Analyse en profondeur les documents ci-joints. Analyse en profondeur les connaissances du projet. Projet : SGLP. Backend : Laravel (PHP 7.3.29 et MySQL). Frontend : Bootstrap 5. Prend en compte les instruction suivantes : - Travail séquentiel - Vérifier l’existant - Plan d'action au démarrage - Pour les mises à jour de code existant Utiliser des délimiteurs textuels exacts et non des numéros de ligne. Me guider pas à pas pour intégrer les modifications. -Si le code devient volumineux, le séquencer en sous-fichiers pour plus de clarté et faciliter la maintenance. - A l'approche de 90 % de ta limite de saturation de conversation, générer un fichier de synthèse détaillé suffisamment documenté pour servir d'information d'entrées pour une autre discussion. L'objectif de cette discussion est de rendre opérationnel la vue confirmation de sorte a uploader et inserer en base de donnée des listes d'adhérents. commencons

Modifier

Je vais analyser en profondeur votre projet SGLP et créer un plan d'action structuré pour rendre opérationnelle la vue confirmation pour l'upload et l'insertion en base de données des listes d'adhérents.

**📋 ANALYSE PRÉALABLE DU PROJET SGLP**

D'après mon analyse des documents, je constate :

**✅ Existant :**

* Vue confirmation.blade.php avec interface modernisée
* Système de chunking JavaScript fonctionnel (chunking-import.js)
* Contrôleurs Laravel partiels (DossierController, ChunkingController)
* Architecture modulaire avec Bootstrap 5

**❌ Problèmes identifiés :**

1. Routes chunking manquantes ou incomplètes
2. Contrôleur ChunkProcessorController absent/incomplet
3. Validation et insertion en base non finalisées
4. Workflow Phase 2 non fonctionnel

**🎯 PLAN D'ACTION SÉQUENTIEL**

**ÉTAPE 1 : VÉRIFICATION ET CRÉATION DES ROUTES**

<?php

// ========================================

// ROUTES CHUNKING - À AJOUTER À LA FIN DE routes/web.php

// ========================================

use App\Http\Controllers\Api\ChunkProcessorController;

// Routes chunking avec middleware auth + verified

Route::middleware(['auth', 'verified'])->group(function () {

// Routes chunking Phase 2 - Priority WEB selon solution Discussion 39

Route::prefix('chunking')->group(function () {

// Route principale de traitement des chunks

Route::post('/process-chunk', [ChunkProcessorController::class, 'processChunk'])

->name('chunking.process-chunk');

// Route de refresh CSRF

Route::get('/csrf-refresh', [ChunkProcessorController::class, 'refreshCSRF'])

->name('chunking.csrf-refresh');

// Route de health check

Route::get('/health', [ChunkProcessorController::class, 'healthCheck'])

->name('chunking.health');

// Route de test authentification

Route::get('/auth-test', [ChunkProcessorController::class, 'authTest'])

->name('chunking.auth-test');

// Route statistiques performance

Route::get('/performance', [ChunkProcessorController::class, 'getPerformanceStats'])

->name('chunking.performance');

});

// Routes Phase 2 - Import adhérents

Route::prefix('operator/dossiers')->group(function () {

// Page import adhérents Phase 2

Route::get('/{dossier}/adherents-import', [App\Http\Controllers\Operator\DossierController::class, 'adherentsImport'])

->name('operator.dossiers.adherents-import');

// Traitement upload adhérents

Route::post('/{dossier}/store-adherents', [App\Http\Controllers\Operator\DossierController::class, 'storeAdherents'])

->name('operator.dossiers.store-adherents');

// Session management pour Phase 2

Route::post('/save-session-adherents', [App\Http\Controllers\Operator\DossierController::class, 'saveSessionAdherents'])

->name('operator.dossiers.save-session-adherents');

Route::get('/get-session-adherents', [App\Http\Controllers\Operator\DossierController::class, 'getSessionAdherents'])

->name('operator.dossiers.get-session-adherents');

});

});

// Routes chunking publiques (sans auth) pour certains endpoints

Route::prefix('chunking/public')->group(function () {

// Health check public

Route::get('/status', function () {

return response()->json([

'status' => 'operational',

'timestamp' => now(),

'service' => 'SGLP Chunking Service'

