**📋 Début de la Discussion 6 : Tableau de bord opérateur**

**🎯 Objectif**

Créer les modèles Eloquent, services métier, contrôleurs et vues pour le tableau de bord opérateur.

**📂 Plan de travail pour cette discussion**

**Étape 1 : Modèles Eloquent** (où nous commençons)

* Organisation avec relations
* Dossier avec workflow
* Adherent avec validations
* Document et DocumentType
* Fondateur avec géolocalisation

**Étape 2 : Services métier**

* OrganisationValidationService
* WorkflowService
* AdherentImportService
* QrCodeService

**Étape 3 : Contrôleurs**

* OrganisationController (create, store)
* GuideController (affichage guide)
* DossierController (soumission)
* DocumentController (upload)

**Étape 4 : Vues**

* Guide de création
* Formulaire multi-étapes
* Upload documents
* Dashboard opérateur

**Étape 5 : Middleware**

* CheckOrganisationLimit
* DossierLock

**🚀 Commençons par l'Étape 1 : Création des modèles Eloquent**

Je vais créer les modèles principaux avec leurs relations. Commençons par le modèle Organisation :

# 📋 SYNTHÈSE COMPLÈTE - Discussion 6 : Tableau de bord opérateur

## 📊 Vue d'ensemble

**Objectif** : Implémenter les modèles Eloquent, services métier, contrôleurs et vues pour le tableau de bord opérateur

**Statut** : Étape 1 (Modèles Eloquent) complétée à 100%

**Travail réalisé** : 11 modèles créés/mis à jour avec toutes leurs relations et méthodes

## ✅ Modèles créés

* 1. Organisation (app/Models/Organisation.php)

<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;

use Illuminate\Database\Eloquent\Model;

use Illuminate\Database\Eloquent\Relations\BelongsTo;

use Illuminate\Database\Eloquent\Relations\HasMany;

use Illuminate\Database\Eloquent\Relations\HasOne;

class Organisation extends Model

{

use HasFactory;

protected $fillable = [

'user\_id',

'type',

'nom',

'sigle',

'objet',

'siege\_social',

'province',

'departement',

'canton',

'prefecture',

'sous\_prefecture',

'regroupement',

'zone\_type',

'ville\_commune',

'arrondissement',

'quartier',

'village',

'lieu\_dit',

'latitude',

'longitude',

'email',

'telephone',

'telephone\_secondaire',

'site\_web',

'numero\_recepisse',

'date\_creation',

'statut',

'is\_active',

'nombre\_adherents\_min',

'organes\_gestion'

];

protected $casts = [

'date\_creation' => 'date',

'is\_active' => 'boolean',

'organes\_gestion' => 'array',

'latitude' => 'decimal:8',

'longitude' => 'decimal:8'

];

// Constantes pour les types d'organisation

const TYPE\_ASSOCIATION = 'association';

const TYPE\_ONG = 'ong';

const TYPE\_PARTI = 'parti\_politique';

const TYPE\_CONFESSION = 'confession\_religieuse';

// Constantes pour les statuts

const STATUT\_BROUILLON = 'brouillon';

const STATUT\_SOUMIS = 'soumis';

const STATUT\_EN\_VALIDATION = 'en\_validation';

const STATUT\_APPROUVE = 'approuve';

const STATUT\_REJETE = 'rejete';

const STATUT\_SUSPENDU = 'suspendu';

const STATUT\_RADIE = 'radie';

// Constantes pour les zones

const ZONE\_URBAINE = 'urbaine';

const ZONE\_RURALE = 'rurale';

/\*\*

\* Relation avec l'utilisateur créateur (opérateur)

\*/

public function user(): BelongsTo

{

return $this->belongsTo(User::class);

}

/\*\*

\* Relation avec les dossiers

\*/

public function dossiers(): HasMany

{

return $this->hasMany(Dossier::class);

}

/\*\*

\* Relation avec le dossier actif

\*/

public function dossierActif(): HasOne

{

return $this->hasOne(Dossier::class)

->where('is\_active', true)

->latest();

}

/\*\*

\* Relation avec les fondateurs

\*/

public function fondateurs(): HasMany

{

return $this->hasMany(Fondateur::class);

}

/\*\*

\* Relation avec les adhérents

\*/

public function adherents(): HasMany

{

return $this->hasMany(Adherent::class);

}

/\*\*

\* Relation avec les adhérents actifs

\*/

public function adherentsActifs(): HasMany

{

return $this->hasMany(Adherent::class)

->where('is\_active', true);

}

/\*\*

\* Relation avec les établissements

\*/

public function etablissements(): HasMany

{

return $this->hasMany(Etablissement::class);

}

/\*\*

\* Relation avec les membres des organes

\*/

public function organeMembres(): HasMany

{

return $this->hasMany(OrganeMember::class);

}

/\*\*

\* Relation avec les déclarations

\*/

public function declarations(): HasMany

{

return $this->hasMany(Declaration::class);

}

/\*\*

\* Scopes

\*/

public function scopeActives($query)

{

return $query->where('is\_active', true);

}

public function scopeByType($query, $type)

{

return $query->where('type', $type);

}

public function scopeByStatut($query, $statut)

{

return $query->where('statut', $statut);

}

public function scopeApprouvees($query)

{

return $query->where('statut', self::STATUT\_APPROUVE);

}

/\*\*

\* Méthodes utilitaires

\*/

public function isPartiPolitique(): bool

{

return $this->type === self::TYPE\_PARTI;

}

public function isConfessionReligieuse(): bool

{

return $this->type === self::TYPE\_CONFESSION;

}

public function isAssociation(): bool

{

return $this->type === self::TYPE\_ASSOCIATION;

}

public function isOng(): bool

{

return $this->type === self::TYPE\_ONG;

}

public function isApprouvee(): bool

{

return $this->statut === self::STATUT\_APPROUVE;

}

public function canAddAdherent(): bool

{

// Pour les partis politiques et confessions religieuses,

// vérifier s'il n'y a pas déjà une organisation active

if (in\_array($this->type, [self::TYPE\_PARTI, self::TYPE\_CONFESSION])) {

return $this->is\_active && $this->isApprouvee();

