->id)
            ->where('published_at', '>=', now()->subDay())
            ->where('status', 'published')
            ->orderByDesc('view_count')
            ->limit(5)
            ->get(),

        'upcoming_events' => DB::table('events')
            ->where('community_id', $community->id)
            ->where('start_date', '>=', now())
            ->where('start_date', '<=', now()->addDays(7))
            ->where('status', 'published')
            ->orderBy('start_date')
            ->limit(5)
            ->get(),

        'featured_businesses' => DB::table('businesses')
            ->where('community_id', $community->id)
            ->where('is_featured', true)
            ->inRandomOrder()
```

```
            ->limit(3)
            ->get(),

        'weather' => $this->getWeatherForecast($community),
    ];
}

/**
 * Get optimal send time for community based on timezone and type
 */
protected function getOptimalSendTime(Community $community, string $type): \Carbon\Carbon
{
    $timezone = $community->timezone ?? 'America/New_York';

    $times = [
        'daily_digest' => '06:00',
        'weekly_newsletter' => '08:00',
        'smb_report' => '07:00',
    ];

    $time = $times[$type] ?? '09:00';

    return now($timezone)->setTimeFromTimeString($time)->setTimezone('UTC');
}

/**
 * Queue recipients for campaign
 */
protected function queueRecipients(EmailCampaign $campaign, bool $priority = false): void
{
    $query = EmailSubscriber::where('community_id', $campaign->community_id)
        ->where('status', 'active');

    // Apply segment filters
    if ($segment = $campaign->segment) {
        foreach ($segment as $key => $value) {
            if (str_starts_with($key, 'preferences.')) {
                $prefKey = str_replace('preferences.', '', $key);
                $query->whereJsonContains("preferences->{$prefKey}", $value);
            } else {
                $query->where($key, $value);
            }
        }
    }
```

```
        $subscribers = $query->get();
        $campaign->update(['total_recipients' => $subscribers->count()]);

        foreach ($subscribers->chunk(100) as $chunk) {
            $sends = $chunk->map(fn($sub) => [
                'campaign_id' => $campaign->id,
                'subscriber_id' => $sub->id,
                'status' => 'queued',
                'created_at' => now(),
                'updated_at' => now(),
            ])->toArray();

            EmailSend::insert($sends);
        }
    }

    /**
     * Render digest HTML with ads
     */
    protected function renderDigestHtml(Community $community, array $content, array $aiContent, ?EmailTemplate $templa
    {
        // Get ads for email
        $ads = app(AdServerService::class)->getEmailAds($community->id, 'daily_digest', 2);

        return view('emails.digest', [
            'community' => $community,
            'content' => $content,
            'aiContent' => $aiContent,
            'ads' => $ads,
        ])->render();
    }

    // Additional render methods...
}
```

## 3.3 Emergency Broadcast Service

```
php

<?php
// app/Services/EmergencyBroadcastService.php

namespace App\Services;
```

```php
use App\Models\Community;
use App\Models\EmergencyAlert;
use App\Models\EmergencySubscription;
use App\Models\EmergencyDelivery;
use App\Models\EmergencyAuditLog;
use App\Jobs\SendEmergencyEmail;
use App\Jobs\SendEmergencySms;
use Illuminate\Support\Facades\DB;

class EmergencyBroadcastService
{
    public function __construct(
        protected EmailDeliveryService $emailService,
        protected SmsService $smsService
    ) {}

    /**
     * Create and optionally publish an emergency alert
     */
    public function createAlert(array $data, ?int $userId = null, ?int $municipalPartnerId = null): EmergencyAlert
    {
        $alert = EmergencyAlert::create([
            'community_id' => $data['community_id'],
            'created_by' => $userId,

            'municipal_partner_id' => $municipalPartnerId,
            'priority' => $data['priority'],
            'category' => $data['category'],
            'title' => $data['title'],
            'message' => $data['message'],
            'instructions' => $data['instructions'] ?? null,
            'source' => $data['source'] ?? null,
            'source_url' => $data['source_url'] ?? null,
            'status' => $data['publish_immediately'] ?? false ? 'active' : 'draft',
            'published_at' => $data['publish_immediately'] ?? false ? now() : null,
            'expires_at' => $data['expires_at'] ?? null,
            'delivery_channels' => $data['channels'] ?? ['email'],
        ]);

        $this->logAction($alert, 'created', $userId, $municipalPartnerId);

        if ($alert->status === 'active') {
            $this->broadcast($alert);
        }

        return $alert;
```

```
    return $alert;
}

/**
 * Publish a draft alert
 */
public function publishAlert(EmergencyAlert $alert, ?int $userId = null): EmergencyAlert
{
    $alert->update([
        'status' => 'active',
        'published_at' => now(),
    ]);

    $this->logAction($alert, 'published', $userId);
    $this->broadcast($alert);

    return $alert;
}

/**
 * Broadcast alert to all eligible subscribers
 */
public function broadcast(EmergencyAlert $alert): void
{
    $channels = $alert->delivery_channels ?? ['email'];

    // Get eligible subscriptions
    $subscriptions = EmergencySubscription::query()
        ->whereHas('subscriber', function ($query) use ($alert) {
            $query->where('community_id', $alert->community_id)
                ->where('status', 'active');
        })
        ->with('subscriber')
        ->get()
        ->filter(fn($sub) => $sub->shouldReceiveAlert($alert));

    foreach ($subscriptions as $subscription) {
        // Queue email delivery
        if (in_array('email', $channels) && $subscription->email_enabled) {
            $this->queueEmailDelivery($alert, $subscription);
        }

        // Queue SMS delivery for critical/urgent if enabled
        if (in_array('sms', $channels) && $subscription->canReceiveSms()) {
            if (in_array($alert->priority, ['critical', 'urgent'])) {
                $this->queueSmsDelivery($alert, $subscription);
```

```
                $this->queueSmsDelivery($alert, $subscription);
            }
        }
    }