]);

})->name('chunking.public.status');

});

ÉTAPE 2 : CRÉATION DU CONTRÔLEUR ChunkProcessorController

<?php

namespace App\Http\Controllers\Api;

use App\Http\Controllers\Controller;

use Illuminate\Http\Request;

use Illuminate\Support\Facades\Log;

use Illuminate\Support\Facades\DB;

use Illuminate\Support\Facades\Validator;

use App\Models\Dossier;

use App\Models\Organisation;

use App\Models\Adherent;

use Carbon\Carbon;

/\*\*

\* ========================================================================

\* CHUNK PROCESSOR CONTROLLER - VERSION COMPLÈTE POUR SGLP

\* Contrôleur principal pour le traitement par lots des adhérents

\* Compatible PHP 7.3.29 + Laravel + MySQL

\* ========================================================================

\*/

class ChunkProcessorController extends Controller

{

/\*\*

\* Timeout de traitement par chunk (25 secondes pour éviter timeout 30s)

\*/

private $chunkTimeout = 25;

/\*\*

\* Taille maximale de batch pour optimiser la mémoire

\*/

private $maxBatchSize = 50;

/\*\*

\* Traiter un chunk d'adhérents - MÉTHODE PRINCIPALE

\*/

public function processChunk(Request $request)

{

$startTime = microtime(true);

try {

Log::info('🚀 DÉBUT TRAITEMENT CHUNK SGLP', [

'ip' => $request->ip(),

'user\_id' => auth()->id(),

'timestamp' => now(),

'memory\_start' => memory\_get\_usage(true) / 1024 / 1024 . ' MB'

]);

// Validation des paramètres d'entrée

$validator = Validator::make($request->all(), [

'chunk\_id' => 'required|integer|min:1',

'chunk\_data' => 'required|string',

'total\_chunks' => 'required|integer|min:1',

'chunk\_start\_index' => 'required|integer|min:0',

'chunk\_end\_index' => 'required|integer|min:0'

]);

if ($validator->fails()) {

return response()->json([

'success' => false,

'message' => 'Paramètres invalides',

'errors' => $validator->errors()

], 400);

}

// Récupération et décodage des données

$chunkId = $request->input('chunk\_id');

$chunkDataRaw = $request->input('chunk\_data');

$totalChunks = $request->input('total\_chunks');

$startIndex = $request->input('chunk\_start\_index');

$endIndex = $request->input('chunk\_end\_index');

// Décoder les données JSON

$chunkData = json\_decode($chunkDataRaw, true);

if (json\_last\_error() !== JSON\_ERROR\_NONE) {

throw new \Exception('Données JSON invalides: ' . json\_last\_error\_msg());

}

Log::info('📊 CHUNK REÇU', [

'chunk\_id' => $chunkId,

'total\_chunks' => $totalChunks,

'data\_count' => count($chunkData),

'start\_index' => $startIndex,

'end\_index' => $endIndex

]);

// Récupérer les informations de dossier depuis la session

$dossierId = session('current\_dossier\_id');

$organisationId = session('current\_organisation\_id');

if (!$dossierId || !$organisationId) {

return response()->json([

'success' => false,

'message' => 'Session expirée - informations de dossier manquantes'

], 422);

}

// Vérifier l'accès au dossier

$dossier = Dossier::with('organisation')

->where('id', $dossierId)

->whereHas('organisation', function($query) {

$query->where('user\_id', auth()->id());

})

->first();

if (!$dossier) {

return response()->json([

'success' => false,

'message' => 'Dossier non trouvé ou accès non autorisé'

], 404);

}

// Traitement des adhérents par batch

$result = $this->processAdherentsChunk($chunkData, $dossier->organisation, $dossier);

$endTime = microtime(true);

$processingTime = round(($endTime - $startTime), 2);

Log::info('✅ CHUNK TRAITÉ AVEC SUCCÈS', [

'chunk\_id' => $chunkId,

'processed' => $result['processed'],

'valid' => $result['valid'],

'errors' => $result['errors'],

'anomalies' => $result['anomalies'],

'processing\_time' => $processingTime . 's',

'memory\_end' => memory\_get\_usage(true) / 1024 / 1024 . ' MB'