}

return $this->isApprouvee();

}

public function hasMinimumAdherents(): bool

{

return $this->adherentsActifs()->count() >= $this->nombre\_adherents\_min;

}

/\*\*

\* Obtenir l'adresse complète formatée

\*/

public function getAdresseCompleteAttribute(): string

{

$parts = array\_filter([

$this->siege\_social,

$this->quartier,

$this->arrondissement,

$this->ville\_commune,

$this->village,

$this->sous\_prefecture,

$this->prefecture,

$this->departement,

$this->province

]);

return implode(', ', $parts);

}

/\*\*

\* Obtenir le label du type

\*/

public function getTypeLabelAttribute(): string

{

$labels = [

self::TYPE\_ASSOCIATION => 'Association',

self::TYPE\_ONG => 'ONG',

self::TYPE\_PARTI => 'Parti politique',

self::TYPE\_CONFESSION => 'Confession religieuse'

];

return $labels[$this->type] ?? $this->type;

}

/\*\*

\* Obtenir le label du statut

\*/

public function getStatutLabelAttribute(): string

{

$labels = [

self::STATUT\_BROUILLON => 'Brouillon',

self::STATUT\_SOUMIS => 'Soumis',

self::STATUT\_EN\_VALIDATION => 'En validation',

self::STATUT\_APPROUVE => 'Approuvé',

self::STATUT\_REJETE => 'Rejeté',

self::STATUT\_SUSPENDU => 'Suspendu',

self::STATUT\_RADIE => 'Radié'

];

return $labels[$this->statut] ?? $this->statut;

}

/\*\*

\* Obtenir la couleur du statut pour l'affichage

\*/

public function getStatutColorAttribute(): string

{

$colors = [

self::STATUT\_BROUILLON => 'secondary',

self::STATUT\_SOUMIS => 'info',

self::STATUT\_EN\_VALIDATION => 'warning',

self::STATUT\_APPROUVE => 'success',

self::STATUT\_REJETE => 'danger',

self::STATUT\_SUSPENDU => 'dark',

self::STATUT\_RADIE => 'danger'

];

return $colors[$this->statut] ?? 'secondary';

}

}

* 1. Dossier (app/Models/Dossier.php)

<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;

use Illuminate\Database\Eloquent\Model;

use Illuminate\Database\Eloquent\Relations\BelongsTo;

use Illuminate\Database\Eloquent\Relations\HasMany;

use Illuminate\Database\Eloquent\Relations\HasOne;

class Dossier extends Model

{

use HasFactory;

protected $fillable = [

'organisation\_id',

'type\_operation',

'numero\_dossier',

'statut',

'date\_soumission',

'date\_traitement',

'motif\_rejet',

'current\_step\_id',

'is\_active',

'metadata'

];

protected $casts = [

'date\_soumission' => 'datetime',

'date\_traitement' => 'datetime',

'is\_active' => 'boolean',

'metadata' => 'array'

];

// Constantes pour les types d'opération

const TYPE\_CREATION = 'creation';

const TYPE\_MODIFICATION = 'modification';

const TYPE\_CESSATION = 'cessation';

const TYPE\_DECLARATION = 'declaration';

const TYPE\_FUSION = 'fusion';

const TYPE\_ABSORPTION = 'absorption';

// Constantes pour les statuts

const STATUT\_BROUILLON = 'brouillon';

const STATUT\_SOUMIS = 'soumis';

const STATUT\_EN\_COURS = 'en\_cours';

const STATUT\_ACCEPTE = 'accepte';

const STATUT\_REJETE = 'rejete';

const STATUT\_ARCHIVE = 'archive';

/\*\*

\* Boot method pour générer automatiquement le numéro de dossier

\*/

protected static function boot()

{

parent::boot();

static::creating(function ($dossier) {

if (empty($dossier->numero\_dossier)) {

$dossier->numero\_dossier = self::generateNumeroDossier($dossier->type\_operation);

}

});

}

/\*\*

\* Générer un numéro de dossier unique

\*/

public static function generateNumeroDossier($typeOperation): string

{

// Version compatible PHP 7.4

switch($typeOperation) {

case self::TYPE\_CREATION:

$prefix = 'CRE';

break;

case self::TYPE\_MODIFICATION:

$prefix = 'MOD';

break;

case self::TYPE\_CESSATION:

$prefix = 'CES';

break;

case self::TYPE\_DECLARATION:

$prefix = 'DEC';

break;

case self::TYPE\_FUSION:

$prefix = 'FUS';

break;

case self::TYPE\_ABSORPTION:

$prefix = 'ABS';

break;

default:

$prefix = 'DOS';

break;

}

$year = date('Y');

$lastDossier = self::where('numero\_dossier', 'like', $prefix . '-' . $year . '-%')

->orderBy('numero\_dossier', 'desc')

->first();

if ($lastDossier) {

$lastNumber = intval(substr($lastDossier->numero\_dossier, -6));

$newNumber = $lastNumber + 1;

} else {

$newNumber = 1;

}

return sprintf('%s-%s-%06d', $prefix, $year, $newNumber);

}

/\*\*

\* Relations

\*/

public function organisation(): BelongsTo

{

return $this->belongsTo(Organisation::class);

}

public function currentStep(): BelongsTo

{

return $this->belongsTo(WorkflowStep::class, 'current\_step\_id');

}

public function documents(): HasMany

{

return $this->hasMany(Document::class);

}

public function validations(): HasMany

{

return $this->hasMany(DossierValidation::class);

}

public function operations(): HasMany

{

return $this->hasMany(DossierOperation::class);

}

public function comments(): HasMany

{

return $this->hasMany(DossierComment::class);

}

public function lock(): HasOne

{

return $this->hasOne(DossierLock::class);

}

/\*\*

\* Scopes

\*/

public function scopeActifs($query)

{

return $query->where('is\_active', true);