    // Update sent counts
    $alert->update([
        'email_sent' => EmergencyDelivery::where('alert_id', $alert->id)
            ->where('channel', 'email')
            ->count(),
        'sms_sent' => EmergencyDelivery::where('alert_id', $alert->id)
            ->where('channel', 'sms')
            ->count(),
    ]);
}

/**
 * Queue email delivery
 */
protected function queueEmailDelivery(EmergencyAlert $alert, EmergencySubscription $subscription): void
{
    $delivery = EmergencyDelivery::create([
        'alert_id' => $alert->id,
        'subscription_id' => $subscription->id,
        'channel' => 'email',
        'status' => 'queued',
    ]);

    // Use high priority queue for critical alerts
    $queue = $alert->priority === 'critical' ? 'emergency-critical' : 'emergency';

    SendEmergencyEmail::dispatch($delivery)->onQueue($queue);
}

/**
 * Queue SMS delivery
 */
protected function queueSmsDelivery(EmergencyAlert $alert, EmergencySubscription $subscription): void
{
    $delivery = EmergencyDelivery::create([
        'alert_id' => $alert->id,
        'subscription_id' => $subscription->id,
        'channel' => 'sms',
        'status' => 'queued',
    ]);
```

```
        // SMS always uses critical queue
        SendEmergencySms::dispatch($delivery)->onQueue('emergency-critical');
    }

    /**
     * Cancel an active alert
     */
    public function cancelAlert(EmergencyAlert $alert, ?int $userId = null, ?string $reason = null): EmergencyAlert
    {
        $alert->update([
            'status' => 'cancelled',
        ]);

        $this->logAction($alert, 'cancelled', $userId, null, ['reason' => $reason]);

        // Cancel any pending deliveries
        EmergencyDelivery::where('alert_id', $alert->id)
            ->where('status', 'queued')
            ->update(['status' => 'failed', 'error_message' => 'Alert cancelled']);

        return $alert;
    }

    /**
     * Log audit action
     */
    protected function logAction(
        EmergencyAlert $alert,
        string $action,
        ?int $userId = null,
        ?int $municipalPartnerId = null,
        ?array $changes = null
    ): void {
        EmergencyAuditLog::create([
            'alert_id' => $alert->id,
            'user_id' => $userId,
            'municipal_partner_id' => $municipalPartnerId,
            'action' => $action,
            'changes' => $changes,
            'ip_address' => request()->ip(),
            'user_agent' => request()->userAgent(),
        ]);
    }

    /**
```

```
     * Get delivery statistics for an alert
     */
    public function getDeliveryStats(EmergencyAlert $alert): array
    {
        return [
            'email' => [
                'queued' => $alert->deliveries()->where('channel', 'email')->where('status', 'queued')->count(),
                'sent' => $alert->deliveries()->where('channel', 'email')->where('status', 'sent')->count(),
                'delivered' => $alert->deliveries()->where('channel', 'email')->where('status', 'delivered')->count(),
                'failed' => $alert->deliveries()->where('channel', 'email')->where('status', 'failed')->count(),
            ],
            'sms' => [
                'queued' => $alert->deliveries()->where('channel', 'sms')->where('status', 'queued')->count(),
                'sent' => $alert->deliveries()->where('channel', 'sms')->where('status', 'sent')->count(),
                'delivered' => $alert->deliveries()->where('channel', 'sms')->where('status', 'delivered')->count(),
                'failed' => $alert->deliveries()->where('channel', 'sms')->where('status', 'failed')->count(),
            ],
        ];
    }
}
```

## 3.4 AI Content Service

php

```
<?php
// app/Services/AIContentService.php

namespace App\Services;

use App\Models\Community;
use Illuminate\Support\Facades\Http;
use Illuminate\Support\Facades\Cache;

class AIContentService
{
    protected string $apiKey;
    protected string $model = 'claude-sonnet-4-20250514';

    public function __construct()
    {
        $this->apiKey = config('services.anthropic.api_key');
    }
```

```php
    /**
     * Generate daily digest content
     */
    public function generateDigestContent(Community $community, array $content): array
    {
        $prompt = $this->buildDigestPrompt($community, $content);

        $response = $this->callClaude($prompt);

        return $this->parseDigestResponse($response);
    }

    /**
     * Generate newsletter editorial content
     */
    public function generateNewsletterContent(Community $community, array $content): array
    {
        $prompt = $this->buildNewsletterPrompt($community, $content);

        $response = $this->callClaude($prompt);

        return $this->parseNewsletterResponse($response);
    }


    /**
     * Generate subject line variations for A/B testing
     */
    public function generateSubjectLines(string $topic, string $type, int $count = 3): array
    {
        $prompt = "Generate {$count} compelling email subject lines for a {$type} email about: {$topic}\n\nRequirements:\n-

        $response = $this->callClaude($prompt);

        return json_decode($response, true) ?? [];
    }

    /**
     * Call Claude API
     */
    protected function callClaude(string $prompt, ?string $systemPrompt = null): string
    {
        $messages = [
            ['role' => 'user', 'content' => $prompt]
        ];
```

```
        $payload = [
            'model' => $this->model,
            'max_tokens' => 2000,
            'messages' => $messages,
        ];

        if ($systemPrompt) {
            $payload['system'] = $systemPrompt;
        }

        $response = Http::withHeaders([
            'x-api-key' => $this->apiKey,
            'anthropic-version' => '2023-06-01',
            'content-type' => 'application/json',
        ])->post('https://api.anthropic.com/v1/messages', $payload);

        if (!$response->successful()) {
            throw new \Exception('Claude API error: ' . $response->body());
        }

        return $response->json('content.0.text', '');
    }

    /**
     * Build prompt for daily digest
     */
    protected function buildDigestPrompt(Community $community, array $content): string
    {
        $stories = collect($content['top_stories'])->map(fn($s) => "- {$s->title}")->join("\n");
        $events = collect($content['upcoming_events'])->map(fn($e) => "- {$e->title} ({$e->start_date})")->join("\n");

        return <<<PROMPT
You are writing a daily news digest email for {$community->name}, {$community->state}.