]);

return response()->json([

'success' => true,

'chunk\_id' => $chunkId,

'processed' => $result['processed'],

'valid' => $result['valid'],

'errors' => $result['errors'],

'anomalies' => $result['anomalies'],

'processing\_time' => $processingTime,

'chunk\_progress' => round(($chunkId / $totalChunks) \* 100, 1),

'memory\_usage' => memory\_get\_usage(true) / 1024 / 1024,

'message' => "Chunk {$chunkId}/{$totalChunks} traité avec succès"

]);

} catch (\Exception $e) {

$endTime = microtime(true);

$processingTime = round(($endTime - $startTime), 2);

Log::error('❌ ERREUR TRAITEMENT CHUNK', [

'chunk\_id' => $chunkId ?? 'unknown',

'error' => $e->getMessage(),

'file' => $e->getFile(),

'line' => $e->getLine(),

'processing\_time' => $processingTime . 's'

]);

return response()->json([

'success' => false,

'message' => 'Erreur traitement chunk: ' . $e->getMessage(),

'chunk\_id' => $chunkId ?? null,

'processing\_time' => $processingTime

], 500);

}

}

/\*\*

\* Traiter les adhérents d'un chunk avec système d'anomalies

\*/

private function processAdherentsChunk($adherentsData, $organisation, $dossier)

{

$processed = 0;

$valid = 0;

$errors = 0;

$anomalies = 0;

$errorDetails = [];

DB::beginTransaction();

try {

// Traiter par batch pour optimiser la mémoire

$batches = array\_chunk($adherentsData, $this->maxBatchSize);

foreach ($batches as $batchIndex => $batch) {

Log::info("🔄 Traitement batch {$batchIndex}", [

'batch\_size' => count($batch),

'organisation\_id' => $organisation->id

]);

foreach ($batch as $index => $adherentData) {

$processed++;

try {

// Validation et nettoyage des données

$cleanedData = $this->validateAndCleanAdherent($adherentData, $organisation);

// Vérifier les doublons par NIP

$existingAdherent = Adherent::where('nip', $cleanedData['nip'])

->where('organisation\_id', $organisation->id)

->first();

if ($existingAdherent) {

Log::warning('⚠️ Doublon détecté', [

'nip' => $cleanedData['nip'],

'nom' => $cleanedData['nom'],

'existing\_id' => $existingAdherent->id

]);

$anomalies++;

continue;

}

// Créer l'adhérent

$adherent = $this->saveAdherent($cleanedData, $organisation, $dossier);

if ($adherent) {

$valid++;

}

// Vérifier le timeout

if ((microtime(true) - LARAVEL\_START) > $this->chunkTimeout) {

Log::warning('⏰ Timeout proche - arrêt préventif du batch');

break 2;

}

} catch (\Exception $e) {

$errors++;

$errorDetails[] = [

'index' => $index,

'nip' => $adherentData['nip'] ?? 'N/A',

'nom' => $adherentData['nom'] ?? 'N/A',

'error' => $e->getMessage()

];

Log::warning('⚠️ Erreur traitement adhérent', [

'index' => $index,

'error' => $e->getMessage(),

'data' => $adherentData

]);

}

}

}

DB::commit();

return [

'processed' => $processed,

'valid' => $valid,

'errors' => $errors,

'anomalies' => $anomalies,

'error\_details' => $errorDetails

];

} catch (\Exception $e) {

DB::rollback();

throw $e;

}

}

/\*\*

\* Valider et nettoyer les données d'un adhérent

\*/

private function validateAndCleanAdherent($data, $organisation)

{

// Validation du NIP (format XX-QQQQ-YYYYMMDD)

$nip = $data['nip'] ?? '';

if (!preg\_match('/^[A-Z0-9]{2}-[0-9]{4}-[0-9]{8}$/', $nip)) {

throw new \Exception("Format NIP invalide: {$nip}");

}

// Validation des champs obligatoires

$requiredFields = ['nom', 'prenom', 'civilite'];

foreach ($requiredFields as $field) {

if (empty($data[$field])) {

throw new \Exception("Champ obligatoire manquant: {$field}");