}

public function scopeByStatut($query, $statut)

{

return $query->where('statut', $statut);

}

public function scopeSoumis($query)

{

return $query->where('statut', self::STATUT\_SOUMIS);

}

public function scopeEnCours($query)

{

return $query->where('statut', self::STATUT\_EN\_COURS);

}

public function scopeNonTraites($query)

{

return $query->whereIn('statut', [self::STATUT\_SOUMIS, self::STATUT\_EN\_COURS]);

}

/\*\*

\* Vérifier si le dossier est verrouillé

\*/

public function isLocked(): bool

{

return $this->lock()->where('is\_active', true)->exists();

}

/\*\*

\* Vérifier si le dossier est verrouillé par un utilisateur spécifique

\*/

public function isLockedBy($userId): bool

{

return $this->lock()

->where('is\_active', true)

->where('user\_id', $userId)

->exists();

}

/\*\*

\* Obtenir l'utilisateur qui a verrouillé le dossier

\*/

public function getLockedByUser()

{

$lock = $this->lock()->where('is\_active', true)->first();

return $lock ? $lock->user : null;

}

/\*\*

\* Vérifier si le dossier peut être modifié

\*/

public function canBeModified(): bool

{

return in\_array($this->statut, [self::STATUT\_BROUILLON, self::STATUT\_REJETE]);

}

/\*\*

\* Vérifier si le dossier peut être soumis

\*/

public function canBeSubmitted(): bool

{

return $this->statut === self::STATUT\_BROUILLON;

}

/\*\*

\* Vérifier si tous les documents obligatoires sont fournis

\*/

public function hasAllRequiredDocuments(): bool

{

$requiredTypes = DocumentType::where('type\_organisation', $this->organisation->type)

->where('is\_active', true)

->where('is\_obligatoire', true)

->pluck('id');

$providedTypes = $this->documents()->pluck('document\_type\_id');

return $requiredTypes->diff($providedTypes)->isEmpty();

}

/\*\*

\* Obtenir les documents manquants

\*/

public function getMissingDocuments()

{

$requiredTypes = DocumentType::where('type\_organisation', $this->organisation->type)

->where('is\_active', true)

->where('is\_obligatoire', true)

->pluck('id');

$providedTypes = $this->documents()->pluck('document\_type\_id');

$missingIds = $requiredTypes->diff($providedTypes);

return DocumentType::whereIn('id', $missingIds)->get();

}

/\*\*

\* Obtenir la prochaine étape du workflow

\*/

public function getNextStep()

{

if (!$this->current\_step\_id) {

return WorkflowStep::where('type\_organisation', $this->organisation->type)

->where('type\_operation', $this->type\_operation)

->where('is\_active', true)

->orderBy('ordre')

->first();

}

return WorkflowStep::where('type\_organisation', $this->organisation->type)

->where('type\_operation', $this->type\_operation)

->where('is\_active', true)

->where('ordre', '>', $this->currentStep->ordre)

->orderBy('ordre')

->first();

}

/\*\*

\* Obtenir l'étape précédente du workflow (pour retour en cas de rejet)

\*/

public function getPreviousStep()

{

if (!$this->current\_step\_id) {

return null;

}

return WorkflowStep::where('type\_organisation', $this->organisation->type)

->where('type\_operation', $this->type\_operation)

->where('is\_active', true)

->where('ordre', '<', $this->currentStep->ordre)

->orderBy('ordre', 'desc')

->first();

}

/\*\*

\* Obtenir le dernier rejet

\*/

public function getLastRejection()

{

return $this->validations()

->where('decision', 'rejete')

->latest()

->first();

}

/\*\*

\* Obtenir le pourcentage de progression

\*/

public function getProgressionPercentage(): int

{

if (!$this->current\_step\_id) {

return 0;

}

$totalSteps = WorkflowStep::where('type\_organisation', $this->organisation->type)

->where('type\_operation', $this->type\_operation)

->where('is\_active', true)

->count();

if ($totalSteps === 0) {

return 0;

}

$currentStepOrder = $this->currentStep->ordre;

return round(($currentStepOrder / $totalSteps) \* 100);

}

/\*\*

\* Accesseurs

\*/

public function getTypeOperationLabelAttribute(): string

{

$labels = [

self::TYPE\_CREATION => 'Création',

self::TYPE\_MODIFICATION => 'Modification',

self::TYPE\_CESSATION => 'Cessation',

self::TYPE\_DECLARATION => 'Déclaration',

self::TYPE\_FUSION => 'Fusion',

self::TYPE\_ABSORPTION => 'Absorption'

];

return $labels[$this->type\_operation] ?? $this->type\_operation;

}

public function getStatutLabelAttribute(): string

{

$labels = [

self::STATUT\_BROUILLON => 'Brouillon',

self::STATUT\_SOUMIS => 'Soumis',

self::STATUT\_EN\_COURS => 'En cours',

self::STATUT\_ACCEPTE => 'Accepté',

self::STATUT\_REJETE => 'Rejeté',

self::STATUT\_ARCHIVE => 'Archivé'

];

return $labels[$this->statut] ?? $this->statut;

}

public function getStatutColorAttribute(): string

{

$colors = [

self::STATUT\_BROUILLON => 'secondary',

self::STATUT\_SOUMIS => 'info',

self::STATUT\_EN\_COURS => 'warning',

self::STATUT\_ACCEPTE => 'success',

self::STATUT\_REJETE => 'danger',

self::STATUT\_ARCHIVE => 'dark'

];

return $colors[$this->statut] ?? 'secondary';

}

}

3. Adherent (app/Models/Adherent.php)

<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;

use Illuminate\Database\Eloquent\Model;

use Illuminate\Database\Eloquent\Relations\BelongsTo;

use Illuminate\Database\Eloquent\Relations\HasMany;

use Illuminate\Database\Eloquent\Relations\HasOne;

class Adherent extends Model

{

use HasFactory;

protected $fillable = [

'organisation\_id',

'nip',

'nom',

'prenom',

'date\_naissance',

'lieu\_naissance',

'sexe',

'nationalite',

'profession',

'adresse',

'province',

'departement',

'canton',

'prefecture',

'sous\_prefecture',

'telephone',

'email',

'photo\_path',

'piece\_identite\_path',

'date\_adhesion',

'numero\_carte',

'is\_fondateur',

'is\_active',

'metadata'

