**🔧 AJUSTEMENT FORMAT NIP - XX-QQQQ-YYYYMMDD**

**📋 NOUVEAU FORMAT NIP SPÉCIFIÉ**

**Format requis :**

XX-QQQQ-YYYYMMDD

**Où :**

* **XX** = 2 caractères alphanumériques (lettres + chiffres)
* **QQQQ** = 4 chiffres
* **YYYYMMDD** = Date au format ANNEEMOISJOUR

**Exemples valides :**

A1-2345-19901225 ✅

B2-0001-20000115 ✅

C3-9999-19850630 ✅

1A-5678-19950408 ✅

**Exemples invalides :**

123-4567-19901225 ❌ (XX doit être alphanumérique)

A1-23456-19901225 ❌ (QQQQ doit être 4 chiffres)

A1-2345-1990125 ❌ (Date incomplète)

A1-2345-19901355 ❌ (Date invalide)

**🔧 CORRECTIONS À APPLIQUER## 🎯 CORRECTIONS JAVASCRIPT FRONTEND## 📋 MISE À JOUR DES ROUTES API## 📋 MODIFICATIONS BASE DE DONNÉES## 📋 RÉSUMÉ COMPLET DES AJUSTEMENTS NIP**

**✅ CORRECTIONS APPLIQUÉES :**

1. **🔧 Validation PHP backend :**
   * Format regex : /^[A-Z0-9]{2}-[0-9]{4}-[0-9]{8}$/
   * Validation date de naissance extraite
   * Vérification âge (18-100 ans)
   * Détection anomalies améliorée
2. **🔧 Validation JavaScript frontend :**
   * Formatage automatique pendant saisie
   * Validation temps réel avec feedback visuel
   * Aide contextuelle pour l'utilisateur
   * Intégration avec API serveur
3. **🔧 Routes API nouvelles :**
   * /api/v1/validate-nip - Validation unitaire
   * /api/v1/generate-nip-example - Exemples
   * /api/v1/validate-nip-batch - Validation par lot
4. **🔧 Migration base de données :**
   * Mise à jour contraintes colonnes NIP
   * Index optimisés pour recherches
   * Sauvegarde données existantes
   * Commande migration artisan

**✅ EXEMPLES FORMAT NOUVEAU :**

A1-2345-19901225 (A1 = préfixe, 2345 = séquence, 19901225 = 25/12/1990)

B2-0001-20000115 (B2 = préfixe, 0001 = séquence, 20000115 = 15/01/2000)

C3-9999-19850630 (C3 = préfixe, 9999 = séquence, 19850630 = 30/06/1985)

**✅ AVANTAGES DU NOUVEAU FORMAT :**

* ✅ **Plus lisible** avec séparateurs
* ✅ **Date de naissance intégrée** directement
* ✅ **Validation automatique âge**
* ✅ **Compatible workflow 2 phases**
* ✅ **Optimisé pour recherches** BD

**📋 ACTIONS IMMÉDIATES :**

1. **Appliquer corrections PHP** dans OrganisationController.php
2. **Ajouter validation JavaScript** dans organisation-create.js
3. **Créer routes API** dans web.php
4. **Exécuter migration** base de données
5. **Tester format** dans formulaire création

**🎯 IMPACT SUR WORKFLOW 2 PHASES :**

* ✅ **Compatible** avec toutes les corrections précédentes
* ✅ **Améliore validation** Phase 1 et Phase 2
* ✅ **Détection anomalies** plus précise
* ✅ **Expérience utilisateur** optimisée

**Le format NIP XX-QQQQ-YYYYMMDD est maintenant intégré dans votre workflow 2 phases !** 🚀

**Status global : 75% + Ajustement NIP = Prêt pour finalisation complète !**

🔧 **CORRECTIONS À APPLIQUER**

<?php

/\*\*

\* 🔧 CORRECTIONS FORMAT NIP - XX-QQQQ-YYYYMMDD

\* Mise à jour des validations et méthodes pour le nouveau format

\*/

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

// 1. CORRECTION VALIDATION LARAVEL

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

/\*\*

\* Dans OrganisationController.php - Méthode validatePhase1DataCorrected()

\* Remplacer les règles de validation NIP

\*/

private function validatePhase1DataCorrected(Request $request, $type)

{

// ... code existant ...

$rules = [

// ÉTAPE 3 : Demandeur - FORMAT NIP CORRIGÉ

'demandeur\_nip' => [

'required',

'string',

'regex:/^[A-Z0-9]{2}-[0-9]{4}-[0-9]{8}$/',

function ($attribute, $value, $fail) {

if (!$this->validateNipFormat($value)) {

$fail('Le format du NIP est invalide. Format attendu: XX-QQQQ-YYYYMMDD');

}

}

],

// ... autres règles ...

// ÉTAPE 6 : Fondateurs - VALIDATION NIP CORRIGÉE

'fondateurs' => [

'required',

function ($attribute, $value, $fail) use ($type) {

// ... code décodage JSON existant ...