Today's top stories:
{$stories}

Upcoming events this week:
{$events}

Generate:
1. A compelling subject line (under 60 chars)
2. A preview text (under 150 chars)
3. A brief intro paragraph (2-3 sentences) that ties together the day's news
```

```
4. Brief summaries (1-2 sentences each) for each top story

Return as JSON:
{
  "subject": "...",
  "preview": "...",
  "intro": "...",
  "story_summaries": ["...", "..."]
}
PROMPT;
    }

    /**
     * Parse digest response
     */
    protected function parseDigestResponse(string $response): array
    {
        // Extract JSON from response
        preg_match('/\{[\s\S]*\}/', $response, $matches);

        if (empty($matches)) {
            return [
                'subject' => 'Your Daily News Digest',
                'preview' => 'The latest news from your community',
                'intro' => '',
                'story_summaries' => [],
            ];
        }

        return json_decode($matches[0], true) ?? [];
    }
}
```

---

# PHASE 4: LARAVEL CONTROLLERS

Controllers are thin - they validate input, call services, and return Inertia responses.

## 4.1 Admin Advertising Controllers

```
php

<?php
```

```php
// app/Http/Controllers/Admin/Advertising/CampaignController.php

namespace App\Http\Controllers\Admin\Advertising;

use App\Http\Controllers\Controller;
use App\Models\AdCampaign;
use App\Models\Business;
use App\Services\AdServerService;
use Illuminate\Http\Request;
use Inertia\Inertia;

class CampaignController extends Controller
{
    public function __construct(protected AdServerService $adService)
    {}

    public function index(Request $request)
    {
        $campaigns = AdCampaign::query()
            ->with('advertiser:id,name')
            ->withCount('creatives')
            ->when($request->status, fn($q, $s) => $q->where('status', $s))
            ->when($request->search, fn($q, $s) => $q->where('name', 'like', "%{$s}%"))
            ->orderByDesc('created_at')
            ->paginate(25);


        // Add stats to each campaign
        $campaigns->getCollection()->transform(function ($campaign) {
            $campaign->stats = $this->adService->getCampaignStats($campaign);
            return $campaign;
        });

        return Inertia::render('Admin/Advertising/Campaigns/Index', [
            'campaigns' => $campaigns,
            'filters' => $request->only(['status', 'search']),
            'statuses' => ['draft', 'pending', 'active', 'paused', 'completed', 'cancelled'],
        ]);
    }

    public function create()
    {
        return Inertia::render('Admin/Advertising/Campaigns/Create', [
            'advertisers' => Business::select('id', 'name')->orderBy('name')->get(),
            'platforms' => ['day_news', 'goeventcity', 'downtown_guide', 'alphasite_community', 'golocalvoices'],
```

```
            'campaignTypes' => ['cpm', 'cpc', 'flat_rate', 'sponsored'],
        ]);
    }

    public function store(Request $request)
    {
        $validated = $request->validate([
            'advertiser_id' => 'required|exists:businesses,id',
            'name' => 'required|string|max:255',
            'description' => 'nullable|string',
            'type' => 'required|in:cpm,cpc,flat_rate,sponsored',
            'budget' => 'required|numeric|min:0',
            'daily_budget' => 'nullable|numeric|min:0',
            'start_date' => 'required|date|after_or_equal:today',
            'end_date' => 'required|date|after:start_date',
            'platforms' => 'required|array|min:1',
            'targeting' => 'nullable|array',
        ]);

        $campaign = AdCampaign::create($validated);

        return redirect()
            ->route('admin.advertising.campaigns.show', $campaign)
            ->with('success', 'Campaign created successfully.');
    }

    public function show(AdCampaign $campaign)
    {
        $campaign->load(['advertiser', 'creatives']);

        return Inertia::render('Admin/Advertising/Campaigns/Show', [
            'campaign' => $campaign,
            'stats' => $this->adService->getCampaignStats($campaign),
            'dailyStats' => $this->getDailyStats($campaign),
        ]);
    }

    public function edit(AdCampaign $campaign)
    {
        return Inertia::render('Admin/Advertising/Campaigns/Edit', [
            'campaign' => $campaign,
            'advertisers' => Business::select('id', 'name')->orderBy('name')->get(),
            'platforms' => ['day_news', 'goeventcity', 'downtown_guide', 'alphasite_community', 'golocalvoices'],
            'campaignTypes' => ['cpm', 'cpc', 'flat_rate', 'sponsored'],
        ]);
    }
```

```
    }

    public function update(Request $request, AdCampaign $campaign)
    {
        $validated = $request->validate([
            'name' => 'required|string|max:255',
            'description' => 'nullable|string',
            'budget' => 'required|numeric|min:' . $campaign->spent,
            'daily_budget' => 'nullable|numeric|min:0',
            'end_date' => 'required|date|after:start_date',
            'platforms' => 'required|array|min:1',
            'targeting' => 'nullable|array',
        ]);

        $campaign->update($validated);

        return redirect()
            ->route('admin.advertising.campaigns.show', $campaign)
            ->with('success', 'Campaign updated successfully.');
    }

    public function updateStatus(Request $request, AdCampaign $campaign)
    {
        $validated = $request->validate([
            'status' => 'required|in:active,paused,cancelled',
        ]);

        $campaign->update($validated);

        return back()->with('success', 'Campaign status updated.');
    }

    protected function getDailyStats(AdCampaign $campaign): array
    {
        // Get last 30 days of stats
        return \DB::table('ad_impressions')
            ->selectRaw('DATE(impressed_at) as date, COUNT(*) as impressions')
            ->whereIn('creative_id', $campaign->creatives->pluck('id'))
            ->where('impressed_at', '>=', now()->subDays(30))
            ->groupBy('date')
            ->orderBy('date')
            ->get()
            ->toArray();
    }
}
```

## 4.2 Admin Email Controllers

php