}

}

// Nettoyage et formatage

return [

'nip' => strtoupper(trim($nip)),

'civilite' => trim($data['civilite']),

'nom' => strtoupper(trim($data['nom'])),

'prenom' => ucwords(strtolower(trim($data['prenom']))),

'telephone' => $this->cleanPhone($data['telephone'] ?? ''),

'profession' => trim($data['profession'] ?? ''),

'adresse' => trim($data['adresse'] ?? ''),

'date\_naissance' => $this->extractDateFromNip($nip),

'age' => $this->calculateAgeFromNip($nip)

];

}

/\*\*

\* Sauvegarder un adhérent en base

\*/

private function saveAdherent($data, $organisation, $dossier)

{

return Adherent::create([

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

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

'nip' => $data['nip'],

'civilite' => $data['civilite'],

'nom' => $data['nom'],

'prenom' => $data['prenom'],

'telephone' => $data['telephone'],

'profession' => $data['profession'],

'adresse' => $data['adresse'],

'date\_naissance' => $data['date\_naissance'],

'age' => $data['age'],

'date\_adhesion' => now(),

'is\_active' => true,

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

]);

}

/\*\*

\* Nettoyer le numéro de téléphone

\*/

private function cleanPhone($phone)

{

if (empty($phone)) return null;

// Retirer les espaces et caractères spéciaux

$cleaned = preg\_replace('/[^0-9]/', '', $phone);

// Ajouter le préfixe +241 si manquant

if (strlen($cleaned) === 8 && !str\_starts\_with($cleaned, '241')) {

$cleaned = '241' . $cleaned;

}

return $cleaned;

}

/\*\*

\* Extraire la date de naissance du NIP

\*/

private function extractDateFromNip($nip)

{

if (preg\_match('/^[A-Z0-9]{2}-[0-9]{4}-([0-9]{8})$/', $nip, $matches)) {

$dateStr = $matches[1];

$year = substr($dateStr, 0, 4);

$month = substr($dateStr, 4, 2);

$day = substr($dateStr, 6, 2);

try {

return Carbon::createFromFormat('Y-m-d', "{$year}-{$month}-{$day}");

} catch (\Exception $e) {

return null;

}

}

return null;

}

/\*\*

\* Calculer l'âge depuis le NIP

\*/

private function calculateAgeFromNip($nip)

{

$birthDate = $this->extractDateFromNip($nip);

if ($birthDate) {

return $birthDate->age;

}

return null;

}

/\*\*

\* Refresh du token CSRF

\*/

public function refreshCSRF(Request $request)

{

try {

$newToken = csrf\_token();

Log::info('🔄 CSRF Token refreshed', [

'user\_id' => auth()->id(),

'timestamp' => now()

]);

return response()->json([

'success' => true,

'csrf\_token' => $newToken,

'timestamp' => now()

]);

} catch (\Exception $e) {

Log::error('❌ Erreur refresh CSRF', [

'error' => $e->getMessage()

]);

return response()->json([

'success' => false,

'message' => 'Erreur refresh CSRF'

], 500);

}

}

/\*\*

\* Health check du système

\*/

public function healthCheck(Request $request)

{

try {

$authenticated = auth()->check();

$user = auth()->user();

return response()->json([

'status' => 'operational',

'user\_authenticated' => $authenticated,

'user\_id' => $user ? $user->id : null,

'timestamp' => now(),

'memory\_usage' => memory\_get\_usage(true) / 1024 / 1024,

'service' => 'SGLP Chunking Service v1.0'

]);

} catch (\Exception $e) {

return response()->json([

'status' => 'error',

'message' => $e->getMessage()

], 500);

}

}

/\*\*

\* Test d'authentification

\*/

public function authTest(Request $request)

{

try {

$user = auth()->user();

return response()->json([

'authenticated' => auth()->check(),

'user' => $user ? [

'id' => $user->id,

'name' => $user->name,

'email' => $user->email

] : null,

'session\_id' => session()->getId(),

'csrf\_token' => csrf\_token(),

'timestamp' => now()

]);

} catch (\Exception $e) {

return response()->json([

'authenticated' => false,

'error' => $e->getMessage()