// Validation détaillée des fondateurs

foreach ($value as $index => $fondateur) {

if (!is\_array($fondateur)) {

$fail("Le fondateur ligne " . ($index + 1) . " doit être un objet valide.");

continue;

}

// VALIDATION NIP FORMAT CORRIGÉ

if (empty($fondateur['nip'])) {

$fail("Le NIP du fondateur ligne " . ($index + 1) . " ne peut pas être vide.");

} elseif (!$this->validateNipFormat($fondateur['nip'])) {

$fail("Le NIP du fondateur ligne " . ($index + 1) . " est invalide. Format: XX-QQQQ-YYYYMMDD");

}

// ... autres validations existantes ...

}

}

],

// ÉTAPE 7 : Adhérents - VALIDATION NIP CORRIGÉE (si présents)

'adherents' => [

'nullable',

function ($attribute, $value, $fail) use ($type) {

if (!empty($value)) {

// ... code décodage JSON existant ...

// Validation détaillée des adhérents

foreach ($value as $index => $adherent) {

if (!is\_array($adherent)) {

$fail("L'adhérent ligne " . ($index + 1) . " doit être un objet valide.");

continue;

}

// VALIDATION NIP FORMAT CORRIGÉ

if (empty($adherent['nip'])) {

$fail("Le NIP de l'adhérent ligne " . ($index + 1) . " ne peut pas être vide.");

} elseif (!$this->validateNipFormat($adherent['nip'])) {

$fail("Le NIP de l'adhérent ligne " . ($index + 1) . " est invalide. Format: XX-QQQQ-YYYYMMDD");

}

// ... autres validations existantes ...

}

}

}

]

];

$messages = [

'demandeur\_nip.required' => 'Le NIP du demandeur est obligatoire.',

'demandeur\_nip.regex' => 'Le NIP doit respecter le format XX-QQQQ-YYYYMMDD (ex: A1-2345-19901225).',

// ... autres messages ...

];

// ... reste du code existant ...

}

/\*\*

\* Dans OrganisationController.php - Méthode validateCompleteOrganisationData()

\* Mise à jour pour cohérence

\*/

private function validateCompleteOrganisationData(Request $request, $type)

{

// ... code existant ...

$rules = [

// ÉTAPE 3 : Demandeur - FORMAT NIP CORRIGÉ

'demandeur\_nip' => [

'required',

'string',

'regex:/^[A-Z0-9]{2}-[0-9]{4}-[0-9]{8}$/',

function ($attribute, $value, $fail) {

if (!$this->validateNipFormat($value)) {

$fail('Le format du NIP est invalide. Format attendu: XX-QQQQ-YYYYMMDD');

}

}

],

// ... mise à jour similaire pour fondateurs et adhérents ...

];

// ... reste du code existant ...

}

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

// 2. NOUVELLE MÉTHODE VALIDATION NIP

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

/\*\*

\* Valider le format du NIP selon le standard gabonais

\* Format: XX-QQQQ-YYYYMMDD

\*/

private function validateNipFormat($nip)

{

if (empty($nip)) {

return false;

}

// Vérification regex de base

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

return false;

}

// Extraction des parties

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

if (count($parts) !== 3) {

return false;

}

$prefix = $parts[0]; // XX (alphanumérique)

$sequence = $parts[1]; // QQQQ (4 chiffres)

$dateStr = $parts[2]; // YYYYMMDD (8 chiffres)

// Validation prefix XX (2 caractères alphanumériques)

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

return false;

}

// Validation sequence QQQQ (4 chiffres)

if (!preg\_match('/^[0-9]{4}$/', $sequence)) {

return false;

}

// Validation date YYYYMMDD

if (!preg\_match('/^[0-9]{8}$/', $dateStr)) {

return false;

}

// Extraction année, mois, jour

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

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

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

// Validation date réelle

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

return false;

}

// Validation plage d'années raisonnable (1900-2100)

if ($year < 1900 || $year > 2100) {

return false;

}

\Log::debug('NIP validé avec succès', [

'nip' => $nip,

'prefix' => $prefix,

'sequence' => $sequence,

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

]);

return true;

}

/\*\*

\* Extraire la date de naissance depuis le NIP

\*/

private function extractBirthDateFromNip($nip)

{

if (!$this->validateNipFormat($nip)) {

return null;

}

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

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

$year = substr($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;

}

}

/\*\*

\* Nettoyer et normaliser un NIP

\*/

private function cleanNipForStorage($nip)

{

if (empty($nip)) {

return '';

}

// Supprimer espaces et caractères indésirables

$cleaned = preg\_replace('/[^A-Z0-9\-]/', '', strtoupper($nip));

// Log du nettoyage pour traçabilité

if ($cleaned !== $nip) {

\Log::info('NIP nettoyé pour stockage', [

'original' => $nip,

'cleaned' => $cleaned

]);

}

return $cleaned;

