];

// Log pour l'instant (vous pouvez sauvegarder en DB plus tard)

\Log::info('Form Analytics', $analytics);

return response()->json([

'success' => true,

'message' => 'Analytics enregistrées'

]);

})->name('form-analytics');

/\*\*

\* Validation complète avant soumission

\* POST /api/v1/validate-complete-form

\*/

Route::post('/validate-complete-form', function (Request $request) {

$request->validate([

'form\_data' => 'required|array'

]);

$formData = $request->input('form\_data');

$errors = [];

// Validation basique - vous pouvez étendre selon vos besoins

if (!isset($formData['metadata']['selectedOrgType']) || empty($formData['metadata']['selectedOrgType'])) {

$errors[] = 'Type d\'organisation non sélectionné';

}

if (!isset($formData['steps'][3]['demandeur\_nip']) || empty($formData['steps'][3]['demandeur\_nip'])) {

$errors[] = 'NIP du demandeur manquant';

}

if (!isset($formData['steps'][4]['org\_nom']) || empty($formData['steps'][4]['org\_nom'])) {

$errors[] = 'Nom de l\'organisation manquant';

}

return response()->json([

'valid' => count($errors) === 0,

'errors' => $errors,

'message' => count($errors) === 0

? 'Formulaire valide'

: 'Erreurs de validation détectées'

]);

})->name('validate-complete-form');

/\*\*

\* 🔧 NOUVELLES ROUTES VALIDATION NIP

\* Format: XX-QQQQ-YYYYMMDD

\*/

// Validation NIP en temps réel

Route::post('/validate-nip', function (Request $request) {

try {

$request->validate([

'nip' => 'required|string|max:20'

]);

$nip = $request->input('nip');

// Pattern NIP gabonais: XX-QQQQ-YYYYMMDD

$pattern = '/^[A-Z0-9]{2}-[0-9]{4}-[0-9]{8}$/';

$response = [

'success' => true,

'valid' => false,

'formatted' => $nip,

'errors' => []

];

if (preg\_match($pattern, $nip)) {

$parts = explode('-', $nip);

$dateStr = $parts[2]; // YYYYMMDD

// Vérifier la date

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

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

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

if (checkdate($month, $day, $year)) {

$birthDate = \Carbon\Carbon::createFromFormat('Y-m-d', $year . '-' . $month . '-' . $day);

$age = $birthDate->diffInYears(now());

if ($age >= 16) {

$response['valid'] = true;

$response['age'] = $age;

$response['birth\_date'] = $birthDate->format('d/m/Y');

} else {

$response['errors'][] = 'Âge minimum requis: 16 ans';

}

} else {

$response['errors'][] = 'Date de naissance invalide';

}

} else {

$response['errors'][] = 'Format invalide. Attendu: XX-QQQQ-YYYYMMDD';

}

return response()->json($response);

} catch (\Exception $e) {

return response()->json([

'success' => false,

'message' => 'Erreur validation NIP: ' . $e->getMessage()

], 500);

}

})->name('validate-nip');

// Générer exemples de NIP valides

Route::get('/generate-nip-example', function () {

try {

// Générer des exemples de NIP valides

$examples = [];

$prefixes = ['A1', 'B2', 'C3', '1A', '2B', '3C'];

$sequences = ['0001', '1234', '5678', '9999'];

foreach (range(1, 5) as $i) {

$prefix = $prefixes[array\_rand($prefixes)];

$sequence = $sequences[array\_rand($sequences)];

// Date aléatoire entre 1960 et 2005

$year = rand(1960, 2005);

$month = rand(1, 12);

$day = rand(1, 28); // Éviter les problèmes de jours invalides

$dateStr = sprintf('%04d%02d%02d', $year, $month, $day);

$example = $prefix . '-' . $sequence . '-' . $dateStr;

$examples[] = [

'nip' => $example,

'prefix' => $prefix,

'sequence' => $sequence,

'birth\_date' => sprintf('%04d-%02d-%02d', $year, $month, $day),

'age' => now()->diffInYears(\Carbon\Carbon::createFromFormat('Y-m-d', sprintf('%04d-%02d-%02d', $year, $month, $day)))

];

}

return response()->json([

'success' => true,

'examples' => $examples,

'format' => 'XX-QQQQ-YYYYMMDD',

'description' => [

'XX' => '2 caractères alphanumériques',

'QQQQ' => '4 chiffres',

'YYYYMMDD' => 'Date de naissance (ANNÉE MOIS JOUR)'

