/\*\*

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

\* PNGDI - Formulaire Création Organisation - VERSION FINALE COMPLÈTE

\* Fichier: public/js/organisation-create.js

\* Compatible: Bootstrap 5 + Laravel + Toutes les 9 étapes

\* Date: 29 juin 2025

\* Version: 1.2 avec système d'anomalies intégré

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

\*/

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

// FONCTION DE COMPATIBILITÉ NAVIGATEURS

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

function elementMatches(element, selector) {

if (!element) return false;

if (element.matches) {

return element.matches(selector);

} else if (element.msMatchesSelector) {

return element.msMatchesSelector(selector);

} else if (element.webkitMatchesSelector) {

return element.webkitMatchesSelector(selector);

} else if (element.mozMatchesSelector) {

return element.mozMatchesSelector(selector);

}

// Fallback pour très anciens navigateurs

return false;

}

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

// 1. CONFIGURATION GLOBALE

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

window.OrganisationApp = {

// État actuel

currentStep: 1,

totalSteps: 9,

selectedOrgType: '',

// Configuration

config: {

autoSaveInterval: 30000, // 30 secondes

validationDelay: 500, // 500ms pour debounce

animationDuration: 300, // 300ms pour animations

// Configuration NIP corrigée (sans checksum strict)

nip: {

length: 13,

pattern: /^[0-9]{13}$/,

strictValidation: false, // Désactivé pour éviter les erreurs

allowTestValues: true

},

// Configuration téléphone gabonais

phone: {

prefixes: ['01', '02', '03', '04', '05', '06', '07'],

minLength: 8,

maxLength: 9,

pattern: /^[0-9]{8,9}$/

},

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

// NOUVEAU : Configuration système anomalies

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

anomalies: {

enabled: true,

types: {

'nip\_invalide': {

level: 'critique',

label: 'NIP invalide ou incorrect',

description: 'Le numéro NIP ne respecte pas le format gabonais standard'

},

'telephone\_invalide': {

level: 'majeure',

label: 'Numéro de téléphone invalide',

description: 'Le numéro de téléphone ne respecte pas le format gabonais'

},

'email\_invalide': {

level: 'majeure',

label: 'Adresse email invalide',

description: 'Le format de l\'adresse email est incorrect'

},

'champs\_incomplets': {

level: 'majeure',

label: 'Informations incomplètes',

description: 'Des champs obligatoires sont manquants'

},

'membre\_existant': {

level: 'critique',

label: 'Membre déjà enregistré ailleurs',

description: 'Cette personne est déjà membre active d\'une autre organisation'

},

'profession\_exclue\_parti': {

level: 'critique',

label: 'Profession exclue pour parti politique',

description: 'Cette profession est interdite pour les membres de partis politiques'

},

'doublon\_fichier': {

level: 'mineure',

label: 'Doublon dans le fichier',

description: 'Ce NIP apparaît plusieurs fois dans le fichier importé'

},

'format\_donnees': {

level: 'mineure',

label: 'Format de données suspect',

description: 'Les données semblent présenter des incohérences de format'

}

},

// Niveaux de gravité

levels: {

'critique': {

priority: 3,

color: 'danger',

icon: 'fa-exclamation-triangle',

badge: 'bg-danger'

},

'majeure': {

priority: 2,

color: 'warning',

icon: 'fa-exclamation-circle',

badge: 'bg-warning'

},

'mineure': {

priority: 1,

color: 'info',

icon: 'fa-info-circle',

badge: 'bg-info'

}

}

},

// Exigences par type d'organisation

orgRequirements: {

'association': {

minFondateurs: 3,

minAdherents: 10,

label: 'Association',

documents: ['statuts', 'pv\_ag', 'liste\_fondateurs', 'justif\_siege']

},

'ong': {

minFondateurs: 5,

minAdherents: 15,

label: 'ONG',

documents: ['statuts', 'pv\_ag', 'liste\_fondateurs', 'justif\_siege', 'projet\_social', 'budget\_previsionnel']

},

'parti\_politique': {

minFondateurs: 3,

minAdherents: 50,

label: 'Parti Politique',

documents: ['statuts', 'pv\_ag', 'liste\_fondateurs', 'justif\_siege', 'programme\_politique', 'liste\_50\_adherents']

},

'confession\_religieuse': {

minFondateurs: 3,

minAdherents: 10,

label: 'Confession Religieuse',

documents: ['statuts', 'pv\_ag', 'liste\_fondateurs', 'justif\_siege', 'expose\_doctrine', 'justif\_lieu\_culte']