}

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

// 3. CORRECTION DÉTECTION ANOMALIES

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

/\*\*

\* Mise à jour détection anomalies NIP

\*/

private function detectAndManageNipAnomalies(array $adherentData, string $typeOrganisation, int $organisationId)

{

$anomalies = [

'critiques' => [],

'majeures' => [],

'mineures' => []

];

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

$profession = $adherentData['profession'] ?? '';

// ✅ ANOMALIE : FORMAT NIP INCORRECT - NOUVEAU FORMAT

if (!$this->validateNipFormat($nip)) {

$anomalies['majeures'][] = [

'code' => 'NIP\_INVALID\_FORMAT',

'message' => 'Le NIP doit respecter le format XX-QQQQ-YYYYMMDD (ex: A1-2345-19901225).',

'nip\_fourni' => $nip,

'format\_attendu' => 'XX-QQQQ-YYYYMMDD'

];

} else {

// Si format correct, extraire la date de naissance pour validation additionnelle

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

if ($birthDate) {

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

// Validation âge raisonnable (18-100 ans)

if ($age < 18) {

$anomalies['critiques'][] = [

'code' => 'AGE\_TOO\_YOUNG',

'message' => 'Personne mineure détectée (âge: ' . $age . ' ans).',

'nip' => $nip,

'age\_calcule' => $age

];

} elseif ($age > 100) {

$anomalies['majeures'][] = [

'code' => 'AGE\_SUSPICIOUS',

'message' => 'Âge suspect détecté (âge: ' . $age . ' ans).',

'nip' => $nip,

'age\_calcule' => $age

];

}

}

}

// ✅ ANOMALIE : NIP DÉJÀ DANS UN AUTRE PARTI POLITIQUE

if ($typeOrganisation === 'parti\_politique') {

$existingInOtherParty = \App\Models\Adherent::whereHas('organisation', function($query) use ($organisationId) {

$query->where('type', 'parti\_politique')

->where('id', '!=', $organisationId);

})->where('nip', $nip)->exists();

if ($existingInOtherParty) {

$anomalies['critiques'][] = [

'code' => 'NIP\_DUPLICATE\_OTHER\_PARTY',

'message' => 'Ce NIP appartient déjà à un autre parti politique.',

'nip' => $nip

];

}

}

// ✅ ANOMALIE CRITIQUE : PROFESSION EXCLUE POUR PARTI POLITIQUE

if ($typeOrganisation === 'parti\_politique' && !empty($profession)) {

$professionsExclues = $this->getProfessionsExcluesParti();

if (in\_array(strtolower($profession), array\_map('strtolower', $professionsExclues))) {

$anomalies['critiques'][] = [

'code' => 'PROFESSION\_EXCLUE\_PARTI',

'message' => 'Profession exclue pour les partis politiques: ' . $profession,

'profession\_fournie' => $profession,

'type\_organisation' => $typeOrganisation,

'regle\_legale' => 'Article 15 - Loi N° 016/2025'

];

}

}

// ✅ ANOMALIE : DOUBLON DANS LA MÊME ORGANISATION

$existingInSameOrg = \App\Models\Adherent::where('organisation\_id', $organisationId)

->where('nip', $nip)

->exists();

if ($existingInSameOrg) {

$anomalies['majeures'][] = [

'code' => 'NIP\_DUPLICATE\_SAME\_ORG',

'message' => 'Ce NIP apparaît plusieurs fois dans cette organisation.',

'nip' => $nip

];

}

// ✅ ANOMALIE MINEURE : INFORMATIONS DE CONTACT MANQUANTES

if (empty($adherentData['telephone']) && empty($adherentData['email'])) {

$anomalies['mineures'][] = [

'code' => 'CONTACT\_INCOMPLET',

'message' => 'Aucun moyen de contact fourni (téléphone ou email).',

'telephone' => $adherentData['telephone'] ?? null,

'email' => $adherentData['email'] ?? null

];

}

return $anomalies;

}

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

// 4. VALIDATION API TEMPS RÉEL

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

/\*\*

\* NOUVELLE ROUTE API : Validation NIP en temps réel

\* POST /api/v1/validate-nip

\*/

public function validateNipApi(Request $request)

{

$request->validate([

'nip' => 'required|string'

]);

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

$isValid = $this->validateNipFormat($nip);

$response = [

'success' => true,

'valid' => $isValid,

'nip' => $nip

];

if ($isValid) {

// Extraire informations du NIP

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

if ($birthDate) {

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

$response['age'] = $birthDate->diffInYears(now());

}

// Vérifier s'il existe déjà

$exists = \App\Models\User::where('nip', $nip)->exists() ||

\App\Models\Adherent::where('nip', $nip)->exists() ||

\App\Models\Fondateur::where('nip', $nip)->exists();

$response['available'] = !$exists;

$response['message'] = $exists ? 'Ce NIP est déjà utilisé' : 'NIP disponible';

} else {

$response['message'] = 'Format NIP invalide. Format attendu: XX-QQQQ-YYYYMMDD';