], 500);

}

}

/\*\*

\* Statistiques de performance

\*/

public function getPerformanceStats(Request $request)

{

try {

return response()->json([

'memory\_usage' => memory\_get\_usage(true) / 1024 / 1024,

'memory\_peak' => memory\_get\_peak\_usage(true) / 1024 / 1024,

'execution\_time' => microtime(true) - LARAVEL\_START,

'php\_version' => PHP\_VERSION,

'laravel\_version' => app()->version(),

'timestamp' => now()

]);

} catch (\Exception $e) {

return response()->json([

'error' => $e->getMessage()

], 500);

}

}

}

ÉTAPE 3 : MISE À JOUR DU DossierController

<?php

/\*\*

\* ========================================================================

\* MÉTHODES À AJOUTER/METTRE À JOUR DANS DossierController

\* Localisation: app/Http/Controllers/Operator/DossierController.php

\* ========================================================================

\*/

// AJOUTER CES MÉTHODES DANS LA CLASSE DossierController

/\*\*

\* Page d'import des adhérents - Phase 2

\*/

public function adherentsImport($dossierId)

{

try {

// Récupérer le dossier

$dossier = Dossier::with(['organisation', 'adherents'])

->where('id', $dossierId)

->whereHas('organisation', function($query) {

$query->where('user\_id', auth()->id());

})

->firstOrFail();

$organisation = $dossier->organisation;

// Statistiques adhérents

$adherents\_stats = [

'existants' => $dossier->adherents()->count(),

'minimum\_requis' => $this->getMinimumAdherentsRequired($organisation->type),

'manquants' => 0,

'peut\_soumettre' => false

];

$adherents\_stats['manquants'] = max(0, $adherents\_stats['minimum\_requis'] - $adherents\_stats['existants']);

$adherents\_stats['peut\_soumettre'] = $adherents\_stats['manquants'] <= 0;

// Configuration upload

$upload\_config = [

'max\_file\_size' => '10MB',

'chunk\_size' => 100,

'max\_adherents' => 50000,

'chunking\_threshold' => 200

];

// URLs pour les actions

$urls = [

'store\_adherents' => route('operator.dossiers.store-adherents', $dossier->id),

'template\_download' => route('operator.templates.adherents-excel'),

'confirmation' => route('operator.dossiers.confirmation', $dossier->id),

'process\_chunk' => route('chunking.process-chunk'),

'health\_check' => route('chunking.health')

];

// Stocker les informations en session pour le chunking

session([

'current\_dossier\_id' => $dossier->id,

'current\_organisation\_id' => $organisation->id

]);

Log::info('📄 PAGE IMPORT ADHÉRENTS PHASE 2', [

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

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

'adherents\_existants' => $adherents\_stats['existants'],

'user\_id' => auth()->id()

]);

return view('operator.dossiers.adherents-import', compact(

'dossier',

'organisation',

'adherents\_stats',

'upload\_config',

'urls'

));

} catch (\Exception $e) {

Log::error('❌ Erreur page import adhérents', [

'dossier\_id' => $dossierId,

'error' => $e->getMessage(),

'user\_id' => auth()->id()

]);

return redirect()->route('operator.dashboard')

->with('error', 'Erreur lors du chargement de la page d\'import');

}

}

/\*\*

\* Confirmation finale - Phase 2 complétée

\*/

public function confirmation($dossierId)

{

try {

// Récupérer le dossier avec toutes ses relations

$dossier = Dossier::with(['organisation', 'adherents', 'documents'])

->where('id', $dossierId)

->whereHas('organisation', function($query) {

$query->where('user\_id', auth()->id());

})

->firstOrFail();

$organisation = $dossier->organisation;

// Statistiques finales

$adherents\_stats = [

'existants' => $dossier->adherents()->count(),

'minimum\_requis' => $this->getMinimumAdherentsRequired($organisation->type),

'manquants' => 0,

'peut\_soumettre' => false

];

$adherents\_stats['manquants'] = max(0, $adherents\_stats['minimum\_requis'] - $adherents\_stats['existants']);

$adherents\_stats['peut\_soumettre'] = $adherents\_stats['manquants'] <= 0;