}

},

// Professions exclues pour partis politiques

professionsExcluesParti: [

'magistrat', 'juge', 'procureur', 'avocat\_general',

'militaire', 'gendarme', 'policier', 'forces\_armee',

'prefet', 'sous\_prefet', 'gouverneur', 'maire',

'fonctionnaire\_administration', 'ambassadeur', 'consul',

'directeur\_general\_public', 'recteur\_universite',

'chef\_etablissement\_public', 'membre\_conseil\_constitutionnel',

'controleur\_etat', 'inspecteur\_general',

'membre\_autorite\_independante', 'comptable\_public'

]

},

// Cache et données

cache: new Map(),

formData: {},

validationErrors: {},

fondateurs: [],

adherents: [],

documents: {},

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

// NOUVEAU : Système de gestion des anomalies

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

rapportAnomalies: {

enabled: false,

adherentsValides: 0,

adherentsAvecAnomalies: 0,

anomalies: [],

statistiques: {

critique: 0,

majeure: 0,

mineure: 0

},

genereAt: null,

version: '1.2'

},

// Timers

timers: {

autoSave: null,

validation: {}

}

};

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

// NOUVELLES FONCTIONS UTILITAIRES ANOMALIES

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

/\*\*

\* Créer une anomalie pour un adhérent

\*/

function createAnomalie(adherent, type, details = '') {

const anomalieConfig = OrganisationApp.config.anomalies.types[type];

if (!anomalieConfig) {

console.warn('Type d\'anomalie non reconnu:', type);

return null;

}

return {

id: `anomalie\_${Date.now()}\_${Math.random().toString(36).substr(2, 9)}`,

adherentId: adherent.id || adherent.nip,

adherentNom: `${adherent.nom} ${adherent.prenom}`,

adherentNip: adherent.nip,

adherentLigne: adherent.lineNumber || null,

type: type,

level: anomalieConfig.level,

label: anomalieConfig.label,

description: anomalieConfig.description,

details: details,

detecteAt: new Date().toISOString(),

status: 'detected'

};

}

/\*\*

\* Ajouter une anomalie au rapport

\*/

function addAnomalieToReport(anomalie) {

if (!anomalie) return;

// Activer le rapport s'il n'est pas déjà activé

if (!OrganisationApp.rapportAnomalies.enabled) {

OrganisationApp.rapportAnomalies.enabled = true;

OrganisationApp.rapportAnomalies.genereAt = new Date().toISOString();

}

// Ajouter l'anomalie

OrganisationApp.rapportAnomalies.anomalies.push(anomalie);

// Mettre à jour les statistiques

const level = anomalie.level;

if (OrganisationApp.rapportAnomalies.statistiques[level] !== undefined) {

OrganisationApp.rapportAnomalies.statistiques[level]++;

}

console.log(`📋 Anomalie ajoutée: ${anomalie.type} (${anomalie.level}) pour ${anomalie.adherentNom}`);

}

/\*\*

\* Obtenir le statut qualité global

\*/

function getQualiteStatut() {

const total = OrganisationApp.adherents.length;

const avecAnomalies = OrganisationApp.rapportAnomalies.adherentsAvecAnomalies;

const valides = total - avecAnomalies;

if (total === 0) return 'aucun';

if (avecAnomalies === 0) return 'excellent';

if (avecAnomalies < total \* 0.1) return 'bon';

if (avecAnomalies < total \* 0.3) return 'moyen';

return 'faible';

}

/\*\*

\* Obtenir les recommandations selon les anomalies

\*/

function getRecommandationsAnomalies() {

const anomalies = OrganisationApp.rapportAnomalies.anomalies;

const stats = OrganisationApp.rapportAnomalies.statistiques;

const recommandations = [];

if (stats.critique > 0) {

recommandations.push({

type: 'urgent',

message: `${stats.critique} anomalie(s) critique(s) détectée(s). Correction immédiate recommandée.`

});

}

if (stats.majeure > 0) {

recommandations.push({

type: 'important',

message: `${stats.majeure} anomalie(s) majeure(s) nécessitent votre attention.`

});

}

if (stats.mineure > 0) {

recommandations.push({

type: 'conseil',

message: `${stats.mineure} anomalie(s) mineure(s) à corriger pour optimiser la qualité.`

});

}

// Recommandations spécifiques selon les types d'anomalies

const typesDetectes = [...new Set(anomalies.map(a => a.type))];

if (typesDetectes.includes('nip\_invalide')) {

recommandations.push({

type: 'conseil',

message: 'Vérifiez les numéros NIP auprès des services d\'état civil.'