```php
<?php
// app/Http/Controllers/Admin/Email/CampaignController.php

namespace App\Http\Controllers\Admin\Email;

use App\Http\Controllers\Controller;
use App\Models\EmailCampaign;
use App\Models\EmailTemplate;
use App\Models\Community;
use App\Services\EmailGeneratorService;
use Illuminate\Http\Request;
use Inertia\Inertia;

class CampaignController extends Controller
{
    public function __construct(protected EmailGeneratorService $emailService)
    {}

    public function index(Request $request)
    {

        $campaigns = EmailCampaign::query()
            ->with('community:id,name')
            ->when($request->community_id, fn($q, $c) => $q->where('community_id', $c))
            ->when($request->type, fn($q, $t) => $q->where('type', $t))
            ->when($request->status, fn($q, $s) => $q->where('status', $s))
            ->orderByDesc('created_at')
            ->paginate(25);

        return Inertia::render('Admin/Email/Campaigns/Index', [
            'campaigns' => $campaigns,
            'filters' => $request->only(['community_id', 'type', 'status']),
            'communities' => Community::select('id', 'name')->orderBy('name')->get(),
            'types' => ['daily_digest', 'breaking_news', 'weekly_newsletter', 'smb_report', 'custom'],
            'statuses' => ['draft', 'scheduled', 'sending', 'sent', 'cancelled'],
        ]);
    }

    public function show(EmailCampaign $campaign)
    {
```

```
        $campaign->load(['community', 'template']);

        return Inertia::render('Admin/Email/Campaigns/Show', [
            'campaign' => $campaign,
            'stats' => [
                'open_rate' => $campaign->open_rate,
                'click_rate' => $campaign->click_rate,
                'bounce_rate' => $campaign->delivered_count > 0
                    ? round(($campaign->bounced_count / $campaign->delivered_count) * 100, 2)
                    : 0,
            ],
        ]);
    }

    public function generateDigest(Request $request)
    {
        $validated = $request->validate([
            'community_id' => 'required|exists:communities,id',
        ]);

        $community = Community::findOrFail($validated['community_id']);
        $campaign = $this->emailService->generateDailyDigest($community);

        return redirect()
            ->route('admin.email.campaigns.show', $campaign)
            ->with('success', 'Daily digest generated and scheduled.');
    }

    public function generateNewsletter(Request $request)
    {
        $validated = $request->validate([
            'community_id' => 'required|exists:communities,id',
        ]);

        $community = Community::findOrFail($validated['community_id']);
        $campaign = $this->emailService->generateWeeklyNewsletter($community);

        return redirect()
            ->route('admin.email.campaigns.show', $campaign)
            ->with('success', 'Weekly newsletter generated and scheduled.');
    }
}
```

## 4.3 Emergency Controllers

php

```php
<?php
// app/Http/Controllers/Admin/Emergency/AlertController.php

namespace App\Http\Controllers\Admin\Emergency;

use App\Http\Controllers\Controller;
use App\Models\EmergencyAlert;
use App\Models\Community;
use App\Services\EmergencyBroadcastService;
use Illuminate\Http\Request;
use Inertia\Inertia;

class AlertController extends Controller
{
    public function __construct(protected EmergencyBroadcastService $emergencyService)
    {}

    public function index(Request $request)
    {
        $alerts = EmergencyAlert::query()
            ->with('community:id,name')
            ->when($request->community_id, fn($q, $c) => $q->where('community_id', $c))
            ->when($request->priority, fn($q, $p) => $q->where('priority', $p))

            ->when($request->status, fn($q, $s) => $q->where('status', $s))
            ->orderByDesc('created_at')
            ->paginate(25);

        return Inertia::render('Admin/Emergency/Alerts/Index', [
            'alerts' => $alerts,
            'filters' => $request->only(['community_id', 'priority', 'status']),
            'communities' => Community::select('id', 'name')->orderBy('name')->get(),
            'priorities' => ['critical', 'urgent', 'advisory', 'info'],
            'statuses' => ['draft', 'active', 'expired', 'cancelled'],
        ]);
    }

    public function create()
    {
        return Inertia::render('Admin/Emergency/Alerts/Create', [
            'communities' => Community::select('id', 'name')->orderBy('name')->get(),
            'priorities' => [
```

```
            ['value' => 'critical', 'label' => 'Critical', 'color' => 'red', 'description' => 'Imminent threat to life/safety'],
            ['value' => 'urgent', 'label' => 'Urgent', 'color' => 'orange', 'description' => 'Significant event requiring attention'],
            ['value' => 'advisory', 'label' => 'Advisory', 'color' => 'yellow', 'description' => 'Awareness recommended'],
            ['value' => 'info', 'label' => 'Info', 'color' => 'blue', 'description' => 'General notice'],
        ],
        'categories' => ['weather', 'crime', 'health', 'utility', 'traffic', 'government', 'school', 'amber'],
        'channels' => ['email', 'sms'],
    ]);
}

public function store(Request $request)
{
    $validated = $request->validate([
        'community_id' => 'required|exists:communities,id',
        'priority' => 'required|in:critical,urgent,advisory,info',
        'category' => 'required|string',
        'title' => 'required|string|max:255',
        'message' => 'required|string',
        'instructions' => 'nullable|string',
        'source' => 'nullable|string',
        'source_url' => 'nullable|url',
        'expires_at' => 'nullable|date|after:now',
        'channels' => 'required|array|min:1',
        'publish_immediately' => 'boolean',
    ]);

    $alert = $this->emergencyService->createAlert($validated, auth()->id());

    $message = $validated['publish_immediately'] ?? false
        ? 'Alert published and broadcast initiated.'
        : 'Alert saved as draft.';

    return redirect()
        ->route('admin.emergency.alerts.show', $alert)
        ->with('success', $message);
}

public function show(EmergencyAlert $alert)
{
    $alert->load(['community', 'creator', 'municipalPartner']);

    return Inertia::render('Admin/Emergency/Alerts/Show', [
        'alert' => $alert,
        'deliveryStats' => $this->emergencyService->getDeliveryStats($alert),
        'auditLog' => $alert->auditLogs()->with('user')->latest()->limit(20)->get(),
```