$response['example'] = 'A1-2345-19901225';

}

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

}

🎯 **CORRECTIONS JAVASCRIPT FRONTEND**

/\*\*

\* 🔧 VALIDATION NIP CÔTÉ CLIENT - Format XX-QQQQ-YYYYMMDD

\* À ajouter dans organisation-create.js ou fichier dédié

\*/

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

// 1. VALIDATION NIP JAVASCRIPT

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

/\*\*

\* Valider le format NIP côté client

\*/

function validateNipFormat(nip) {

if (!nip || typeof nip !== 'string') {

return {

valid: false,

message: 'NIP requis'

};

}

// Nettoyer le NIP (supprimer espaces)

nip = nip.trim().toUpperCase();

// Vérification regex de base

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

if (!nipRegex.test(nip)) {

return {

valid: false,

message: 'Format invalide. Attendu: XX-QQQQ-YYYYMMDD (ex: A1-2345-19901225)',

example: 'A1-2345-19901225'

};

}

// Extraction des parties

const parts = nip.split('-');

const prefix = parts[0]; // XX

const sequence = parts[1]; // QQQQ

const dateStr = parts[2]; // YYYYMMDD

// Validation date

const year = parseInt(dateStr.substring(0, 4));

const month = parseInt(dateStr.substring(4, 6));

const day = parseInt(dateStr.substring(6, 8));

// Vérifier si la date est valide

const date = new Date(year, month - 1, day);

if (date.getFullYear() !== year ||

date.getMonth() !== month - 1 ||

date.getDate() !== day) {

return {

valid: false,

message: 'Date de naissance invalide dans le NIP',

extracted\_date: `${year}-${month.toString().padStart(2, '0')}-${day.toString().padStart(2, '0')}`

};

}

// Validation plage d'années

if (year < 1900 || year > 2100) {

return {

valid: false,

message: 'Année de naissance invalide (doit être entre 1900 et 2100)'

};

}

// Calculer l'âge

const today = new Date();

const birthDate = new Date(year, month - 1, day);

const age = Math.floor((today - birthDate) / (365.25 \* 24 \* 60 \* 60 \* 1000));

// Validation âge

if (age < 18) {

return {

valid: false,

message: `Personne mineure détectée (âge: ${age} ans). Seules les personnes majeures sont autorisées.`,

age: age

};

}

if (age > 100) {

return {

valid: true,

message: `Âge suspect détecté (${age} ans). Veuillez vérifier.`,

warning: true,

age: age

};

}

return {

valid: true,

message: 'NIP valide',

extracted\_info: {

prefix: prefix,

sequence: sequence,

birth\_date: `${year}-${month.toString().padStart(2, '0')}-${day.toString().padStart(2, '0')}`,

age: age

}

};

}

/\*\*

\* Formater automatiquement le NIP pendant la saisie

\*/

function formatNipInput(input) {

// Supprimer tout ce qui n'est pas alphanumérique

let value = input.toUpperCase().replace(/[^A-Z0-9]/g, '');

// Appliquer le format XX-QQQQ-YYYYMMDD

if (value.length > 2) {

value = value.substring(0, 2) + '-' + value.substring(2);

}

if (value.length > 7) {

value = value.substring(0, 7) + '-' + value.substring(7);

}

if (value.length > 16) {

value = value.substring(0, 16);

}

return value;

}

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

// 2. INTEGRATION DANS LE FORMULAIRE

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

/\*\*

\* Initialiser la validation NIP sur les champs

\*/

function initNipValidation() {

const nipInputs = document.querySelectorAll('input[data-validate="nip"], input[name\*="nip"]');

nipInputs.forEach(input => {

// Formatage automatique pendant la saisie

input.addEventListener('input', function(e) {

const formatted = formatNipInput(e.target.value);

e.target.value = formatted;

// Validation en temps réel

if (formatted.length === 16) { // XX-QQQQ-YYYYMMDD = 16 caractères

validateNipRealTime(e.target, formatted);

} else {

clearValidationFeedback(e.target);

}

});

// Validation finale à la perte de focus

input.addEventListener('blur', function(e) {

validateNipRealTime(e.target, e.target.value);

});

// Placeholder et aide

input.setAttribute('placeholder', 'A1-2345-19901225');

input.setAttribute('maxlength', '16');

input.setAttribute('pattern', '[A-Z0-9]{2}-[0-9]{4}-[0-9]{8}');

// Ajouter aide contextuelle

addNipHelp(input);

});

}

/\*\*

\* Validation temps réel avec feedback visuel

\*/

function validateNipRealTime(input, nip) {

const validation = validateNipFormat(nip);

const container = input.closest('.form-group, .mb-3, .input-group') || input.parentElement;

// Nettoyer ancien feedback

clearValidationFeedback(input);

if (validation.valid) {

input.classList.remove('is-invalid');

input.classList.add('is-valid');

if (validation.warning) {

showValidationMessage(container, validation.message, 'warning');