});

}

if (typesDetectes.includes('membre\_existant')) {

recommandations.push({

type: 'urgent',

message: 'Contactez les membres concernés pour régulariser leur situation.'

});

}

if (typesDetectes.includes('profession\_exclue\_parti')) {

recommandations.push({

type: 'urgent',

message: 'Les personnes avec professions exclues ne peuvent être membres de partis politiques.'

});

}

return recommandations;

}

/\*\*

\* Fonctions utilitaires pour l'affichage des anomalies

\*/

function getQualiteBadgeClass(qualite) {

const classes = {

'excellent': 'bg-success',

'bon': 'bg-info',

'moyen': 'bg-warning',

'faible': 'bg-danger'

};

return classes[qualite] || 'bg-secondary';

}

function getQualiteLabel(qualite) {

const labels = {

'excellent': 'Excellente qualité',

'bon': 'Bonne qualité',

'moyen': 'Qualité moyenne',

'faible': 'Qualité faible'

};

return labels[qualite] || 'Non évalué';

}

console.log('✅ Configuration globale avec anomalies - Version 1.2 harmonisée');

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

// 2. FONCTIONS DE NAVIGATION

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

/\*\*

\* Navigation entre les étapes

\*/

function changeStep(direction) {

console.log(`🔄 Changement d'étape: direction ${direction}, étape actuelle: ${OrganisationApp.currentStep}`);

// Validation avant d'avancer

if (direction === 1 && !validateCurrentStep()) {

console.log('❌ Validation échouée pour l\'étape', OrganisationApp.currentStep);

showNotification('Veuillez compléter tous les champs obligatoires avant de continuer', 'warning');

return false;

}

// Sauvegarder l'étape actuelle

saveCurrentStepData();

// Calculer la nouvelle étape

const newStep = OrganisationApp.currentStep + direction;

if (newStep >= 1 && newStep <= OrganisationApp.totalSteps) {

OrganisationApp.currentStep = newStep;

updateStepDisplay();

updateNavigationButtons();

// Actions spécifiques selon l'étape

handleStepSpecificActions(newStep);

scrollToTop();

return true;

}

return false;

}

/\*\*

\* Actions spécifiques selon l'étape

\*/

function handleStepSpecificActions(stepNumber) {

switch (stepNumber) {

case 2:

updateGuideContent();

break;

case 4:

updateOrganizationRequirements();

break;

case 6:

updateFoundersRequirements();

break;

case 7:

updateMembersRequirements();

break;

case 8:

updateDocumentsRequirements();

break;

case 9:

generateRecap();

break;

}

}

/\*\*

\* Aller directement à une étape

\*/

function goToStep(stepNumber) {

if (stepNumber >= 1 && stepNumber <= OrganisationApp.totalSteps) {

// Valider toutes les étapes jusqu'à celle-ci

for (let i = 1; i < stepNumber; i++) {

if (!validateStep(i)) {

showNotification(`Veuillez compléter l'étape ${i} avant de continuer`, 'warning');

return false;

}

}

OrganisationApp.currentStep = stepNumber;

updateStepDisplay();

updateNavigationButtons();

handleStepSpecificActions(stepNumber);

scrollToTop();

return true;

}

return false;

}

/\*\*

\* Mise à jour de l'affichage des étapes

\*/

function updateStepDisplay() {

// Masquer toutes les étapes

document.querySelectorAll('.step-content').forEach(content => {

content.style.display = 'none';

});

// Afficher l'étape actuelle avec animation

const currentStepElement = document.getElementById('step' + OrganisationApp.currentStep);

if (currentStepElement) {

currentStepElement.style.display = 'block';

// Animation d'entrée

currentStepElement.style.opacity = '0';

currentStepElement.style.transform = 'translateY(20px)';

setTimeout(() => {

currentStepElement.style.transition = 'all 0.3s ease';

currentStepElement.style.opacity = '1';

currentStepElement.style.transform = 'translateY(0)';

}, 10);

console.log('✅ Affichage étape', OrganisationApp.currentStep);

} else {

console.warn('⚠️ Élément step' + OrganisationApp.currentStep + ' non trouvé');

}

// Mettre à jour la barre de progression

updateProgressBar();