];

protected $casts = [

'date\_naissance' => 'date',

'date\_adhesion' => 'date',

'is\_fondateur' => 'boolean',

'is\_active' => 'boolean',

'metadata' => 'array'

];

// Constantes

const SEXE\_MASCULIN = 'M';

const SEXE\_FEMININ = 'F';

/\*\*

\* Boot method pour gérer les événements

\*/

protected static function boot()

{

parent::boot();

static::creating(function ($adherent) {

// Générer le numéro de carte si non fourni

if (empty($adherent->numero\_carte)) {

$adherent->numero\_carte = self::generateNumeroCarte($adherent->organisation\_id);

}

// Vérifier l'unicité pour les partis politiques

if ($adherent->organisation->isPartiPolitique()) {

self::verifyUnicityForPartiPolitique($adherent);

}

});

static::created(function ($adherent) {

// Créer l'historique d'adhésion

AdherentHistory::create([

'adherent\_id' => $adherent->id,

'organisation\_id' => $adherent->organisation\_id,

'type\_mouvement' => 'adhesion',

'date\_mouvement' => $adherent->date\_adhesion ?? now(),

'motif' => 'Adhésion initiale'

]);

});

}

/\*\*

\* Générer un numéro de carte unique

\*/

public static function generateNumeroCarte($organisationId): string

{

$organisation = Organisation::find($organisationId);

$prefix = strtoupper(substr($organisation->sigle ?? $organisation->nom, 0, 3));

$year = date('Y');

$lastAdherent = self::where('organisation\_id', $organisationId)

->where('numero\_carte', 'like', $prefix . '-' . $year . '-%')

->orderBy('numero\_carte', 'desc')

->first();

if ($lastAdherent) {

$lastNumber = intval(substr($lastAdherent->numero\_carte, -6));

$newNumber = $lastNumber + 1;

} else {

$newNumber = 1;

}

return sprintf('%s-%s-%06d', $prefix, $year, $newNumber);

}

/\*\*

\* Vérifier l'unicité pour les partis politiques

\*/

protected static function verifyUnicityForPartiPolitique($adherent)

{

$existingAdhesion = self::join('organisations', 'adherents.organisation\_id', '=', 'organisations.id')

->where('adherents.nip', $adherent->nip)

->where('adherents.is\_active', true)

->where('organisations.type', Organisation::TYPE\_PARTI)

->where('adherents.organisation\_id', '!=', $adherent->organisation\_id)

->first();

if ($existingAdhesion) {

throw new \Exception(

"Cette personne (NIP: {$adherent->nip}) est déjà membre du parti politique '{$existingAdhesion->organisation->nom}'. " .

"Une exclusion formelle est requise avant de pouvoir adhérer à un autre parti."

);

}

}

/\*\*

\* Relations

\*/

public function organisation(): BelongsTo

{

return $this->belongsTo(Organisation::class);

}

public function histories(): HasMany

{

return $this->hasMany(AdherentHistory::class);

}

public function exclusion(): HasOne

{

return $this->hasOne(AdherentExclusion::class)->latest();

}

public function imports(): HasMany

{

return $this->hasMany(AdherentImport::class);

}

/\*\*

\* Scopes

\*/

public function scopeActifs($query)

{

return $query->where('is\_active', true);

}

public function scopeInactifs($query)

{

return $query->where('is\_active', false);

}

public function scopeFondateurs($query)

{

return $query->where('is\_fondateur', true);

}

public function scopeByNip($query, $nip)

{

return $query->where('nip', $nip);

}

public function scopeHommes($query)

{

return $query->where('sexe', self::SEXE\_MASCULIN);

}

public function scopeFemmes($query)

{

return $query->where('sexe', self::SEXE\_FEMININ);

}

/\*\*

\* Méthodes utilitaires

\*/

public function isExcluded(): bool

{

return $this->exclusion()->exists();

}

public function canBeTransferred(): bool

{

// Un adhérent peut être transféré s'il n'est pas fondateur et est actif

return !$this->is\_fondateur && $this->is\_active;

}

public function getAge(): int

{

return $this->date\_naissance ? $this->date\_naissance->age : 0;

}

public function getNomCompletAttribute(): string

{

return trim($this->nom . ' ' . $this->prenom);

}

public function getSexeLabelAttribute(): string

{

return $this->sexe === self::SEXE\_MASCULIN ? 'Masculin' : 'Féminin';

}

/\*\*

\* Exclure l'adhérent

\*/

public function exclude($motif, $dateExclusion = null, $documentPath = null)

{

// Marquer l'adhérent comme inactif

$this->update(['is\_active' => false]);

// Créer l'enregistrement d'exclusion

AdherentExclusion::create([

'adherent\_id' => $this->id,

'organisation\_id' => $this->organisation\_id,

'date\_exclusion' => $dateExclusion ?? now(),

'motif' => $motif,

'document\_path' => $documentPath

]);

// Créer l'historique

AdherentHistory::create([

'adherent\_id' => $this->id,

'organisation\_id' => $this->organisation\_id,

'type\_mouvement' => 'exclusion',

'date\_mouvement' => $dateExclusion ?? now(),

'motif' => $motif

]);

return true;

}

/\*\*

\* Réactiver l'adhérent

\*/

public function reactivate($motif = 'Réactivation')

{

$this->update(['is\_active' => true]);

AdherentHistory::create([

'adherent\_id' => $this->id,

'organisation\_id' => $this->organisation\_id,

'type\_mouvement' => 'reactivation',

'date\_mouvement' => now(),

'motif' => $motif

]);

return true;

}

/\*\*

\* Transférer vers une autre organisation

\*/

public function transferTo($newOrganisationId, $motif = 'Transfert', $documentPath = null)