```
        ]);
    }

    public function publish(EmergencyAlert $alert)
    {
        if ($alert->status !== 'draft') {
            return back()->with('error', 'Only draft alerts can be published.');
        }

        $this->emergencyService->publishAlert($alert, auth()->id());

        return back()->with('success', 'Alert published and broadcast initiated.');
    }

    public function cancel(Request $request, EmergencyAlert $alert)
    {
        if (!in_array($alert->status, ['draft', 'active'])) {
            return back()->with('error', 'This alert cannot be cancelled.');
        }

        $this->emergencyService->cancelAlert($alert, auth()->id(), $request->reason);

        return back()->with('success', 'Alert cancelled.');
    }
}
```

---

# PHASE 5: INERTIA PAGES (Simple Display)

These are simple React pages that receive data from Laravel and display it. Minimal state, just display and navigation.

## 5.1 Page Structure

```
resources/js/Pages/Admin/
├── Advertising/
│   ├── Dashboard.tsx
│   └── Campaigns/
│       ├── Index.tsx
│       ├── Create.tsx
│       ├── Show.tsx
│       └── Edit.tsx
├── Email/
```

```
│   ├── Dashboard.tsx
│   ├── Campaigns/
│   │   ├── Index.tsx
│   │   └── Show.tsx
│   ├── Subscribers/
│   │   └── Index.tsx
│   └── Templates/
│       └── Index.tsx
└── Emergency/
    ├── Dashboard.tsx
    └── Alerts/
        ├── Index.tsx
        ├── Create.tsx
        └── Show.tsx
```

## 5.2 Example Page - Campaign Index

tsx

```tsx
// resources/js/Pages/Admin/Advertising/Campaigns/Index.tsx

import { Head, Link, router } from '@inertiajs/react';
import AdminLayout from '@/Layouts/AdminLayout';
import { PageProps } from '@/types';

interface Campaign {
    id: number;
    uuid: string;
    name: string;
    status: string;
    type: string;
    budget: string;
    spent: string;
    start_date: string;
    end_date: string;
    advertiser: { id: number; name: string };
    creatives_count: number;
    stats: {
        impressions: number;
        clicks: number;
        ctr: number;
        budget_utilization: number;
    };
}
```

```tsx
interface Props extends PageProps {
    campaigns: {
        data: Campaign[];
        links: any;
        meta: any;
    };
    filters: {
        status?: string;
        search?: string;
    };
    statuses: string[];
}

export default function Index({ campaigns, filters, statuses }: Props) {
    const handleFilter = (key: string, value: string) => {
        router.get(route('admin.advertising.campaigns.index'), {
            ...filters,
            [key]: value || undefined,
        }, { preserveState: true });
    };

    const statusColors: Record<string, string> = {
        draft: 'bg-gray-100 text-gray-800',
        pending: 'bg-yellow-100 text-yellow-800',
        active: 'bg-green-100 text-green-800',
        paused: 'bg-orange-100 text-orange-800',
        completed: 'bg-blue-100 text-blue-800',
        cancelled: 'bg-red-100 text-red-800',
    };

    return (
        <AdminLayout>
            <Head title="Ad Campaigns" />

            <div className="px-4 sm:px-6 lg:px-8">
                {/* Header */}
                <div className="sm:flex sm:items-center sm:justify-between">
                    <div>
                        <h1 className="text-2xl font-semibold text-gray-900">Ad Campaigns</h1>
                        <p className="mt-2 text-sm text-gray-700">
                            Manage advertising campaigns across all platforms
                        </p>
                    </div>
                    <Link
```

```
        href={route('admin.advertising.campaigns.create')}
        className="inline-flex items-center rounded-md bg-blue-600 px-3 py-2 text-sm font-semibold text-white hov
    >
        Create Campaign
    </Link>
</div>

{/* Filters */}
<div className="mt-4 flex gap-4">
    <select
        value={filters.status || ''}
        onChange={(e) => handleFilter('status', e.target.value)}
        className="rounded-md border-gray-300 text-sm"
    >
        <option value="">All Statuses</option>
        {statuses.map((status) => (
            <option key={status} value={status}>
                {status.charAt(0).toUpperCase() + status.slice(1)}
            </option>
        ))}
    </select>
    <input
        type="text"
        placeholder="Search campaigns..."
        value={filters.search || ''}
        onChange={(e) => handleFilter('search', e.target.value)}
        className="rounded-md border-gray-300 text-sm"
    />
</div>

{/* Table */}
<div className="mt-6 overflow-hidden shadow ring-1 ring-black ring-opacity-5 rounded-lg">
    <table className="min-w-full divide-y divide-gray-300">
        <thead className="bg-gray-50">
            <tr>
                <th className="py-3.5 pl-4 pr-3 text-left text-sm font-semibold text-gray-900">Campaign</th>
                <th className="px-3 py-3.5 text-left text-sm font-semibold text-gray-900">Status</th>
                <th className="px-3 py-3.5 text-left text-sm font-semibold text-gray-900">Budget</th>
                <th className="px-3 py-3.5 text-left text-sm font-semibold text-gray-900">Performance</th>
                <th className="px-3 py-3.5 text-left text-sm font-semibold text-gray-900">Dates</th>
                <th className="relative py-3.5 pl-3 pr-4"><span className="sr-only">Actions</span></th>
            </tr>
        </thead>
        <tbody className="divide-y divide-gray-200 bg-white">
            {campaigns.data.map((campaign) => (
```