// Mettre à jour le numéro d'étape

const currentStepNumber = document.getElementById('currentStepNumber');

if (currentStepNumber) {

currentStepNumber.textContent = OrganisationApp.currentStep;

}

// Mettre à jour les indicateurs d'étapes

updateStepIndicators();

}

/\*\*

\* Mise à jour de la barre de progression

\*/

function updateProgressBar() {

const progress = (OrganisationApp.currentStep / OrganisationApp.totalSteps) \* 100;

const progressBar = document.getElementById('globalProgress');

if (progressBar) {

progressBar.style.width = progress + '%';

progressBar.setAttribute('aria-valuenow', progress);

// Animation de la barre

progressBar.classList.add('progress-bar-animated');

setTimeout(() => {

progressBar.classList.remove('progress-bar-animated');

}, 1000);

}

}

/\*\*

\* Mise à jour des indicateurs d'étapes

\*/

function updateStepIndicators() {

document.querySelectorAll('.step-indicator').forEach((indicator, index) => {

const stepNumber = index + 1;

// Retirer toutes les classes d'état

indicator.classList.remove('active', 'completed');

if (stepNumber === OrganisationApp.currentStep) {

indicator.classList.add('active');

// Animation pour l'étape active

indicator.style.transform = 'scale(1.05)';

setTimeout(() => {

indicator.style.transform = '';

}, 300);

} else if (stepNumber < OrganisationApp.currentStep) {

indicator.classList.add('completed');

}

// Ajouter un gestionnaire de clic

indicator.addEventListener('click', () => {

if (stepNumber <= OrganisationApp.currentStep || stepNumber === OrganisationApp.currentStep + 1) {

goToStep(stepNumber);

}

});

});

}

/\*\*

\* Mise à jour des boutons de navigation

\*/

function updateNavigationButtons() {

const prevBtn = document.getElementById('prevBtn');

const nextBtn = document.getElementById('nextBtn');

const submitBtn = document.getElementById('submitBtn');

// Bouton précédent

if (prevBtn) {

prevBtn.style.display = OrganisationApp.currentStep === 1 ? 'none' : 'inline-block';

}

// Boutons suivant et soumettre

if (OrganisationApp.currentStep === OrganisationApp.totalSteps) {

if (nextBtn) nextBtn.style.display = 'none';

if (submitBtn) submitBtn.classList.remove('d-none');

} else {

if (nextBtn) nextBtn.style.display = 'inline-block';

if (submitBtn) submitBtn.classList.add('d-none');

}

}

/\*\*

\* Scroll vers le haut avec animation

\*/

function scrollToTop() {

window.scrollTo({

top: 0,

behavior: 'smooth'

});

}

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

// 3. GESTION TYPE D'ORGANISATION

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

/\*\*

\* Sélection du type d'organisation

\*/

function selectOrganizationType(card) {

console.log('🏢 Sélection du type d\'organisation');

// Retirer la sélection précédente avec animation

document.querySelectorAll('.organization-type-card').forEach(c => {

c.classList.remove('active');

c.style.transform = '';

});

// Appliquer la nouvelle sélection avec animation

card.classList.add('active');

card.style.transform = 'scale(1.02)';

setTimeout(() => {

card.style.transform = '';

}, 300);

// Cocher le radio button et sauvegarder le type

const radio = card.querySelector('input[type="radio"]');

if (radio) {

radio.checked = true;

OrganisationApp.selectedOrgType = radio.value;

// Mettre à jour l'input caché

const hiddenInput = document.getElementById('organizationType');

if (hiddenInput) {

hiddenInput.value = radio.value;

}

// Sauvegarder dans les données du formulaire

OrganisationApp.formData.organizationType = radio.value;

}

// Afficher les informations de sélection

showSelectedTypeInfo(radio.value);

// Mettre à jour le guide de l'étape 2

updateGuideContent();

}

/\*\*

\* Affichage des informations du type sélectionné

\*/

function showSelectedTypeInfo(type) {

const selectedInfo = document.getElementById('selectedTypeInfo');

const selectedTypeName = document.getElementById('selectedTypeName');

if (selectedInfo && selectedTypeName) {

selectedTypeName.textContent = getOrganizationTypeLabel(type);

// Animation d'apparition

selectedInfo.style.opacity = '0';

selectedInfo.classList.remove('d-none');

setTimeout(() => {

selectedInfo.style.transition = 'opacity 0.3s ease';

selectedInfo.style.opacity = '1';