]

]);

} catch (\Exception $e) {

return response()->json([

'success' => false,

'message' => 'Erreur génération exemples',

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

], 500);

}

})->name('generate-nip-example');

// Validation de lot de NIP

Route::post('/validate-nip-batch', function (Request $request) {

try {

$request->validate([

'nips' => 'required|array',

'nips.\*' => 'required|string|max:20'

]);

$nips = $request->input('nips');

$results = [];

$validCount = 0;

$invalidCount = 0;

foreach ($nips as $nip) {

$pattern = '/^[A-Z0-9]{2}-[0-9]{4}-[0-9]{8}$/';

$isValid = false;

$errors = [];

$age = null;

if (preg\_match($pattern, $nip)) {

$parts = explode('-', $nip);

$dateStr = $parts[2];

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

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

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

if (checkdate($month, $day, $year)) {

$birthDate = \Carbon\Carbon::createFromFormat('Y-m-d', $year . '-' . $month . '-' . $day);

$age = $birthDate->diffInYears(now());

if ($age >= 16) {

$isValid = true;

$validCount++;

} else {

$errors[] = 'Âge minimum requis: 16 ans';

$invalidCount++;

}

} else {

$errors[] = 'Date de naissance invalide';

$invalidCount++;

}

} else {

$errors[] = 'Format invalide';

$invalidCount++;

}

$results[] = [

'nip' => $nip,

'valid' => $isValid,

'age' => $age,

'errors' => $errors

];

}

return response()->json([

'success' => true,

'total' => count($nips),

'valid\_count' => $validCount,

'invalid\_count' => $invalidCount,

'results' => $results

]);

} catch (\Exception $e) {

return response()->json([

'success' => false,

'message' => 'Erreur validation lot NIP: ' . $e->getMessage()

], 500);

}

})->name('validate-nip-batch');

});

/\*

|--------------------------------------------------------------------------

| Routes API pour Gestion par Étapes - NOUVELLES ROUTES COMPLÈTES

|--------------------------------------------------------------------------

\*/

Route::prefix('api/v1/organisation')->name('api.organisation.')->middleware(['auth', 'throttle:120,1'])->group(function () {

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

// GESTION DES ÉTAPES - NOUVELLES ROUTES

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

/\*\*

\* Sauvegarder une étape spécifique

\* POST /api/v1/organisation/step/{step}/save

\*/

Route::post('/step/{step}/save', function (Request $request, $step) {

try {

$request->validate([

'data' => 'required|array',

'session\_id' => 'nullable|string'

]);

// Simulation de sauvegarde d'étape

\Log::info('Sauvegarde étape', [

'step' => $step,

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

'data\_keys' => array\_keys($request->input('data'))

]);

return response()->json([

'success' => true,

'step' => (int) $step,

'message' => "Étape {$step} sauvegardée avec succès"

]);

} catch (\Exception $e) {

return response()->json([

'success' => false,

'message' => 'Erreur sauvegarde: ' . $e->getMessage()

], 500);

}

})->name('step.save');

/\*\*

\* Valider une étape sans sauvegarder

\* POST /api/v1/organisation/step/{step}/validate

\*/

Route::post('/step/{step}/validate', function (Request $request, $step) {

try {

$request->validate([

'data' => 'required|array'

]);

$data = $request->input('data');

$errors = [];

// Validation basique selon l'étape

switch ((int) $step) {

case 3:

if (empty($data['demandeur\_nip'])) {

$errors[] = 'NIP du demandeur requis';

}

break;

case 4:

if (empty($data['org\_nom'])) {

$errors[] = 'Nom de l\'organisation requis';

}

break;

}

return response()->json([

'success' => true,

'valid' => count($errors) === 0,

'errors' => $errors,

'step' => (int) $step

]);

} catch (\Exception $e) {

return response()->json([

'success' => false,

'message' => 'Erreur validation: ' . $e->getMessage()

], 500);

}

})->name('step.validate');

/\*\*

\* Récupérer les données d'une étape

\* GET /api/v1/organisation/draft/{draftId}/step/{step}

\*/

Route::get('/draft/{draftId}/step/{step}', function ($draftId, $step) {

try {

// Simulation de récupération de données d'étape

return response()->json([

'success' => true,

'data' => [],

'step' => (int) $step,

'draft\_id' => (int) $draftId

]);