{

$oldOrganisationId = $this->organisation\_id;

// Si c'est un parti politique, vérifier l'unicité

$newOrganisation = Organisation::find($newOrganisationId);

if ($newOrganisation->isPartiPolitique()) {

// Vérifier qu'il n'est pas déjà dans un autre parti

$existingInParti = self::join('organisations', 'adherents.organisation\_id', '=', 'organisations.id')

->where('adherents.nip', $this->nip)

->where('adherents.is\_active', true)

->where('organisations.type', Organisation::TYPE\_PARTI)

->where('adherents.id', '!=', $this->id)

->exists();

if ($existingInParti) {

throw new \Exception("Cette personne est déjà membre d'un parti politique actif.");

}

}

// Effectuer le transfert

$this->update([

'organisation\_id' => $newOrganisationId,

'date\_adhesion' => now(),

'numero\_carte' => self::generateNumeroCarte($newOrganisationId)

]);

// Créer l'historique de sortie

AdherentHistory::create([

'adherent\_id' => $this->id,

'organisation\_id' => $oldOrganisationId,

'type\_mouvement' => 'sortie',

'date\_mouvement' => now(),

'motif' => $motif,

'organisation\_destination\_id' => $newOrganisationId,

'document\_path' => $documentPath

]);

// Créer l'historique d'entrée

AdherentHistory::create([

'adherent\_id' => $this->id,

'organisation\_id' => $newOrganisationId,

'type\_mouvement' => 'entree',

'date\_mouvement' => now(),

'motif' => $motif,

'organisation\_source\_id' => $oldOrganisationId

]);

return true;

}

/\*\*

\* Vérifier si l'adhérent peut adhérer à une organisation

\*/

public static function canJoinOrganisation($nip, $organisationId): array

{

$organisation = Organisation::find($organisationId);

if (!$organisation) {

return ['can\_join' => false, 'reason' => 'Organisation non trouvée'];

}

// Pour les partis politiques

if ($organisation->isPartiPolitique()) {

$existingInParti = self::join('organisations', 'adherents.organisation\_id', '=', 'organisations.id')

->where('adherents.nip', $nip)

->where('adherents.is\_active', true)

->where('organisations.type', Organisation::TYPE\_PARTI)

->first();

if ($existingInParti) {

return [

'can\_join' => false,

'reason' => "Déjà membre du parti politique '{$existingInParti->organisation->nom}'",

'existing\_organisation' => $existingInParti->organisation

];

}

}

// Pour les confessions religieuses (si restriction similaire)

if ($organisation->isConfessionReligieuse()) {

$existingInConfession = self::join('organisations', 'adherents.organisation\_id', '=', 'organisations.id')

->where('adherents.nip', $nip)

->where('adherents.is\_active', true)

->where('organisations.type', Organisation::TYPE\_CONFESSION)

->first();

if ($existingInConfession) {

return [

'can\_join' => false,

'reason' => "Déjà membre de la confession religieuse '{$existingInConfession->organisation->nom}'",

'existing\_organisation' => $existingInConfession->organisation

];

}

}

return ['can\_join' => true];

}

/\*\*

\* Obtenir toutes les organisations de l'adhérent

\*/

public function getAllOrganisations()

{

return Organisation::whereHas('adherents', function ($query) {

$query->where('nip', $this->nip);

})->get();

}

/\*\*

\* Obtenir les organisations actives

\*/

public function getActiveOrganisations()

{

return Organisation::whereHas('adherents', function ($query) {

$query->where('nip', $this->nip)

->where('is\_active', true);

})->get();

}

}

4. Document (app/Models/Document.php)

<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;

use Illuminate\Database\Eloquent\Model;

use Illuminate\Database\Eloquent\Relations\BelongsTo;

use Illuminate\Support\Facades\Storage;

use Illuminate\Support\Str;

class Document extends Model

{

use HasFactory;

protected $fillable = [

'dossier\_id',

'document\_type\_id',

'nom\_fichier',

'nom\_original',

'chemin\_fichier',

'taille',

'mime\_type',

'hash\_fichier',

'metadata',

'uploaded\_by',

'validated\_by',

'validated\_at',

'is\_validated',

'validation\_comment'

];

protected $casts = [

'taille' => 'integer',

'is\_validated' => 'boolean',

'validated\_at' => 'datetime',

'metadata' => 'array'

];

// Constantes pour les statuts

const STATUS\_PENDING = 'pending';

const STATUS\_VALIDATED = 'validated';

const STATUS\_REJECTED = 'rejected';

// Taille maximale par défaut (10 MB)

const MAX\_FILE\_SIZE = 10485760;

// Extensions autorisées

const ALLOWED\_EXTENSIONS = ['pdf', 'jpg', 'jpeg', 'png', 'doc', 'docx'];

// Types MIME autorisés

const ALLOWED\_MIMES = [

'application/pdf',

'image/jpeg',

'image/png',

'application/msword',

'application/vnd.openxmlformats-officedocument.wordprocessingml.document'

];

/\*\*

\* Boot method

\*/

protected static function boot()

{

parent::boot();

static::creating(function ($document) {

// Générer un nom unique pour le fichier

if (empty($document->nom\_fichier)) {

$extension = pathinfo($document->nom\_original, PATHINFO\_EXTENSION);

$document->nom\_fichier = Str::uuid() . '.' . $extension;

}

// Calculer le hash du fichier si non fourni

if (empty($document->hash\_fichier) && $document->chemin\_fichier) {

$fullPath = Storage::path($document->chemin\_fichier);

if (file\_exists($fullPath)) {

$document->hash\_fichier = hash\_file('sha256', $fullPath);

}

}

});

static::deleting(function ($document) {

// Supprimer le fichier physique lors de la suppression de l'enregistrement

if ($document->chemin\_fichier && Storage::exists($document->chemin\_fichier)) {

Storage::delete($document->chemin\_fichier);

}

});

}

/\*\*

\* Relations

\*/

public function dossier(): BelongsTo

{

return $this->belongsTo(Dossier::class);