}, 10);

}

}

/\*\*

\* Obtenir le label d'un type d'organisation

\*/

function getOrganizationTypeLabel(type) {

const labels = {

'association': 'Association',

'ong': 'Organisation Non Gouvernementale (ONG)',

'parti\_politique': 'Parti Politique',

'confession\_religieuse': 'Confession Religieuse'

};

return labels[type] || type;

}

/\*\*

\* Mise à jour du contenu du guide selon le type

\*/

function updateGuideContent() {

const guideContent = document.getElementById('guide-content');

const selectedTypeTitle = document.getElementById('selectedTypeTitle');

if (!OrganisationApp.selectedOrgType) return;

if (selectedTypeTitle) {

selectedTypeTitle.textContent = getOrganizationTypeLabel(OrganisationApp.selectedOrgType);

}

if (guideContent) {

const content = getGuideContentForType(OrganisationApp.selectedOrgType);

guideContent.innerHTML = content;

}

}

/\*\*

\* Contenu du guide selon le type d'organisation

\*/

function getGuideContentForType(type) {

const guides = {

'association': `

<div class="alert alert-success border-0 mb-4 shadow-sm">

<div class="d-flex align-items-center">

<i class="fas fa-handshake fa-3x me-3 text-success"></i>

<div>

<h5 class="alert-heading mb-1">Guide pour créer une Association au Gabon</h5>

<p class="mb-0">Procédures légales selon la législation gabonaise en vigueur</p>

</div>

</div>

</div>

<div class="row">

<div class="col-md-6">

<h6 class="text-success"><i class="fas fa-check me-2"></i>Exigences minimales</h6>

<ul class="list-unstyled">

<li>• Minimum 3 membres fondateurs majeurs</li>

<li>• Minimum 10 adhérents à la création</li>

<li>• Siège social au Gabon</li>

<li>• But exclusivement non lucratif</li>

</ul>

</div>

<div class="col-md-6">

<h6 class="text-info"><i class="fas fa-file-alt me-2"></i>Documents requis</h6>

<ul class="list-unstyled">

<li>• Statuts signés et légalisés</li>

<li>• PV de l'assemblée constitutive</li>

<li>• Liste des fondateurs avec NIP</li>

<li>• Justificatif du siège social</li>

</ul>

</div>

</div>

`,

'ong': `

<div class="alert alert-info border-0 mb-4 shadow-sm">

<div class="d-flex align-items-center">

<i class="fas fa-globe-africa fa-3x me-3 text-info"></i>

<div>

<h5 class="alert-heading mb-1">Guide pour créer une ONG au Gabon</h5>

<p class="mb-0">Organisation Non Gouvernementale à vocation humanitaire</p>

</div>

</div>

</div>

<div class="row">

<div class="col-md-6">

<h6 class="text-success"><i class="fas fa-check me-2"></i>Exigences minimales</h6>

<ul class="list-unstyled">

<li>• Minimum 5 membres fondateurs majeurs</li>

<li>• Minimum 15 adhérents à la création</li>

<li>• Mission d'intérêt général</li>

<li>• Projet social déterminé</li>

</ul>

</div>

<div class="col-md-6">

<h6 class="text-info"><i class="fas fa-file-alt me-2"></i>Documents requis</h6>

<ul class="list-unstyled">

<li>• Statuts de l'ONG</li>

<li>• Plan d'action et budget prévisionnel</li>

<li>• CV des dirigeants</li>

<li>• Projet social détaillé</li>

</ul>

</div>

</div>

`,

'parti\_politique': `

<div class="alert alert-warning border-0 mb-4 shadow-sm">

<div class="d-flex align-items-center">

<i class="fas fa-vote-yea fa-3x me-3 text-warning"></i>

<div>

<h5 class="alert-heading mb-1">Guide pour créer un Parti Politique au Gabon</h5>

<p class="mb-0">Organisation politique pour participer à la vie démocratique</p>

</div>

</div>

</div>

<div class="row">

<div class="col-md-6">

<h6 class="text-success"><i class="fas fa-check me-2"></i>Exigences minimales</h6>

<ul class="list-unstyled">

<li>• Minimum 3 membres fondateurs majeurs</li>

<li>• <strong>Minimum 50 adhérents</strong> répartis sur 3 provinces</li>

<li>• Programme politique détaillé</li>

<li>• Vocation démocratique</li>

</ul>

</div>

<div class="col-md-6">

<h6 class="text-info"><i class="fas fa-file-alt me-2"></i>Documents requis</h6>

<ul class="list-unstyled">

<li>• Statuts du parti</li>

<li>• Programme politique</li>

<li>• Liste de 50 adhérents minimum</li>

<li>• Répartition géographique</li>

</ul>

</div>

</div>

<div class="alert alert-danger mt-3">

<strong>⚠️ Important :</strong> 22 professions sont exclues des partis politiques (magistrats, militaires, fonctionnaires, etc.)