```
                <tr key={campaign.id}>
                  <td className="py-4 pl-4 pr-3">
                    <div className="font-medium text-gray-900">{campaign.name}</div>
                    <div className="text-sm text-gray-500">{campaign.advertiser.name}</div>
                  </td>
                  <td className="px-3 py-4">
                    <span className={`inline-flex rounded-full px-2 py-1 text-xs font-medium ${statusColors[campaig
                      {campaign.status}
                    </span>
                  </td>
                  <td className="px-3 py-4">
                    <div className="text-sm text-gray-900">
                      ${parseFloat(campaign.spent).toLocaleString()} / ${parseFloat(campaign.budget).toLocaleString
                    </div>
                    <div className="mt-1 h-2 w-24 bg-gray-200 rounded-full overflow-hidden">
                      <div
                        className="h-full bg-blue-600"
                        style={{ width: `${campaign.stats.budget_utilization}%` }}
                      />
                    </div>
                  </td>
                  <td className="px-3 py-4 text-sm text-gray-500">
                    <div>{campaign.stats.impressions.toLocaleString()} impr</div>
                    <div>{campaign.stats.clicks.toLocaleString()} clicks ({campaign.stats.ctr}% CTR)</div>
                  </td>
                  <td className="px-3 py-4 text-sm text-gray-500">
                    <div>{campaign.start_date}</div>
                    <div>{campaign.end_date}</div>
                  </td>
                  <td className="relative py-4 pl-3 pr-4 text-right text-sm">
                    <Link
                      href={route('admin.advertising.campaigns.show', campaign.id)}
                      className="text-blue-600 hover:text-blue-900"
                    >
                      View
                    </Link>
                  </td>
                </tr>
              ))}
            </tbody>
          </table>
        </div>

        {/* Pagination - Use Inertia Link components */}
        {campaigns.links && (
```

```
            <div className="mt-4 flex justify-center gap-2">
              {campaigns.links.map((link: any, i: number) => (
                <Link
                  key={i}
                  href={link.url || '#'}
                  className={`px-3 py-1 rounded ${link.active ? 'bg-blue-600 text-white' : 'bg-gray-100'}`}
                  dangerouslySetInnerHTML={{ __html: link.label }}
                />
              ))}
            </div>
          )}
        </div>
      </AdminLayout>
    );
}
```

## 5.3 Example Page - Emergency Alert Create

tsx

```
// resources/js/Pages/Admin/Emergency/Alerts/Create.tsx

import { Head, useForm, router } from '@inertiajs/react';
import AdminLayout from '@/Layouts/AdminLayout';
import { PageProps } from '@/types';
import { FormEvent } from 'react';


interface Priority {
    value: string;
    label: string;
    color: string;
    description: string;
}

interface Props extends PageProps {
    communities: { id: number; name: string }[];
    priorities: Priority[];
    categories: string[];
    channels: string[];
}

export default function Create({ communities, priorities, categories, channels }: Props) {
    const { data, setData, post, processing, errors } = useForm({
```

```
    community_id: '',
    priority: 'advisory',
    category: '',
    title: '',
    message: '',
    instructions: '',
    source: '',
    source_url: '',
    expires_at: '',
    channels: ['email'],
    publish_immediately: false,
});

const handleSubmit = (e: FormEvent) => {
    e.preventDefault();
    post(route('admin.emergency.alerts.store'));
};

const toggleChannel = (channel: string) => {
    const newChannels = data.channels.includes(channel)
        ? data.channels.filter(c => c !== channel)
        : [...data.channels, channel];
    setData('channels', newChannels);
};

const selectedPriority = priorities.find(p => p.value === data.priority);

return (
    <AdminLayout>
        <Head title="Create Emergency Alert" />

        <div className="max-w-3xl mx-auto px-4 py-8">
            <h1 className="text-2xl font-semibold text-gray-900 mb-6">Create Emergency Alert</h1>

            <form onSubmit={handleSubmit} className="space-y-6">
                {/* Community */}
                <div>
                    <label className="block text-sm font-medium text-gray-700">Community</label>
                    <select
                        value={data.community_id}
                        onChange={(e) => setData('community_id', e.target.value)}
                        className="mt-1 block w-full rounded-md border-gray-300 shadow-sm"
                    >
                        <option value="">Select community...</option>
                        {communities.map((c) => (
```

```
            <option key={c.id} value={c.id}>{c.name}</option>
          ))}
        </select>
        {errors.community_id && <p className="mt-1 text-sm text-red-600">{errors.community_id}</p>}
      </div>

      {/* Priority */}
      <div>
        <label className="block text-sm font-medium text-gray-700 mb-2">Priority Level</label>
        <div className="grid grid-cols-2 gap-3">
          {priorities.map((priority) => (
            <button
              key={priority.value}
              type="button"
              onClick={() => setData('priority', priority.value)}
              className={`p-3 rounded-lg border-2 text-left ${
                data.priority === priority.value
                  ? `border-${priority.color}-500 bg-${priority.color}-50`
                  : 'border-gray-200'
              }`}
            >
              <div className="font-medium">{priority.label}</div>
              <div className="text-xs text-gray-500">{priority.description}</div>
            </button>
          ))}
        </div>
      </div>

      {/* Category */}
      <div>
        <label className="block text-sm font-medium text-gray-700">Category</label>
        <select
          value={data.category}
          onChange={(e) => setData('category', e.target.value)}
          className="mt-1 block w-full rounded-md border-gray-300 shadow-sm"
        >
          <option value="">Select category...</option>
          {categories.map((cat) => (
            <option key={cat} value={cat}>
              {cat.charAt(0).toUpperCase() + cat.slice(1)}
            </option>
          ))}
        </select>
        {errors.category && <p className="mt-1 text-sm text-red-600">{errors.category}</p>}
      </div>
```

```
{/* Title */}
<div>
  <label className="block text-sm font-medium text-gray-700">Alert Title</label>
  <input
    type="text"
    value={data.title}
    onChange={(e) => setData('title', e.target.value)}
    className="mt-1 block w-full rounded-md border-gray-300 shadow-sm"
    placeholder="Brief, clear title for the alert"
  />
  {errors.title && <p className="mt-1 text-sm text-red-600">{errors.title}</p>}
</div>