}

public function documentType(): BelongsTo

{

return $this->belongsTo(DocumentType::class);

}

public function uploadedBy(): BelongsTo

{

return $this->belongsTo(User::class, 'uploaded\_by');

}

public function validatedBy(): BelongsTo

{

return $this->belongsTo(User::class, 'validated\_by');

}

/\*\*

\* Scopes

\*/

public function scopeValidated($query)

{

return $query->where('is\_validated', true);

}

public function scopePending($query)

{

return $query->whereNull('is\_validated');

}

public function scopeRejected($query)

{

return $query->where('is\_validated', false)->whereNotNull('validated\_at');

}

/\*\*

\* Accesseurs

\*/

public function getTailleLisibleAttribute(): string

{

$bytes = $this->taille;

$units = ['B', 'KB', 'MB', 'GB'];

for ($i = 0; $bytes > 1024 && $i < count($units) - 1; $i++) {

$bytes /= 1024;

}

return round($bytes, 2) . ' ' . $units[$i];

}

public function getExtensionAttribute(): string

{

return strtolower(pathinfo($this->nom\_original, PATHINFO\_EXTENSION));

}

public function getIsImageAttribute(): bool

{

return in\_array($this->mime\_type, ['image/jpeg', 'image/png', 'image/gif']);

}

public function getIsPdfAttribute(): bool

{

return $this->mime\_type === 'application/pdf';

}

public function getStatusAttribute(): string

{

if ($this->is\_validated === null) {

return self::STATUS\_PENDING;

}

return $this->is\_validated ? self::STATUS\_VALIDATED : self::STATUS\_REJECTED;

}

public function getStatusLabelAttribute(): string

{

$labels = [

self::STATUS\_PENDING => 'En attente',

self::STATUS\_VALIDATED => 'Validé',

self::STATUS\_REJECTED => 'Rejeté'

];

return $labels[$this->status] ?? 'Inconnu';

}

public function getStatusColorAttribute(): string

{

$colors = [

self::STATUS\_PENDING => 'warning',

self::STATUS\_VALIDATED => 'success',

self::STATUS\_REJECTED => 'danger'

];

return $colors[$this->status] ?? 'secondary';

}

/\*\*

\* Méthodes utilitaires

\*/

public function validate($userId, $comment = null): bool

{

$this->update([

'is\_validated' => true,

'validated\_by' => $userId,

'validated\_at' => now(),

'validation\_comment' => $comment

]);

return true;

}

public function reject($userId, $comment): bool

{

if (empty($comment)) {

throw new \Exception('Un commentaire est obligatoire pour rejeter un document');

}

$this->update([

'is\_validated' => false,

'validated\_by' => $userId,

'validated\_at' => now(),

'validation\_comment' => $comment

]);

return true;

}

/\*\*

\* Obtenir l'URL de téléchargement

\*/

public function getDownloadUrl(): string

{

return route('operator.documents.download', $this->id);

}

/\*\*

\* Obtenir l'URL de prévisualisation

\*/

public function getPreviewUrl(): ?string

{

if ($this->is\_image || $this->is\_pdf) {

return route('operator.documents.preview', $this->id);

}

return null;

}

/\*\*

\* Vérifier si le fichier existe physiquement

\*/

public function fileExists(): bool

{

return $this->chemin\_fichier && Storage::exists($this->chemin\_fichier);

}

/\*\*

\* Obtenir le chemin complet du fichier

\*/

public function getFullPath(): ?string

{

if (!$this->chemin\_fichier) {

return null;

}

return Storage::path($this->chemin\_fichier);

}

/\*\*

\* Dupliquer le document pour un autre dossier

\*/

public function duplicateFor($dossierId): Document

{

// Créer une copie du fichier

$newPath = str\_replace(

$this->dossier\_id,

$dossierId,

$this->chemin\_fichier

);

Storage::copy($this->chemin\_fichier, $newPath);

// Créer un nouvel enregistrement

return self::create([

'dossier\_id' => $dossierId,

'document\_type\_id' => $this->document\_type\_id,

'nom\_fichier' => $this->nom\_fichier,

'nom\_original' => $this->nom\_original,

'chemin\_fichier' => $newPath,

'taille' => $this->taille,

'mime\_type' => $this->mime\_type,

'hash\_fichier' => $this->hash\_fichier,

'metadata' => $this->metadata,

'uploaded\_by' => auth()->id()

]);

}

/\*\*

\* Vérifier si le document est un doublon

\*/

public function isDuplicate(): bool

{

return self::where('hash\_fichier', $this->hash\_fichier)

->where('id', '!=', $this->id)

->where('dossier\_id', $this->dossier\_id)

->exists();

}

/\*\*

\* Vérifier si l'extension est autorisée

\*/

public static function isAllowedExtension($extension): bool

{

return in\_array(strtolower($extension), self::ALLOWED\_EXTENSIONS);

}

/\*\*

\* Vérifier si le type MIME est autorisé

\*/

public static function isAllowedMimeType($mimeType): bool

{

return in\_array($mimeType, self::ALLOWED\_MIMES);

}

/\*\*

\* Obtenir le répertoire de stockage pour un dossier

\*/

public static function getStoragePath($dossierId, $documentTypeId): string

{

return "documents/{$dossierId}/{$documentTypeId}";

}

/\*\*

\* Générer un nom de fichier unique

\*/

public static function generateFileName($originalName): string

{

$extension = pathinfo($originalName, PATHINFO\_EXTENSION);

return Str::uuid() . '.' . strtolower($extension);

}

/\*\*

\* Nettoyer le nom de fichier original

\*/

public static function sanitizeFileName($fileName): string

{

// Remplacer les caractères spéciaux

$fileName = preg\_replace('/[^a-zA-Z0-9.\_-]/', '\_', $fileName);

// Limiter la longueur

if (strlen($fileName) > 100) {

$extension = pathinfo($fileName, PATHINFO\_EXTENSION);

$name = pathinfo($fileName, PATHINFO\_FILENAME);