} else {

showValidationMessage(container, validation.message +

` (Âge: ${validation.extracted\_info.age} ans)`, 'success');

}

} else {

input.classList.remove('is-valid');

input.classList.add('is-invalid');

showValidationMessage(container, validation.message, 'error');

}

}

/\*\*

\* Afficher message de validation

\*/

function showValidationMessage(container, message, type) {

const existingFeedback = container.querySelector('.nip-validation-feedback');

if (existingFeedback) {

existingFeedback.remove();

}

const feedback = document.createElement('div');

feedback.className = `nip-validation-feedback small mt-1`;

feedback.textContent = message;

switch (type) {

case 'success':

feedback.className += ' text-success';

feedback.innerHTML = '<i class="fas fa-check-circle me-1"></i>' + message;

break;

case 'warning':

feedback.className += ' text-warning';

feedback.innerHTML = '<i class="fas fa-exclamation-triangle me-1"></i>' + message;

break;

case 'error':

feedback.className += ' text-danger';

feedback.innerHTML = '<i class="fas fa-times-circle me-1"></i>' + message;

break;

}

container.appendChild(feedback);

}

/\*\*

\* Nettoyer feedback de validation

\*/

function clearValidationFeedback(input) {

input.classList.remove('is-valid', 'is-invalid');

const container = input.closest('.form-group, .mb-3, .input-group') || input.parentElement;

const feedback = container.querySelector('.nip-validation-feedback');

if (feedback) {

feedback.remove();

}

}

/\*\*

\* Ajouter aide contextuelle pour le NIP

\*/

function addNipHelp(input) {

const container = input.closest('.form-group, .mb-3') || input.parentElement;

if (!container.querySelector('.nip-help')) {

const help = document.createElement('small');

help.className = 'nip-help text-muted';

help.innerHTML = `

<i class="fas fa-info-circle me-1"></i>

Format: <strong>XX-QQQQ-YYYYMMDD</strong>

<br>• XX = 2 caractères (lettres/chiffres)

<br>• QQQQ = 4 chiffres

<br>• YYYYMMDD = Date de naissance

`;

container.appendChild(help);

}

}

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

// 3. VALIDATION AJAX AVEC SERVEUR

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

/\*\*

\* Validation NIP avec le serveur (vérifier doublons)

\*/

async function validateNipWithServer(nip) {

try {

const response = await fetch('/api/v1/validate-nip', {

method: 'POST',

headers: {

'Content-Type': 'application/json',

'X-CSRF-TOKEN': document.querySelector('meta[name="csrf-token"]').content

},

body: JSON.stringify({ nip: nip })

});

const result = await response.json();

return result;

} catch (error) {

console.error('Erreur validation NIP serveur:', error);

return {

success: false,

valid: false,

message: 'Erreur de connexion au serveur'

};

}

}

/\*\*

\* Validation complète NIP (client + serveur)

\*/

async function validateNipComplete(input, nip) {

// 1. Validation côté client

const clientValidation = validateNipFormat(nip);

if (!clientValidation.valid) {

validateNipRealTime(input, nip);

return clientValidation;

}

// 2. Validation côté serveur

const container = input.closest('.form-group, .mb-3, .input-group') || input.parentElement;

showValidationMessage(container, 'Vérification en cours...', 'info');

const serverValidation = await validateNipWithServer(nip);

if (serverValidation.success && serverValidation.valid) {

input.classList.remove('is-invalid');

input.classList.add('is-valid');

const message = serverValidation.available ?

`NIP valide et disponible (Âge: ${serverValidation.age} ans)` :

'NIP valide mais déjà utilisé';

showValidationMessage(container, message,

serverValidation.available ? 'success' : 'warning');

} else {

input.classList.remove('is-valid');

input.classList.add('is-invalid');

showValidationMessage(container, serverValidation.message, 'error');

}

return serverValidation;

}

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

// 4. INITIALISATION

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

// Initialiser au chargement de la page

document.addEventListener('DOMContentLoaded', function() {

initNipValidation();

console.log('✅ Validation NIP initialisée - Format: XX-QQQQ-YYYYMMDD');

});

// Réinitialiser lors de l'ajout dynamique de champs

function reinitNipValidation() {

initNipValidation();

}

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

// 5. EXEMPLE D'USAGE DANS LE FORMULAIRE

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

/\*

HTML exemple:

<div class="mb-3">

<label for="demandeur\_nip" class="form-label">

NIP du demandeur <span class="text-danger">\*</span>

</label>

<input type="text"

class="form-control"

id="demandeur\_nip"

name="demandeur\_nip"

data-validate="nip"

placeholder="A1-2345-19901225"

required>

</div>

JavaScript pour validation complète:

document.getElementById('demandeur\_nip').addEventListener('blur', async function(e) {

if (e.target.value.length === 16) {

await validateNipComplete(e.target, e.target.value);

}

});