// Configuration pour interface finale

$upload\_config = [

'max\_file\_size' => '10MB',

'chunk\_size' => 100,

'max\_adherents' => 50000,

'chunking\_threshold' => 200

];

// URLs finales

$urls = [

'store\_adherents' => route('operator.dossiers.store-adherents', $dossier->id),

'template\_download' => route('operator.templates.adherents-excel'),

'process\_chunk' => route('chunking.process-chunk'),

'health\_check' => route('chunking.health'),

'final\_submit' => route('operator.dossiers.final-submit', $dossier->id)

];

Log::info('📋 PAGE CONFIRMATION FINALE', [

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

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

'adherents\_count' => $adherents\_stats['existants'],

'peut\_soumettre' => $adherents\_stats['peut\_soumettre'],

'user\_id' => auth()->id()

]);

return view('operator.dossiers.confirmation', compact(

'dossier',

'organisation',

'adherents\_stats',

'upload\_config',

'urls'

));

} catch (\Exception $e) {

Log::error('❌ Erreur page confirmation', [

'dossier\_id' => $dossierId,

'error' => $e->getMessage(),

'user\_id' => auth()->id()

]);

return redirect()->route('operator.dashboard')

->with('error', 'Erreur lors du chargement de la confirmation');

}

}

/\*\*

\* Stocker les adhérents - Compatible chunking et upload classique

\*/

public function storeAdherents(Request $request, $dossierId)

{

$startTime = microtime(true);

try {

Log::info('📥 DÉBUT STORE ADHÉRENTS', [

'dossier\_id' => $dossierId,

'user\_id' => auth()->id(),

'request\_size' => strlen(json\_encode($request->all())),

'is\_chunked' => $request->has('chunk\_id'),

'timestamp' => now()

]);

// Vérifier l'accès au dossier

$dossier = Dossier::with('organisation')

->where('id', $dossierId)

->whereHas('organisation', function($query) {

$query->where('user\_id', auth()->id());

})

->firstOrFail();

$organisation = $dossier->organisation;

// Déterminer le mode de traitement

if ($request->has('chunk\_id')) {

// Mode chunking - déléguer au ChunkProcessorController

return $this->handleChunkRequest($request, $dossier, $organisation);

} else {

// Mode classique - traitement direct

return $this->handleClassicUpload($request, $dossier, $organisation);

}

} catch (\Exception $e) {

$processingTime = round((microtime(true) - $startTime), 2);

Log::error('❌ ERREUR STORE ADHÉRENTS', [

'dossier\_id' => $dossierId,

'error' => $e->getMessage(),

'file' => $e->getFile(),

'line' => $e->getLine(),

'processing\_time' => $processingTime . 's',

'user\_id' => auth()->id()

]);

if ($request->expectsJson()) {

return response()->json([

'success' => false,

'message' => 'Erreur lors du traitement: ' . $e->getMessage(),

'processing\_time' => $processingTime

], 500);

}

return redirect()->back()

->with('error', 'Erreur lors de l\'import des adhérents: ' . $e->getMessage());

}

}

/\*\*

\* Gérer une requête en mode chunking

\*/

private function handleChunkRequest($request, $dossier, $organisation)

{

// Stocker les informations en session pour le ChunkProcessorController

session([

'current\_dossier\_id' => $dossier->id,

'current\_organisation\_id' => $organisation->id

]);

// Rediriger vers le ChunkProcessorController

$chunkController = new \App\Http\Controllers\Api\ChunkProcessorController();

return $chunkController->processChunk($request);

}

/\*\*

\* Gérer un upload classique (sans chunking)

\*/

private function handleClassicUpload($request, $dossier, $organisation)

{

// Validation du fichier ou des données

if ($request->hasFile('adherents\_file')) {

return $this->processFileUpload($request, $dossier, $organisation);

} elseif ($request->has('adherents\_data')) {

return $this->processJsonData($request, $dossier, $organisation);

} else {

throw new \Exception('Aucune donnée d\'adhérents fournie');

}

}

/\*\*

\* Traiter l'upload d'un fichier

\*/

private function processFileUpload($request, $dossier, $organisation)

{

$file = $request->file('adherents\_file');