$fileName = substr($name, 0, 90) . '.' . $extension;

}

return $fileName;

}

}

5. DocumentType (app/Models/DocumentType.php)

<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;

use Illuminate\Database\Eloquent\Model;

use Illuminate\Database\Eloquent\Relations\HasMany;

use Illuminate\Database\Eloquent\Relations\HasOne;

class DocumentType extends Model

{

use HasFactory;

protected $fillable = [

'code',

'nom',

'description',

'type\_organisation',

'type\_operation',

'is\_active',

'is\_obligatoire',

'ordre',

'taille\_max',

'extensions\_autorisees',

'instructions',

'exemple\_path'

];

protected $casts = [

'is\_active' => 'boolean',

'is\_obligatoire' => 'boolean',

'ordre' => 'integer',

'taille\_max' => 'integer',

'extensions\_autorisees' => 'array'

];

// Taille maximale par défaut (5 MB)

const DEFAULT\_MAX\_SIZE = 5242880;

// Extensions par défaut

const DEFAULT\_EXTENSIONS = ['pdf', 'jpg', 'jpeg', 'png'];

/\*\*

\* Boot method

\*/

protected static function boot()

{

parent::boot();

static::creating(function ($documentType) {

// Générer un code unique si non fourni

if (empty($documentType->code)) {

$documentType->code = self::generateCode($documentType->nom, $documentType->type\_organisation);

}

// Définir la taille maximale par défaut

if (empty($documentType->taille\_max)) {

$documentType->taille\_max = self::DEFAULT\_MAX\_SIZE;

}

// Définir les extensions par défaut

if (empty($documentType->extensions\_autorisees)) {

$documentType->extensions\_autorisees = self::DEFAULT\_EXTENSIONS;

}

// Définir l'ordre si non fourni

if (empty($documentType->ordre)) {

$maxOrdre = self::where('type\_organisation', $documentType->type\_organisation)

->where('type\_operation', $documentType->type\_operation)

->max('ordre');

$documentType->ordre = ($maxOrdre ?? 0) + 1;

}

});

}

/\*\*

\* Générer un code unique

\*/

public static function generateCode($nom, $typeOrganisation): string

{

// Préfixe selon le type d'organisation

$prefixes = [

Organisation::TYPE\_ASSOCIATION => 'ASS',

Organisation::TYPE\_ONG => 'ONG',

Organisation::TYPE\_PARTI => 'PP',

Organisation::TYPE\_CONFESSION => 'CR'

];

$prefix = $prefixes[$typeOrganisation] ?? 'DOC';

// Créer un code à partir du nom

$namePart = strtoupper(substr(preg\_replace('/[^a-zA-Z]/', '', $nom), 0, 10));

// Ajouter un numéro si nécessaire pour l'unicité

$baseCode = $prefix . '\_' . $namePart;

$code = $baseCode;

$counter = 1;

while (self::where('code', $code)->exists()) {

$code = $baseCode . '\_' . $counter;

$counter++;

}

return $code;

}

/\*\*

\* Relations

\*/

public function documents(): HasMany

{

return $this->hasMany(Document::class);

}

public function template(): HasOne

{

return $this->hasOne(DocumentTemplate::class);

}

/\*\*

\* Scopes

\*/

public function scopeActifs($query)

{

return $query->where('is\_active', true);

}

public function scopeObligatoires($query)

{

return $query->where('is\_obligatoire', true);

}

public function scopeFacultatifs($query)

{

return $query->where('is\_obligatoire', false);

}

public function scopeForOrganisationType($query, $type)

{

return $query->where('type\_organisation', $type);

}

public function scopeForOperation($query, $operation)

{

return $query->where('type\_operation', $operation);

}

public function scopeOrdered($query)

{

return $query->orderBy('ordre');

}

/\*\*

\* Obtenir les types de documents pour une organisation et une opération

\*/

public static function getForOrganisationAndOperation($typeOrganisation, $typeOperation)

{

return self::actifs()

->where('type\_organisation', $typeOrganisation)

->where(function ($query) use ($typeOperation) {

$query->where('type\_operation', $typeOperation)

->orWhereNull('type\_operation');

})

->ordered()

->get();

}

/\*\*

\* Obtenir les types de documents obligatoires

\*/

public static function getRequiredFor($typeOrganisation, $typeOperation)

{

return self::getForOrganisationAndOperation($typeOrganisation, $typeOperation)

->where('is\_obligatoire', true);

}

/\*\*

\* Accesseurs

\*/

public function getTailleMaxLisibleAttribute(): string

{

$bytes = $this->taille\_max;

$units = ['B', 'KB', 'MB', 'GB'];

for ($i = 0; $bytes > 1024 && $i < count($units) - 1; $i++) {

$bytes /= 1024;

}

return round($bytes, 2) . ' ' . $units[$i];

}

public function getExtensionsStringAttribute(): string

{

return implode(', ', $this->extensions\_autorisees ?? []);

}

public function getStatutLabelAttribute(): string

{

return $this->is\_active ? 'Actif' : 'Inactif';

}

public function getStatutColorAttribute(): string

{

return $this->is\_active ? 'success' : 'danger';

}

public function getObligatoireLabelAttribute(): string

{

return $this->is\_obligatoire ? 'Obligatoire' : 'Facultatif';

}

public function getObligatoireColorAttribute(): string

{

return $this->is\_obligatoire ? 'danger' : 'info';

}

/\*\*

\* Méthodes utilitaires

\*/

public function hasTemplate(): bool

{

return $this->template()->exists();

}

public function hasExample(): bool

{

return !empty($this->exemple\_path) && file\_exists(storage\_path('app/public/' . $this->exemple\_path));

}

public function getExampleUrl(): ?string

{

if ($this->hasExample()) {

return asset('storage/' . $this->exemple\_path);

}

return null;

}

/\*\*

\* Vérifier si une extension est autorisée

\*/

public function isExtensionAllowed($extension): bool

{

return in\_array(strtolower($extension), $this->extensions\_autorisees ?? []);