\*/

📋 **MISE À JOUR DES ROUTES API**

<?php

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

// ROUTES API VALIDATION NIP - À AJOUTER DANS web.php

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

// Dans la section Route::prefix('api/v1')->name('api.')->middleware(['auth', 'throttle:60,1'])->group(function () {

/\*\*

\* Validation NIP en temps réel

\* POST /api/v1/validate-nip

\*/

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

$request->validate([

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

]);

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

// Utiliser la méthode du contrôleur pour validation

$controller = app(\App\Http\Controllers\Operator\OrganisationController::class);

$isValid = $controller->validateNipFormat($nip);

$response = [

'success' => true,

'valid' => $isValid,

'nip' => $nip,

'format\_expected' => 'XX-QQQQ-YYYYMMDD'

];

if ($isValid) {

// Extraire informations du NIP

$birthDate = $controller->extractBirthDateFromNip($nip);

if ($birthDate) {

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

$response['age'] = $birthDate->diffInYears(now());

// Validation âge

if ($response['age'] < 18) {

$response['valid'] = false;

$response['message'] = 'Personne mineure détectée (âge: ' . $response['age'] . ' ans)';

$response['error\_code'] = 'UNDERAGE';

} elseif ($response['age'] > 100) {

$response['warning'] = true;

$response['message'] = 'Âge suspect détecté (' . $response['age'] . ' ans)';

}

}

// Vérifier si le NIP existe déjà

if ($response['valid']) {

$exists = \App\Models\User::where('nip', $nip)->exists() ||

\App\Models\Adherent::where('nip', $nip)->exists() ||

\App\Models\Fondateur::where('nip', $nip)->exists();

$response['available'] = !$exists;

if ($exists) {

// Trouver où le NIP est utilisé

$usage = [];

if (\App\Models\User::where('nip', $nip)->exists()) {

$usage[] = 'utilisateur';

}

if (\App\Models\Adherent::where('nip', $nip)->exists()) {

$usage[] = 'adhérent';

}

if (\App\Models\Fondateur::where('nip', $nip)->exists()) {

$usage[] = 'fondateur';

}

$response['message'] = 'NIP déjà utilisé comme: ' . implode(', ', $usage);

$response['usage'] = $usage;

} else {

$response['message'] = 'NIP valide et disponible';

}

}

} else {

$response['message'] = 'Format NIP invalide. Format attendu: XX-QQQQ-YYYYMMDD';

$response['example'] = 'A1-2345-19901225';

$response['help'] = [

'XX = 2 caractères alphanumériques (A-Z, 0-9)',

'QQQQ = 4 chiffres (0000-9999)',

'YYYYMMDD = Date de naissance (ex: 19901225 pour 25/12/1990)'

];

}

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

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

/\*\*

\* Générer exemple de NIP valide

\* GET /api/v1/generate-nip-example

\*/

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

// 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)'

]

]);

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

/\*\*

\* Validation de lot de NIP

\* POST /api/v1/validate-nip-batch

\*/

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

$request->validate([

'nips' => 'required|array|max:100',

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

]);

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

$controller = app(\App\Http\Controllers\Operator\OrganisationController::class);

$results = [];

foreach ($nips as $index => $nip) {

$isValid = $controller->validateNipFormat($nip);

$result = [

'index' => $index,

'nip' => $nip,

'valid' => $isValid

];

if ($isValid) {

$birthDate = $controller->extractBirthDateFromNip($nip);

if ($birthDate) {

$result['age'] = $birthDate->diffInYears(now());

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

}

// Vérifier existence

$exists = \App\Models\User::where('nip', $nip)->exists() ||

\App\Models\Adherent::where('nip', $nip)->exists() ||

\App\Models\Fondateur::where('nip', $nip)->exists();

$result['available'] = !$exists;

} else {

$result['message'] = 'Format invalide';

}

$results[] = $result;

}

// Statistiques

$stats = [

'total' => count($results),

'valid' => count(array\_filter($results, fn($r) => $r['valid'])),

'invalid' => count(array\_filter($results, fn($r) => !$r['valid'])),

'available' => count(array\_filter($results, fn($r) => ($r['valid'] ?? false) && ($r['available'] ?? false))),

'duplicates' => count(array\_filter($results, fn($r) => ($r['valid'] ?? false) && !($r['available'] ?? true)))

];

return response()->json([

'success' => true,

'results' => $results,

'statistics' => $stats

]);

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

📋 **MODIFICATIONS BASE DE DONNÉES**

<?php

/\*\*

\* MIGRATION BASE DE DONNÉES - Format NIP XX-QQQQ-YYYYMMDD

\* Fichier: database/migrations/xxxx\_xx\_xx\_update\_nip\_format.php

\*/

use Illuminate\Database\Migrations\Migration;

use Illuminate\Database\Schema\Blueprint;

use Illuminate\Support\Facades\Schema;

use Illuminate\Support\Facades\DB;

class UpdateNipFormat extends Migration

{

/\*\*

\* Run the migrations.