} catch (\Exception $e) {

return response()->json([

'success' => false,

'message' => 'Erreur récupération: ' . $e->getMessage()

], 500);

}

})->name('draft.step.get');

/\*\*

\* Vérifier si on peut accéder à une étape

\* GET /api/v1/organisation/draft/{draftId}/step/{step}/can-access

\*/

Route::get('/draft/{draftId}/step/{step}/can-access', function ($draftId, $step) {

return response()->json([

'success' => true,

'can\_access' => true, // Simulation - toujours accessible

'step' => (int) $step,

'draft\_id' => (int) $draftId

]);

})->name('draft.step.can-access');

/\*\*

\* Marquer une étape comme complétée

\* POST /api/v1/organisation/step/{step}/complete

\*/

Route::post('/step/{step}/complete', function (Request $request, $step) {

try {

$request->validate([

'draft\_id' => 'required|integer',

'data' => 'required|array'

]);

\Log::info('Étape complétée', [

'step' => $step,

'draft\_id' => $request->input('draft\_id'),

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

]);

return response()->json([

'success' => true,

'step' => (int) $step,

'message' => "Étape {$step} complétée avec succès"

]);

} catch (\Exception $e) {

return response()->json([

'success' => false,

'message' => 'Erreur completion: ' . $e->getMessage()

], 500);

}

})->name('step.complete');

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

// GESTION DES BROUILLONS

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

/\*\*

\* Créer un nouveau brouillon

\* POST /api/v1/organisation/draft/create

\*/

Route::post('/draft/create', function (Request $request) {

try {

$request->validate([

'organization\_type' => 'nullable|in:association,ong,parti\_politique,confession\_religieuse',

'session\_id' => 'nullable|string'

]);

// Simulation de création de brouillon

$draftId = rand(1000, 9999);

return response()->json([

'success' => true,

'message' => 'Brouillon créé avec succès',

'draft\_id' => $draftId,

'organization\_type' => $request->input('organization\_type'),

'current\_step' => 1

]);

} catch (\Exception $e) {

return response()->json([

'success' => false,

'message' => 'Erreur création brouillon: ' . $e->getMessage()

], 500);

}

})->name('draft.create');

/\*\*

\* Récupérer un brouillon

\* GET /api/v1/organisation/draft/{draftId}

\*/

Route::get('/draft/{draftId}', function ($draftId) {

return response()->json([

'success' => true,

'draft' => [

'id' => (int) $draftId,

'organization\_type' => 'association',

'current\_step' => 1,

'completion\_percentage' => 0,

'form\_data' => [],

'last\_saved\_at' => now()->toISOString()

],

'statistics' => [

'steps\_completed' => 0,

'total\_steps' => 8

]

]);

})->name('draft.get');

/\*\*

\* Lister les brouillons de l'utilisateur

\* GET /api/v1/organisation/drafts

\*/

Route::get('/drafts', function (Request $request) {

return response()->json([

'success' => true,

'drafts' => [],

'count' => 0

]);

})->name('drafts.list');

/\*\*

\* Supprimer un brouillon

\* DELETE /api/v1/organisation/draft/{draftId}

\*/

Route::delete('/draft/{draftId}', function ($draftId) {

return response()->json([

'success' => true,

'message' => 'Brouillon supprimé avec succès'

]);

})->name('draft.delete');

/\*\*

\* Finaliser et créer l'organisation

\* POST /api/v1/organisation/draft/{draftId}/finalize

\*/

Route::post('/draft/{draftId}/finalize', function ($draftId) {

try {

\Log::info('Finalisation organisation', [

'draft\_id' => $draftId,

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

]);

return response()->json([

'success' => true,

'organisation\_id' => rand(1000, 9999),

'dossier\_id' => rand(100, 999),

'message' => 'Organisation créée avec succès'

]);

} catch (\Exception $e) {

return response()->json([

'success' => false,

'message' => 'Erreur finalisation: ' . $e->getMessage()

], 500);

}

})->name('draft.finalize');

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

// UTILITAIRES ET STATISTIQUES

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

/\*\*

\* Obtenir les statistiques globales de l'utilisateur

\* GET /api/v1/organisation/user-stats

\*/

Route::get('/user-stats', function () {

$userId = auth()->id();

return response()->json([

'success' => true,

'statistics' => [

'total\_drafts' => 0,

'active\_drafts' => 0,

'expired\_drafts' => 0,

'completed\_organisations' => 0,

'average\_completion' => 0

]