}

/\*\*

\* Vérifier si une taille de fichier est acceptable

\*/

public function isSizeAllowed($size): bool

{

return $size <= $this->taille\_max;

}

/\*\*

\* Activer/Désactiver le type de document

\*/

public function toggleActive(): bool

{

$this->is\_active = !$this->is\_active;

return $this->save();

}

/\*\*

\* Rendre obligatoire/facultatif

\*/

public function toggleObligatoire(): bool

{

$this->is\_obligatoire = !$this->is\_obligatoire;

return $this->save();

}

/\*\*

\* Réordonner les types de documents

\*/

public static function reorder(array $orderedIds): void

{

foreach ($orderedIds as $ordre => $id) {

self::where('id', $id)->update(['ordre' => $ordre + 1]);

}

}

/\*\*

\* Dupliquer pour un autre type d'organisation

\*/

public function duplicateFor($typeOrganisation): DocumentType

{

$newDocumentType = $this->replicate();

$newDocumentType->type\_organisation = $typeOrganisation;

$newDocumentType->code = self::generateCode($this->nom, $typeOrganisation);

$newDocumentType->save();

// Dupliquer le template s'il existe

if ($this->hasTemplate()) {

$template = $this->template;

$newTemplate = $template->replicate();

$newTemplate->document\_type\_id = $newDocumentType->id;

$newTemplate->save();

}

return $newDocumentType;

}

/\*\*

\* Obtenir les statistiques d'utilisation

\*/

public function getStatistics(): array

{

$totalDocuments = $this->documents()->count();

$validatedDocuments = $this->documents()->validated()->count();

$rejectedDocuments = $this->documents()->rejected()->count();

$pendingDocuments = $this->documents()->pending()->count();

return [

'total' => $totalDocuments,

'validated' => $validatedDocuments,

'rejected' => $rejectedDocuments,

'pending' => $pendingDocuments,

'validation\_rate' => $totalDocuments > 0

? round(($validatedDocuments / $totalDocuments) \* 100, 2)

: 0

];

}

/\*\*

\* Obtenir la liste des types d'organisation comme options

\*/

public static function getTypeOrganisationOptions(): array

{

return [

Organisation::TYPE\_ASSOCIATION => 'Association',

Organisation::TYPE\_ONG => 'ONG',

Organisation::TYPE\_PARTI => 'Parti politique',

Organisation::TYPE\_CONFESSION => 'Confession religieuse'

];

}

/\*\*

\* Obtenir la liste des types d'opération comme options

\*/

public static function getTypeOperationOptions(): array

{

return [

Dossier::TYPE\_CREATION => 'Création',

Dossier::TYPE\_MODIFICATION => 'Modification',

Dossier::TYPE\_CESSATION => 'Cessation',

Dossier::TYPE\_DECLARATION => 'Déclaration',

Dossier::TYPE\_FUSION => 'Fusion',

Dossier::TYPE\_ABSORPTION => 'Absorption'

];

}

}

### 6. Fondateur (app/Models/Fondateur.php)

[Code complet fourni dans la discussion...]

### 7. Etablissement (app/Models/Etablissement.php)

[Code complet fourni dans la discussion...]

### 8. WorkflowStep (app/Models/WorkflowStep.php)

[Code complet fourni dans la discussion...]

### 9. ValidationEntity (app/Models/ValidationEntity.php)

[Code complet fourni dans la discussion...]

### 10. DossierValidation (app/Models/DossierValidation.php)

[Code complet fourni dans la discussion...]

### 11. User (app/Models/User.php) - Mise à jour

[Code complet fourni dans la discussion...]

## 📝 Migration créée

### Migration add\_pngdi\_fields\_to\_users\_table

[Code complet de la migration fourni dans la discussion...]

## 🚀 Prochaines étapes (Discussion 6 - Suite)

### Étape 2 : Services métier

* OrganisationValidationService
* WorkflowService
* AdherentImportService
* QrCodeService
* DossierService
* NotificationService

### Étape 3 : Contrôleurs

* OrganisationController (create, store, edit, update)
* GuideController (show guides before operations)
* DossierController (submit, track, modify)
* DocumentController (upload, download, preview)
* AdherentController (add, import, export)

### Étape 4 : Vues

* Dashboard opérateur
* Formulaire création organisation multi-étapes
* Guide de procédure
* Upload de documents avec drag & drop
* Liste des dossiers avec statuts
* Gestion des adhérents

### Étape 5 : Middleware

* CheckOrganisationLimit
* DossierLock
* VerifyOperatorRole

## 💡 Points clés réalisés

1. **Architecture robuste** : 11 modèles avec relations complètes
2. **Gestion du workflow** : Système flexible et paramétrable
3. **Unicité des adhérents** : Contrôle pour partis politiques
4. **Documents sécurisés** : Hash, validation, stockage
5. **Géolocalisation** : Complète pour organisations et établissements
6. **Historisation** : Traçabilité complète des actions
7. **Permissions** : Système de rôles intégré

## 📋 Pour continuer

1. **Vérifier** que tous les modèles sont créés
2. **Exécuter** la migration des users
3. **Commencer** l'étape 2 avec les services métier
4. **Créer** les modèles manquants si nécessaire (ActivityLog, Message, etc.)

## 🎉 Bilan de la Discussion 6 - Étape 1

**Statut** : ✅ **Étape 1 TERMINÉE**

* 11 modèles principaux créés
* Relations établies
* Méthodes métier implémentées
* Migration User créée

Prêt pour l'**Étape 2 : Services métier** dans la prochaine discussion !

Voici la synthèse de la Discussion 6 - Modele. Passons à la discussion 6 : **Suite.**

Le projet suit une logique de développement séquentiel avec validation manuelle à chaque action. Ne passe à l'étape suivante qu’après ma validation explicite.

Travail étape par étape. À 90 % de la limite de message, génère un fichier synthèse avec tout l’historique, code inclus.

Les Cahiers de charge complet et la structuration du projet, déjà partagé sont ci-joint,

Considère tous les détails.