\*

\* @return void

\*/

public function up()

{

// 1. Modifier la contrainte sur la colonne NIP dans les tables

$this->updateNipConstraints();

// 2. Ajouter des index pour optimiser les recherches

$this->addNipIndexes();

// 3. Migrer les données existantes si nécessaire

$this->migrateExistingNipData();

}

/\*\*

\* Reverse the migrations.

\*

\* @return void

\*/

public function down()

{

// Restaurer les contraintes précédentes

$this->revertNipConstraints();

// Supprimer les index ajoutés

$this->removeNipIndexes();

}

/\*\*

\* Mettre à jour les contraintes NIP

\*/

private function updateNipConstraints()

{

// Table users

Schema::table('users', function (Blueprint $table) {

// Modifier la longueur pour supporter XX-QQQQ-YYYYMMDD (16 caractères)

$table->string('nip', 20)->change();

// Ajouter commentaire pour documentation

DB::statement("ALTER TABLE users MODIFY COLUMN nip VARCHAR(20) COMMENT 'NIP au format XX-QQQQ-YYYYMMDD'");

});

// Table adherents

Schema::table('adherents', function (Blueprint $table) {

$table->string('nip', 20)->change();

DB::statement("ALTER TABLE adherents MODIFY COLUMN nip VARCHAR(20) COMMENT 'NIP au format XX-QQQQ-YYYYMMDD'");

});

// Table fondateurs

Schema::table('fondateurs', function (Blueprint $table) {

$table->string('nip', 20)->change();

DB::statement("ALTER TABLE fondateurs MODIFY COLUMN nip VARCHAR(20) COMMENT 'NIP au format XX-QQQQ-YYYYMMDD'");

});

echo "✅ Contraintes NIP mises à jour pour format XX-QQQQ-YYYYMMDD\n";

}

/\*\*

\* Ajouter des index pour optimiser les recherches

\*/

private function addNipIndexes()

{

// Index sur les préfixes NIP (2 premiers caractères)

DB::statement("CREATE INDEX idx\_users\_nip\_prefix ON users ((SUBSTRING(nip, 1, 2)))");

DB::statement("CREATE INDEX idx\_adherents\_nip\_prefix ON adherents ((SUBSTRING(nip, 1, 2)))");

DB::statement("CREATE INDEX idx\_fondateurs\_nip\_prefix ON fondateurs ((SUBSTRING(nip, 1, 2)))");

// Index sur les dates de naissance extraites du NIP

DB::statement("CREATE INDEX idx\_users\_nip\_birthdate ON users ((SUBSTRING(nip, 8, 8)))");

DB::statement("CREATE INDEX idx\_adherents\_nip\_birthdate ON adherents ((SUBSTRING(nip, 8, 8)))");

DB::statement("CREATE INDEX idx\_fondateurs\_nip\_birthdate ON fondateurs ((SUBSTRING(nip, 8, 8)))");

echo "✅ Index NIP ajoutés pour optimisation\n";

}

/\*\*

\* Migrer les données existantes (si applicable)

\*/

private function migrateExistingNipData()

{

echo "ℹ️ Migration des données NIP existantes...\n";

// Compter les NIP existants

$usersCount = DB::table('users')->whereNotNull('nip')->count();

$adherentsCount = DB::table('adherents')->whereNotNull('nip')->count();

$fondateursCount = DB::table('fondateurs')->whereNotNull('nip')->count();

echo "📊 NIP existants trouvés:\n";

echo " - Users: $usersCount\n";

echo " - Adherents: $adherentsCount\n";

echo " - Fondateurs: $fondateursCount\n";

// Si des données existent, proposer une migration

if ($usersCount > 0 || $adherentsCount > 0 || $fondateursCount > 0) {

echo "⚠️ Des NIP existants ont été détectés.\n";

echo "ℹ️ Vous devrez mettre à jour manuellement les NIP existants pour respecter le nouveau format XX-QQQQ-YYYYMMDD\n";

// Créer une table temporaire pour stocker les anciens NIP

Schema::create('nip\_migration\_backup', function (Blueprint $table) {

$table->id();

$table->string('table\_name');

$table->unsignedBigInteger('record\_id');

$table->string('old\_nip');

$table->string('new\_nip')->nullable();

$table->timestamp('migrated\_at')->nullable();

$table->timestamps();

$table->index(['table\_name', 'record\_id']);

});

// Sauvegarder les anciens NIP

$this->backupExistingNips();

echo "✅ Sauvegarde des anciens NIP créée dans la table 'nip\_migration\_backup'\n";

}

}

/\*\*

\* Sauvegarder les anciens NIP

\*/

private function backupExistingNips()

{

// Sauvegarder users

DB::statement("

INSERT INTO nip\_migration\_backup (table\_name, record\_id, old\_nip, created\_at, updated\_at)

SELECT 'users', id, nip, NOW(), NOW()

FROM users

WHERE nip IS NOT NULL AND nip != ''

");