]);

})->name('user.stats');

/\*\*

\* Nettoyer les brouillons expirés

\* POST /api/v1/organisation/cleanup-expired-drafts

\*/

Route::post('/cleanup-expired-drafts', function () {

return response()->json([

'success' => true,

'message' => "0 brouillon(s) expiré(s) supprimé(s)",

'deleted\_count' => 0

]);

})->name('cleanup.expired');

/\*\*

\* Étendre l'expiration d'un brouillon

\* POST /api/v1/organisation/draft/{draftId}/extend

\*/

Route::post('/draft/{draftId}/extend', function (Request $request, $draftId) {

$request->validate([

'days' => 'nullable|integer|min:1|max:30'

]);

$days = $request->input('days', 7);

return response()->json([

'success' => true,

'message' => "Expiration étendue de {$days} jour(s)",

'new\_expiration' => now()->addDays($days)->toISOString()

]);

})->name('draft.extend');

});

/\*

|--------------------------------------------------------------------------

| Routes pour accusés de réception par étapes

|--------------------------------------------------------------------------

\*/

Route::prefix('api/v1/accuses')->name('api.accuses.')->middleware(['auth'])->group(function () {

/\*\*

\* Générer un accusé pour une étape

\* POST /api/v1/accuses/step/{step}/generate

\*/

Route::post('/step/{step}/generate', function (Request $request, $step) {

$request->validate([

'draft\_id' => 'required|integer'

]);

return response()->json([

'success' => true,

'message' => "Accusé généré pour l'étape {$step}",

'step' => (int) $step,

'draft\_id' => $request->input('draft\_id')

]);

})->name('step.generate');

/\*\*

\* Lister les accusés d'un brouillon

\* GET /api/v1/accuses/draft/{draftId}

\*/

Route::get('/draft/{draftId}', function ($draftId) {

return response()->json([

'success' => true,

'accuses' => [],

'count' => 0,

'message' => 'Système d\'accusés en cours de développement'

]);

})->name('draft.list');

});

/\*

|--------------------------------------------------------------------------

| Routes de test (développement uniquement) - TEMPORAIRES 🧪

|--------------------------------------------------------------------------

\*/

if (config('app.debug')) {

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

// 🚀 ROUTE DE TEST POUR VALIDATION DE LA CORRECTION

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

Route::get('/test-correction-middleware', function() {

echo "<style>

body { font-family: Arial, sans-serif; margin: 20px; line-height: 1.6; }

h2 { color: #28a745; border-bottom: 2px solid #28a745; padding-bottom: 10px; }

h3 { color: #FFA500; margin-top: 25px; }

.success { color: #28a745; font-weight: bold; }

.error { color: #dc3545; font-weight: bold; }

.warning { color: #ffc107; font-weight: bold; }

.info { color: #17a2b8; }

.test-link { background: #28a745; color: white; padding: 10px 15px; text-decoration: none; border-radius: 5px; display: inline-block; margin: 10px 5px; }

.test-link:hover { background: #218838; color: white; }

</style>";

echo "<h2>🎉 TEST DE VALIDATION - CORRECTION MIDDLEWARE APPLIQUÉE</h2>";

echo "<p><strong>Solution 1 implémentée :</strong> Route confirmation exclue du middleware dossier.lock</p>";

echo "<h3>✅ Tests à effectuer :</h3>";

echo "<p><strong>Test 1 :</strong> Accès direct à la page de confirmation (sans middleware bloquant)</p>";

echo "<a href='/operator/dossiers/confirmation/19' class='test-link' target='\_blank'>➤ Tester la page de confirmation</a>";

echo "<p><em>Devrait maintenant fonctionner et afficher la page !</em></p>";

echo "<p><strong>Test 2 :</strong> Simulation de redirection (pour confirmer le fix)</p>";

echo "<a href='/test-redirect-simulation' class='test-link'>➤ Tester la redirection</a>";

echo "<p><em>La redirection devrait maintenant aboutir sur la bonne page</em></p>";

echo "<h3>🔍 Changements apportés :</h3>";

echo "<ul>";

echo "<li>✅ Route <code>/confirmation/{dossier}</code> sortie du groupe <code>middleware(['dossier.lock'])</code></li>";