// Valider le fichier

$request->validate([

'adherents\_file' => 'required|file|mimes:xlsx,csv|max:10240' // 10MB max

]);

// Traiter le fichier selon son type

if ($file->getClientOriginalExtension() === 'csv') {

$adherentsData = $this->parseCsvFile($file);

} else {

$adherentsData = $this->parseExcelFile($file);

}

// Traiter les données

return $this->processAdherentsData($adherentsData, $dossier, $organisation);

}

/\*\*

\* Traiter des données JSON

\*/

private function processJsonData($request, $dossier, $organisation)

{

$adherentsJson = $request->input('adherents\_data');

$adherentsData = json\_decode($adherentsJson, true);

if (json\_last\_error() !== JSON\_ERROR\_NONE) {

throw new \Exception('Données JSON invalides');

}

return $this->processAdherentsData($adherentsData, $dossier, $organisation);

}

/\*\*

\* Traiter les données d'adhérents avec validation et insertion

\*/

private function processAdherentsData($adherentsData, $dossier, $organisation)

{

$successCount = 0;

$errorCount = 0;

$anomaliesCount = 0;

$errors = [];

DB::beginTransaction();

try {

foreach ($adherentsData as $index => $adherentData) {

try {

// Validation et nettoyage

$cleanedData = $this->validateAndCleanAdherentData($adherentData);

// Vérifier doublon

$existingAdherent = Adherent::where('nip', $cleanedData['nip'])

->where('organisation\_id', $organisation->id)

->first();

if ($existingAdherent) {

$anomaliesCount++;

Log::warning('Doublon détecté', [

'nip' => $cleanedData['nip'],

'nom' => $cleanedData['nom']

]);

continue;

}

// Créer l'adhérent

Adherent::create([

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

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

'nip' => $cleanedData['nip'],

'civilite' => $cleanedData['civilite'],

'nom' => $cleanedData['nom'],

'prenom' => $cleanedData['prenom'],

'telephone' => $cleanedData['telephone'],

'profession' => $cleanedData['profession'],

'adresse' => $cleanedData['adresse'],

'date\_naissance' => $cleanedData['date\_naissance'],

'age' => $cleanedData['age'],

'date\_adhesion' => now(),

'is\_active' => true,

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

]);

$successCount++;

} catch (\Exception $e) {

$errorCount++;

$errors[] = [

'index' => $index,

'nip' => $adherentData['nip'] ?? 'N/A',

'nom' => $adherentData['nom'] ?? 'N/A',

'error' => $e->getMessage()

];

}

}

DB::commit();

Log::info('✅ IMPORT TERMINÉ', [

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

'total\_processed' => count($adherentsData),

'success\_count' => $successCount,

'error\_count' => $errorCount,

'anomalies\_count' => $anomaliesCount

]);

if (request()->expectsJson()) {

return response()->json([

'success' => true,

'message' => "Import terminé : {$successCount} adhérents créés",

'data' => [

'total\_processed' => count($adherentsData),

'success\_count' => $successCount,

'error\_count' => $errorCount,

'anomalies\_count' => $anomaliesCount,

'errors' => $errors

]

]);

}

return redirect()->route('operator.dossiers.confirmation', $dossier->id)

->with('success', "Import réussi : {$successCount} adhérents ajoutés");

} catch (\Exception $e) {

DB::rollback();

throw $e;

}

}

/\*\*

\* Valider et nettoyer les données d'un adhérent

\*/

private function validateAndCleanAdherentData($data)

{

// Validation NIP

$nip = $data['nip'] ?? '';

if (!preg\_match('/^[A-Z0-9]{2}-[0-9]{4}-[0-9]{8}$/', $nip)) {

throw new \Exception("Format NIP invalide: {$nip}");

}

// Validation champs obligatoires

$requiredFields = ['nom', 'prenom', 'civilite'];

foreach ($requiredFields as $field) {

if (empty($data[$field])) {

throw new \Exception("Champ obligatoire manquant: {$field}");