// Sauvegarder adherents

DB::statement("

INSERT INTO nip\_migration\_backup (table\_name, record\_id, old\_nip, created\_at, updated\_at)

SELECT 'adherents', id, nip, NOW(), NOW()

FROM adherents

WHERE nip IS NOT NULL AND nip != ''

");

// Sauvegarder fondateurs

DB::statement("

INSERT INTO nip\_migration\_backup (table\_name, record\_id, old\_nip, created\_at, updated\_at)

SELECT 'fondateurs', id, nip, NOW(), NOW()

FROM fondateurs

WHERE nip IS NOT NULL AND nip != ''

");

}

/\*\*

\* Restaurer les contraintes précédentes

\*/

private function revertNipConstraints()

{

// Restaurer la longueur précédente (13 caractères)

Schema::table('users', function (Blueprint $table) {

$table->string('nip', 13)->change();

});

Schema::table('adherents', function (Blueprint $table) {

$table->string('nip', 13)->change();

});

Schema::table('fondateurs', function (Blueprint $table) {

$table->string('nip', 13)->change();

});

}

/\*\*

\* Supprimer les index ajoutés

\*/

private function removeNipIndexes()

{

DB::statement("DROP INDEX IF EXISTS idx\_users\_nip\_prefix");

DB::statement("DROP INDEX IF EXISTS idx\_adherents\_nip\_prefix");

DB::statement("DROP INDEX IF EXISTS idx\_fondateurs\_nip\_prefix");

DB::statement("DROP INDEX IF EXISTS idx\_users\_nip\_birthdate");

DB::statement("DROP INDEX IF EXISTS idx\_adherents\_nip\_birthdate");

DB::statement("DROP INDEX IF EXISTS idx\_fondateurs\_nip\_birthdate");

}

}

/\*\*

\* COMMANDE ARTISAN POUR MIGRATION NIP

\* Fichier: app/Console/Commands/MigrateNipFormat.php

\*/

namespace App\Console\Commands;

use Illuminate\Console\Command;

use Illuminate\Support\Facades\DB;

class MigrateNipFormat extends Command

{

protected $signature = 'nip:migrate-format {--dry-run : Afficher les modifications sans les appliquer}';

protected $description = 'Migrer les NIP existants vers le nouveau format XX-QQQQ-YYYYMMDD';

public function handle()

{

$this->info('🔄 Migration format NIP vers XX-QQQQ-YYYYMMDD');

$isDryRun = $this->option('dry-run');

if ($isDryRun) {

$this->warn('Mode DRY-RUN activé - Aucune modification ne sera appliquée');

}

// Récupérer les NIP à migrer

$nipToMigrate = DB::table('nip\_migration\_backup')

->whereNull('migrated\_at')

->get();

if ($nipToMigrate->isEmpty()) {

$this->info('✅ Aucun NIP à migrer');

return;

}

$this->info("📊 {$nipToMigrate->count()} NIP à migrer");

$migrated = 0;

$errors = 0;

foreach ($nipToMigrate as $record) {

try {

$newNip = $this->convertToNewFormat($record->old\_nip);

if ($newNip) {

$this->line("🔄 {$record->table\_name}#{$record->record\_id}: {$record->old\_nip} → {$newNip}");

if (!$isDryRun) {

// Mettre à jour le NIP dans la table source

DB::table($record->table\_name)

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

->update(['nip' => $newNip]);

// Marquer comme migré

DB::table('nip\_migration\_backup')

->where('id', $record->id)

->update([

'new\_nip' => $newNip,

'migrated\_at' => now()

]);

}

$migrated++;

} else {

$this->error("❌ Impossible de convertir: {$record->old\_nip}");

$errors++;

}

} catch (\Exception $e) {

$this->error("❌ Erreur pour {$record->old\_nip}: " . $e->getMessage());

$errors++;

}

}

$this->info("✅ Migration terminée:");

$this->info(" - Migrés: $migrated");

$this->info(" - Erreurs: $errors");

if ($isDryRun) {

$this->warn('Exécutez sans --dry-run pour appliquer les modifications');

}

}

/\*\*

\* Convertir ancien format vers nouveau format

\*/

private function convertToNewFormat($oldNip)

{

// Exemple de conversion - à adapter selon votre ancien format

// Si ancien format était 13 chiffres: 1234567890123

if (preg\_match('/^[0-9]{13}$/', $oldNip)) {

// Extraire année de naissance (supposons positions 7-10)

$year = substr($oldNip, 7, 4);

// Extraire mois/jour (supposons positions 11-14)

$month = substr($oldNip, 11, 2);

$day = substr($oldNip, 13, 2);

// Générer nouveau format

$prefix = substr($oldNip, 0, 2); // Premiers 2 chiffres → lettres

$sequence = substr($oldNip, 2, 4); // Positions 3-6

return strtoupper(dechex($prefix)) . '-' . $sequence . '-' . $year . $month . $day;

}

return null; // Conversion impossible

}

}