{/* Message */}
<div>
  <label className="block text-sm font-medium text-gray-700">Message</label>
  <textarea
    value={data.message}
    onChange={(e) => setData('message', e.target.value)}
    rows={4}
    className="mt-1 block w-full rounded-md border-gray-300 shadow-sm"
    placeholder="Detailed alert message"
  />
  {errors.message && <p className="mt-1 text-sm text-red-600">{errors.message}</p>}
</div>

{/* Instructions */}
<div>
  <label className="block text-sm font-medium text-gray-700">Instructions (Optional)</label>
  <textarea
    value={data.instructions}
    onChange={(e) => setData('instructions', e.target.value)}
    rows={2}
    className="mt-1 block w-full rounded-md border-gray-300 shadow-sm"
    placeholder="What should residents do?"
  />
</div>

{/* Channels */}
<div>
  <label className="block text-sm font-medium text-gray-700 mb-2">Delivery Channels</label>
  <div className="flex gap-4">
    {channels.map((channel) => (
      <label key={channel} className="flex items-center">
```

```
            <input
              type="checkbox"
              checked={data.channels.includes(channel)}
              onChange={() => toggleChannel(channel)}
              className="rounded border-gray-300 text-blue-600"
            />
            <span className="ml-2 text-sm">{channel.toUpperCase()}</span>
          </label>
        ))}
      </div>
      {data.channels.includes('sms') && (
        <p className="mt-1 text-xs text-amber-600">
          SMS delivery only available for Critical and Urgent alerts to subscribers with SMS enabled.
        </p>
      )}
    </div>

    {/* Expiration */}
    <div>
      <label className="block text-sm font-medium text-gray-700">Expires At (Optional)</label>
      <input
        type="datetime-local"
        value={data.expires_at}
        onChange={(e) => setData('expires_at', e.target.value)}
        className="mt-1 block w-full rounded-md border-gray-300 shadow-sm"
      />
    </div>

    {/* Publish Immediately */}
    <div className="flex items-center">
      <input
        type="checkbox"
        id="publish_immediately"
        checked={data.publish_immediately}
        onChange={(e) => setData('publish_immediately', e.target.checked)}
        className="rounded border-gray-300 text-blue-600"
      />
      <label htmlFor="publish_immediately" className="ml-2 text-sm text-gray-700">
        Publish and broadcast immediately
      </label>
    </div>

    {data.publish_immediately && selectedPriority?.value === 'critical' && (
      <div className="p-4 bg-red-50 border border-red-200 rounded-md">
        <p className="text-sm text-red-800">
```

```
                        <strong>Warning:</strong> This will immediately send notifications to all subscribers
                        in the selected community via email and SMS (for SMS-enabled subscribers).
                    </p>
                </div>
            )}

            {/* Actions */}
            <div className="flex justify-end gap-3 pt-4 border-t">
                <button
                    type="button"
                    onClick={() => router.visit(route('admin.emergency.alerts.index'))}
                    className="px-4 py-2 text-sm font-medium text-gray-700 bg-white border border-gray-300 rounded-md h
                >
                    Cancel
                </button>
                <button
                    type="submit"
                    disabled={processing}
                    className="px-4 py-2 text-sm font-medium text-white bg-blue-600 border border-transparent rounded-md
                >
                    {data.publish_immediately ? 'Create & Publish' : 'Save as Draft'}
                </button>
            </div>
        </form>
    </div>
</AdminLayout>
);
}
```

---

# PHASE 6: ROUTES

php

```
<?php
// routes/web.php

use App\Http\Controllers\Admin\Advertising\CampaignController as AdCampaignController;
use App\Http\Controllers\Admin\Advertising\CreativeController;
use App\Http\Controllers\Admin\Advertising\PlacementController;
use App\Http\Controllers\Admin\Advertising\ReportController as AdReportController;
use App\Http\Controllers\Admin\Email\CampaignController as EmailCampaignController;
use App\Http\Controllers\Admin\Email\SubscriberController;
```

```php
use App\Http\Controllers\Admin\Email\TemplateController;
use App\Http\Controllers\Admin\Emergency\AlertController;
use App\Http\Controllers\Admin\Emergency\SubscriptionController as EmergencySubController;
use App\Http\Controllers\Admin\Emergency\MunicipalController;
use App\Http\Controllers\Ads\AdController;
use Illuminate\Support\Facades\Route;

/*
|--------------------------------------------------------------------------
| Ad Serving (Public)
|--------------------------------------------------------------------------
*/

Route::get('/ads/{platform}/{slot}/{community}', [AdController::class, 'serve'])->name('ads.serve');
Route::get('/ads/click/{impression}', [AdController::class, 'click'])->name('ads.click');

/*
|--------------------------------------------------------------------------
| Admin Routes
|--------------------------------------------------------------------------
*/

Route::middleware(['auth', 'admin'])->prefix('admin')->name('admin.')->group(function () {

    // Advertising
    Route::prefix('advertising')->name('advertising.')->group(function () {
        Route::resource('campaigns', AdCampaignController::class);
        Route::patch('campaigns/{campaign}/status', [AdCampaignController::class, 'updateStatus'])->name('campaigns.status')

        Route::resource('campaigns.creatives', CreativeController::class)->shallow();
        Route::resource('placements', PlacementController::class);
        Route::get('reports', [AdReportController::class, 'index'])->name('reports.index');
        Route::get('reports/export', [AdReportController::class, 'export'])->name('reports.export');
    });

    // Email
    Route::prefix('email')->name('email.')->group(function () {
        Route::resource('campaigns', EmailCampaignController::class)->only(['index', 'show']);
        Route::post('campaigns/generate-digest', [EmailCampaignController::class, 'generateDigest'])->name('campaigns.genera
        Route::post('campaigns/generate-newsletter', [EmailCampaignController::class, 'generateNewsletter'])->name('campaign
        Route::resource('subscribers', SubscriberController::class);
        Route::resource('templates', TemplateController::class);
    });