echo "<li>✅ Ajout du middleware <code>throttle:60,1</code> pour protection anti-abus</li>";

echo "<li>✅ Les autres routes restent protégées par le middleware dossier.lock</li>";

echo "<li>✅ La page de confirmation est maintenant accessible en lecture seule</li>";

echo "</ul>";

echo "<h3>📋 Résultats attendus :</h3>";

echo "<ul>";

echo "<li><strong>✅ Page de confirmation accessible</strong> même si le dossier est verrouillé</li>";

echo "<li><strong>✅ Logs générés</strong> dans laravel.log lors de l'accès</li>";

echo "<li><strong>✅ Redirection fonctionnelle</strong> depuis OrganisationController::store()</li>";

echo "<li><strong>✅ Sécurité maintenue</strong> pour les autres opérations sur les dossiers</li>";

echo "</ul>";

return "";

})->middleware(['auth'])->name('test.correction.middleware');

// Routes de test existantes

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

return [

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

'php\_version' => PHP\_VERSION,

'environment' => config('app.env'),

'database\_connected' => DB::connection()->getPdo() ? 'Yes' : 'No',

'current\_user' => auth()->check() ? auth()->user()->email : 'Non connecté',

'middleware\_loaded' => [

'check.organisation.limit' => class\_exists(\App\Http\Middleware\CheckOrganisationLimit::class),

'dossier.lock' => class\_exists(\App\Http\Middleware\DossierLock::class),

'operator' => class\_exists(\App\Http\Middleware\VerifyOperatorRole::class),

]

];

})->name('test');

// Créer des utilisateurs de test

Route::get('/create-test-users', function () {

\App\Models\User::firstOrCreate(

['email' => 'admin@pngdi.ga'],

[

'name' => 'Admin PNGDI',

'password' => bcrypt('admin123'),

'role' => 'admin',

'phone' => '+24101234567',

'city' => 'Libreville',

'is\_active' => true,

'email\_verified\_at' => now(),

]

);

\App\Models\User::firstOrCreate(

['email' => 'agent@pngdi.ga'],

[

'name' => 'Agent PNGDI',

'password' => bcrypt('agent123'),

'role' => 'agent',

'phone' => '+24101234568',

'city' => 'Libreville',

'is\_active' => true,

'email\_verified\_at' => now(),

]

);

\App\Models\User::firstOrCreate(

['email' => 'operator@pngdi.ga'],

[

'name' => 'Jean NGUEMA',

'password' => bcrypt('operator123'),

'role' => 'operator',

'phone' => '+24101234569',

'city' => 'Port-Gentil',

'is\_active' => true,

'email\_verified\_at' => now(),

]

);

return 'Utilisateurs de test créés !<br>' .

'<strong>Admin :</strong> admin@pngdi.ga / admin123<br>' .

'<strong>Agent :</strong> agent@pngdi.ga / agent123<br>' .

'<strong>Opérateur :</strong> operator@pngdi.ga / operator123<br>' .

'<a href="/login">Se connecter</a>';

})->name('create-test-users');

// Route de test pour la création d'organisation

Route::post('/test-organisation-debug', function(Request $request) {

return response()->json([

'success' => true,

'message' => 'Route de test fonctionnelle',

'data' => $request->all()

]);

})->name('test-organisation-debug');

// Routes de debug existantes (conservées pour compatibilité)

Route::get('/debug-route-test', function() {

echo "<h2>Test diagnostic des routes</h2>";

echo "<h3>⚠️ ROUTE DÉPRÉCIÉE</h3>";

echo "<p>Utilisez plutôt <a href='/test-correction-middleware'>la nouvelle route de test</a></p>";

return "";

})->middleware(['auth'])->name('debug.route.test');

Route::get('/quick-debug', function() {

$dossier = \App\Models\Dossier::with('organisation')->find(19);

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

return response()->json([

'dossier\_exists' => $dossier ? true : false,

'dossier\_user\_id' => $dossier ? $dossier->organisation->user\_id : null,

'dossier\_user\_id\_type' => $dossier ? gettype($dossier->organisation->user\_id) : null,

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

'auth\_user\_id\_type' => $user ? gettype($user->id) : null,

'comparison\_result' => $dossier && $user ? ((int)$dossier->organisation->user\_id === (int)$user->id) : false,

'raw\_comparison' => $dossier && $user ? ($dossier->organisation->user\_id === $user->id) : false

]);