}

}

return [

'nip' => strtoupper(trim($nip)),

'civilite' => trim($data['civilite']),

'nom' => strtoupper(trim($data['nom'])),

'prenom' => ucwords(strtolower(trim($data['prenom']))),

'telephone' => $this->cleanPhoneNumber($data['telephone'] ?? ''),

'profession' => trim($data['profession'] ?? ''),

'adresse' => trim($data['adresse'] ?? ''),

'date\_naissance' => $this->extractDateFromNip($nip),

'age' => $this->calculateAgeFromNip($nip)

];

}

/\*\*

\* Nettoyer numéro de téléphone

\*/

private function cleanPhoneNumber($phone)

{

if (empty($phone)) return null;

$cleaned = preg\_replace('/[^0-9]/', '', $phone);

if (strlen($cleaned) === 8 && !str\_starts\_with($cleaned, '241')) {

$cleaned = '241' . $cleaned;

}

return $cleaned;

}

/\*\*

\* Extraire date de naissance du NIP

\*/

private function extractDateFromNip($nip)

{

if (preg\_match('/^[A-Z0-9]{2}-[0-9]{4}-([0-9]{8})$/', $nip, $matches)) {

$dateStr = $matches[1];

$year = substr($dateStr, 0, 4);

$month = substr($dateStr, 4, 2);

$day = substr($dateStr, 6, 2);

try {

return \Carbon\Carbon::createFromFormat('Y-m-d', "{$year}-{$month}-{$day}");

} catch (\Exception $e) {

return null;

}

}

return null;

}

/\*\*

\* Calculer âge depuis le NIP

\*/

private function calculateAgeFromNip($nip)

{

$birthDate = $this->extractDateFromNip($nip);

return $birthDate ? $birthDate->age : null;

}

/\*\*

\* Obtenir le minimum d'adhérents requis selon le type d'organisation

\*/

private function getMinimumAdherentsRequired($type)

{

$minimums = [

'association' => 10,

'ong' => 15,

'parti\_politique' => 50,

'confession\_religieuse' => 10

];

return $minimums[$type] ?? 10;

}

/\*\*

\* Parser un fichier CSV

\*/

private function parseCsvFile($file)

{

$data = [];

$handle = fopen($file->getRealPath(), 'r');

if ($handle) {

$headers = fgetcsv($handle);

while (($row = fgetcsv($handle)) !== false) {

if (count($row) >= count($headers)) {

$data[] = array\_combine($headers, $row);

}

}

fclose($handle);

}

return $data;

}

/\*\*

\* Parser un fichier Excel (nécessite PhpSpreadsheet)

\*/

private function parseExcelFile($file)

{

// Implémentation basique - à adapter selon vos besoins

// Nécessite l'installation de PhpSpreadsheet via Composer

throw new \Exception('Support Excel en cours d\'implémentation');

}

// FIN DES MÉTHODES À AJOUTER

### ****ÉTAPE 4 : GUIDE D'INTÉGRATION PAS À PAS****

Maintenant, je vais vous guider pour intégrer ces modifications :

#### **4.1 - Ajouter les routes**

1. **Ouvrez le fichier routes/web.php**
2. **Localisez la fin du fichier** (avant la dernière accolade } s'il y en a une)
3. **Ajoutez le contenu** de l'artifact "Routes Web Chunking" à la fin du fichier

#### **4.2 - Créer le contrôleur ChunkProcessorController**

1. **Créez le répertoire** app/Http/Controllers/Api/ s'il n'existe pas
2. **Créez le fichier** app/Http/Controllers/Api/ChunkProcessorController.php
3. **Copiez le contenu complet** de l'artifact "ChunkProcessorController"

#### **4.3 - Mettre à jour DossierController**

1. **Ouvrez le fichier** app/Http/Controllers/Operator/DossierController.php
2. **Localisez la fin de la classe** (avant l'accolade fermante finale)
3. **Ajoutez toutes les méthodes** de l'artifact "Méthodes DossierController"

#### **4.4 - Vérifier les dépendances**

Assurez-vous que ces lignes sont présentes au début de DossierController.php :

use Illuminate\Support\Facades\Log;

use Illuminate\Support\Facades\DB;

use App\Models\Dossier;

use App\Models\Organisation;

use App\Models\Adherent;

use Carbon\Carbon;

ÉTAPE 5 : CRÉATION DE LA VUE ADHERENTS-IMPORT