    // Emergency
```

```
    Route::prefix('emergency')->name('emergency.')->group(function () {
        Route::resource('alerts', AlertController::class);
        Route::post('alerts/{alert}/publish', [AlertController::class, 'publish'])->name('alerts.publish');
        Route::post('alerts/{alert}/cancel', [AlertController::class, 'cancel'])->name('alerts.cancel');
        Route::resource('subscriptions', EmergencySubController::class)->only(['index', 'show']);
        Route::resource('municipal', MunicipalController::class);
    });
});
```

---

# PHASE 7: SCHEDULED JOBS

php

```php
<?php
// app/Console/Kernel.php

namespace App\Console;

use Illuminate\Console\Scheduling\Schedule;
use Illuminate\Foundation\Console\Kernel as ConsoleKernel;

class Kernel extends ConsoleKernel
{
    protected function schedule(Schedule $schedule): void
    {
        // Generate daily digests at 2 AM for each timezone

        $schedule->command('email:generate-digests')
            ->dailyAt('02:00')
            ->withoutOverlapping();

        // Generate weekly newsletters on Saturday at 10 PM
        $schedule->command('email:generate-newsletters')
            ->weeklyOn(6, '22:00')
            ->withoutOverlapping();

        // Generate SMB reports on Sunday at 6 PM
        $schedule->command('email:generate-smb-reports')
            ->weeklyOn(0, '18:00')
            ->withoutOverlapping();

        // Process email send queue
        $schedule->command('email:process-queue')
```

```
        ->everyMinute()
        ->withoutOverlapping();

    // Update ad inventory metrics
    $schedule->command('ads:update-inventory')
        ->hourly();

    // Clean up expired ad inventory
    $schedule->command('ads:cleanup-expired')
        ->dailyAt('03:00');

    // Expire old emergency alerts
    $schedule->command('emergency:expire-alerts')
        ->everyFiveMinutes();

    // Update campaign performance stats
    $schedule->command('ads:update-stats')
        ->hourly();
  }

  protected function commands(): void
  {
    $this->load(__DIR__.'/Commands');
    require base_path('routes/console.php');
  }
}
```

php

```
<?php
// app/Console/Commands/GenerateDailyDigests.php

namespace App\Console\Commands;

use App\Models\Community;
use App\Services\EmailGeneratorService;
use Illuminate\Console\Command;

class GenerateDailyDigests extends Command
{
  protected $signature = 'email:generate-digests';
  protected $description = 'Generate daily digest emails for all communities';
```

```
    public function handle(EmailGeneratorService $service): int
    {
        $communities = Community::where('status', 'active')->get();

        $bar = $this->output->createProgressBar($communities->count());
        $bar->start();

        foreach ($communities as $community) {
            try {
                $service->generateDailyDigest($community);
                $this->info(" Generated digest for {$community->name}");
            } catch (\Exception $e) {
                $this->error(" Failed for {$community->name}: {$e->getMessage()}");
            }
            $bar->advance();
        }

        $bar->finish();
        $this->newLine();

        return Command::SUCCESS;
    }
}
```

---

# PHASE 8: QUEUE JOBS

php

```
<?php
// app/Jobs/SendEmergencyEmail.php

namespace App\Jobs;

use App\Models\EmergencyDelivery;
use App\Services\EmailDeliveryService;
use Illuminate\Bus\Queueable;
use Illuminate\Contracts\Queue\ShouldQueue;
use Illuminate\Foundation\Bus\Dispatchable;
use Illuminate\Queue\InteractsWithQueue;
use Illuminate\Queue\SerializesModels;

class SendEmergencyEmail implements ShouldQueue
```

```
class SendEmergencyEmail implements ShouldQueue
{
    use Dispatchable, InteractsWithQueue, Queueable, SerializesModels;

    public int $tries = 3;
    public int $backoff = 30;

    public function __construct(public EmergencyDelivery $delivery)
    {}

    public function handle(EmailDeliveryService $emailService): void
    {
        $alert = $this->delivery->alert;
        $subscriber = $this->delivery->subscription->subscriber;

        try {
            $messageId = $emailService->sendEmergencyAlert($subscriber, $alert);

            $this->delivery->update([
                'status' => 'sent',
                'external_id' => $messageId,
                'sent_at' => now(),
            ]);
        } catch (\Exception $e) {
            $this->delivery->update([
                'status' => 'failed',
                'error_message' => $e->getMessage(),
            ]);

            throw $e;
        }
    }
}
```

php

```
<?php
// app/Jobs/SendEmergencySms.php

namespace App\Jobs;

use App\Models\EmergencyDelivery;
use App\Services\SmsService;
use Illuminate\Bus\Queueable;
use Illuminate\Contracts\Queue\ShouldQueue;
```

```php
use Illuminate\Foundation\Bus\Dispatchable;
use Illuminate\Queue\InteractsWithQueue;
use Illuminate\Queue\SerializesModels;

class SendEmergencySms implements ShouldQueue
{
    use Dispatchable, InteractsWithQueue, Queueable, SerializesModels;

    public int $tries = 3;
    public int $backoff = 10;

    public function __construct(public EmergencyDelivery $delivery)
    {}

    public function handle(SmsService $smsService): void
    {
        $alert = $this->delivery->alert;
        $subscription = $this->delivery->subscription;

        if (!$subscription->canReceiveSms()) {
            $this->delivery->update([
                'status' => 'failed',
                'error_message' => 'SMS not enabled or verified',
            ]);
            return;
        }

        try {
            $sid = $smsService->sendEmergencyAlert(
                $subscription->phone_number,
                $alert

            );

            $this->delivery->update([
                'status' => 'sent',
                'external_id' => $sid,
                'sent_at' => now(),
            ]);
        } catch (\Exception $e) {
            $this->delivery->update([
                'status' => 'failed',
                'error_message' => $e->getMessage(),
            ]);
```