})->middleware(['auth']);

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

// 🔄 SIMULATION DE REDIRECTION - CORRIGÉE

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

Route::get('/test-redirect-simulation', function() {

try {

\Log::info('=== TEST REDIRECTION SIMULATION ===', [

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

'timestamp' => now()

]);

// Simuler la redirection exacte depuis OrganisationController::store()

return redirect()->route('operator.dossiers.confirmation', 19)

->with('success', 'Test de redirection depuis simulation - Correction appliquée');

} catch (\Exception $e) {

\Log::error('=== ERREUR SIMULATION REDIRECTION ===', [

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

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

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

]);

return response()->json([

'error' => 'Erreur simulation redirection',

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

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

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

]);

}

})->middleware(['auth'])->name('test.redirect.simulation');

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

// 🧪 NOUVELLE ROUTE DE TEST JAVASCRIPT

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

Route::get('/test-javascript-fixes', function() {

echo "<style>

body { font-family: Arial, sans-serif; margin: 20px; line-height: 1.6; }

h2 { color: #28a745; border-bottom: 2px solid #28a745; padding-bottom: 10px; }

h3 { color: #FFA500; margin-top: 25px; }

.success { color: #28a745; font-weight: bold; }

.error { color: #dc3545; font-weight: bold; }

.warning { color: #ffc107; font-weight: bold; }

.info { color: #17a2b8; }

.test-link { background: #28a745; color: white; padding: 10px 15px; text-decoration: none; border-radius: 5px; display: inline-block; margin: 10px 5px; }

.test-link:hover { background: #218838; color: white; }

.code-block { background: #f8f9fa; padding: 15px; border-left: 4px solid #007bff; margin: 10px 0; }

</style>";

echo "<h2>🔧 TEST DES CORRECTIONS JAVASCRIPT</h2>";

echo "<p><strong>Vérification des corrections appliquées :</strong></p>";

echo "<h3>✅ Corrections apportées :</h3>";

echo "<ul>";

echo "<li>✅ <strong>workflow-2phases.js v2.0</strong> - Erreur 'this.log is not a function' corrigée</li>";

echo "<li>✅ <strong>Routes /operator/dossiers/\*</strong> - Toutes les routes manquantes ajoutées</li>";

echo "<li>✅ <strong>chunking-import.js v1.6</strong> - Intégration workflow 2.0 optimisée</li>";

echo "</ul>";

echo "<h3>🧪 Tests à effectuer dans la console :</h3>";

echo "<div class='code-block'>";

echo "<strong>1. Test initialisation workflow :</strong><br>";

echo "<code>console.log(window.Workflow2Phases);</code><br>";

echo "<code>console.log(window.workflowInstance?.getStatus());</code>";

echo "</div>";

echo "<div class='code-block'>";

echo "<strong>2. Test chunking integration :</strong><br>";

echo "<code>console.log(window.ChunkingImport);</code><br>";

echo "<code>console.log(window.ChunkingImport?.version);</code>";

echo "</div>";

echo "<div class='code-block'>";

echo "<strong>3. Test routes API :</strong><br>";

echo "<code>fetch('/operator/dossiers/health').then(r => r.json()).then(console.log);</code><br>";

echo "<code>fetch('/operator/dossiers/csrf-refresh').then(r => r.json()).then(console.log);</code>";

echo "</div>";

echo "<h3>📋 Pages de test disponibles :</h3>";

echo "<a href='/operator/dossiers/create/parti' class='test-link'>➤ Formulaire création (parti)</a>";

echo "<a href='/operator/dossiers/create/association' class='test-link'>➤ Formulaire création (association)</a>";

echo "<a href='/operator/dossiers/confirmation/19' class='test-link'>➤ Page confirmation</a>";

echo "<h3>🎯 Résultats attendus :</h3>";

echo "<ul>";

echo "<li><strong>✅ Plus d'erreur 'this.log is not a function'</strong></li>";

echo "<li><strong>✅ Plus d'erreur 404 sur les routes /operator/dossiers/</strong></li>";

echo "<li><strong>✅ Workflow 2 phases fonctionnel</strong></li>";

echo "<li><strong>✅ Chunking v1.6 opérationnel</strong></li>";

echo "<li><strong>✅ Formulaire 8 étapes fonctionnel</strong></li>";

echo "</ul>";

return "";

})->middleware(['auth'])->name('test.javascript.fixes');

}

// Inclure les routes admin supplémentaires

if (file\_exists(\_\_DIR\_\_.'/admin.php')) {

require \_\_DIR\_\_.'/admin.php';

}

// Inclure les routes operator supplémentaires

if (file\_exists(\_\_DIR\_\_.'/operator.php')) {

require \_\_DIR\_\_.'/operator.php';

}

// Routes pour gestion CSRF longue durée

Route::get('/csrf-token', function () {

return response()->json([

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

'expires\_at' => now()->addMinutes(config('session.lifetime'))->toISOString()

]);

})->middleware('auth');

/\*

|--------------------------------------------------------------------------

| ROUTES CHUNKING - Import de gros volumes d'adhérents (WEB ALTERNATIVE)

| Compatible avec authentification session Laravel standard

|--------------------------------------------------------------------------

\*/

use App\Http\Controllers\Api\ChunkProcessorController;

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

/\*\*

\* Traitement des chunks d'adhérents (MODE WEB ALTERNATIVE)

\* POST /chunking/process-chunk

\*/

Route::post('/chunking/process-chunk', function (Request $request) {

try {

// Validation

$request->validate([

'chunk\_id' => 'required|integer',

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

'conservation\_stats' => 'sometimes|string',

'is\_chunk' => 'required|string',

'total\_chunks' => 'required|integer'

]);

// Décoder les données

$chunkData = json\_decode($request->chunk\_data, true);

$conservationStats = $request->conservation\_stats ?

json\_decode($request->conservation\_stats, true) : null;

\Log::info('Chunking Web Alternative v1.6', [

'chunk\_id' => $request->chunk\_id,

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

'conservation\_stats' => $conservationStats,

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

]);

// Simulation de traitement

$processed = count($chunkData);

$errors = 0;

return response()->json([

'success' => true,

'data' => [

'processed' => $processed,

'errors' => $errors,

'conservationStats' => $conservationStats,

'chunk\_id' => $request->chunk\_id,

'version' => '1.6'

],

'message' => "Chunk {$request->chunk\_id} traité avec succès (Web Alternative v1.6)"

]);

} catch (\Exception $e) {

\Log::error('Erreur chunking Web Alternative', [

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

'request' => $request->all()

]);

return response()->json([

'success' => false,

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

], 500);

}

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

/\*\*

\* Rafraîchissement du token CSRF (MODE WEB ALTERNATIVE)

\* GET /chunking/csrf-refresh

\*/

Route::get('/chunking/csrf-refresh', function () {

return response()->json([

'success' => true,

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

'timestamp' => now()->toISOString(),

'mode' => 'web\_alternative',

'user\_authenticated' => auth()->check(),

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

]);

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

/\*\*

\* Statistiques de performance du chunking (MODE WEB ALTERNATIVE)

\* GET /chunking/performance

\*/

Route::get('/chunking/performance', function () {

return response()->json([

'success' => true,

'performance' => [

'average\_chunk\_time' => '1.2s',

'total\_processed' => 0,

'error\_rate' => '0%',

'last\_24h\_activity' => 0

],

'mode' => 'web\_alternative',

'version' => '1.6'

]);

})->name('web.chunking.performance');

/\*\*

\* Health check pour le chunking (MODE WEB ALTERNATIVE)

\* GET /chunking/health

\*/

Route::get('/chunking/health', function () {

return response()->json([

'success' => true,

'system' => 'chunking',

'status' => 'operational',

'mode' => 'web\_routes\_alternative',

'version' => '1.6',

'timestamp' => now()->toISOString(),

'user\_authenticated' => auth()->check(),

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

'user\_role' => auth()->user()->role ?? 'N/A',

'limits' => [

'max\_chunk\_size' => 100,

'max\_execution\_time' => 25,

'memory\_limit' => '256M'

],

'middleware' => 'web + auth + verified (session-based)',

'debug\_info' => [

'csrf\_token\_present' => !empty(csrf\_token()),

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

'session\_lifetime' => config('session.lifetime')

]

]);

})->name('web.chunking.health');

});

// Route publique pour refresh CSRF (sans auth) - Alternative Web

Route::get('/chunking/csrf-refresh-public', function () {

return response()->json([

'success' => true,

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

'timestamp' => now()->toISOString(),

'mode' => 'public\_web\_alternative'

]);

})->middleware(['throttle:10,1'])->name('web.chunking.csrf-refresh-public');