</div>

`,

'confession\_religieuse': `

<div class="alert alert-secondary border-0 mb-4 shadow-sm" style="background: linear-gradient(135deg, rgba(111, 66, 193, 0.1) 0%, rgba(232, 62, 140, 0.05) 100%);">

<div class="d-flex align-items-center">

<i class="fas fa-pray fa-3x me-3 text-purple"></i>

<div>

<h5 class="alert-heading mb-1">Guide pour créer une Confession Religieuse au Gabon</h5>

<p class="mb-0">Organisation religieuse pour l'exercice du culte</p>

</div>

</div>

</div>

<div class="row">

<div class="col-md-6">

<h6 class="text-success"><i class="fas fa-check me-2"></i>Exigences minimales</h6>

<ul class="list-unstyled">

<li>• Minimum 3 membres fondateurs majeurs</li>

<li>• Minimum 10 fidèles à la création</li>

<li>• Doctrine religieuse clairement définie</li>

<li>• Lieu de culte identifié</li>

</ul>

</div>

<div class="col-md-6">

<h6 class="text-info"><i class="fas fa-file-alt me-2"></i>Documents requis</h6>

<ul class="list-unstyled">

<li>• Statuts de la confession</li>

<li>• Doctrine religieuse</li>

<li>• Liste des fidèles fondateurs</li>

<li>• Attestation du lieu de culte</li>

</ul>

</div>

</div>

`

};

return guides[type] || '<p>Guide non disponible pour ce type d\'organisation.</p>';

}

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

// 4. VALIDATION COMPLÈTE TOUTES ÉTAPES

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

/\*\*

\* Validation de l'étape actuelle

\*/

function validateCurrentStep() {

return validateStep(OrganisationApp.currentStep);

}

/\*\*

\* Validation d'une étape spécifique

\*/

function validateStep(stepNumber) {

switch (stepNumber) {

case 1: return validateStep1();

case 2: return validateStep2();

case 3: return validateStep3();

case 4: return validateStep4();

case 5: return validateStep5();

case 6: return validateStep6();

case 7: return validateStep7();

case 8: return validateStep8();

case 9: return validateStep9();

default: return true;

}

}

/\*\*

\* Validation étape 1 : Type d'organisation

\*/

function validateStep1() {

const selectedType = document.querySelector('input[name="type\_organisation"]:checked');

if (!selectedType) {

showFieldError(null, 'Veuillez sélectionner un type d\'organisation');

// Faire clignoter les cartes

document.querySelectorAll('.organization-type-card').forEach(card => {

card.style.animation = 'shake 0.5s ease-in-out';

setTimeout(() => {

card.style.animation = '';

}, 500);

});

return false;

}

return true;

}

/\*\*

\* Validation étape 2 : Guide lu

\*/

function validateStep2() {

const guideConfirm = document.getElementById('guideReadConfirm');

if (!guideConfirm || !guideConfirm.checked) {

showFieldError(guideConfirm, 'Veuillez confirmer avoir lu et compris le guide');

if (guideConfirm) {

guideConfirm.focus();

guideConfirm.parentElement.style.animation = 'shake 0.5s ease-in-out';

setTimeout(() => {

guideConfirm.parentElement.style.animation = '';

}, 500);

}

return false;

}

return true;

}

/\*\*

\* Validation étape 3 : Informations demandeur

\*/

function validateStep3() {

const requiredFields = [

'demandeur\_nip',

'demandeur\_civilite',

'demandeur\_nom',

'demandeur\_prenom',

'demandeur\_date\_naissance',

'demandeur\_nationalite',

'demandeur\_telephone',

'demandeur\_email',

'demandeur\_adresse',

'demandeur\_role'

];

let isValid = true;

let firstErrorField = null;

requiredFields.forEach(fieldId => {

const field = document.getElementById(fieldId);

if (field) {

const fieldValid = validateField(field);

if (!fieldValid && !firstErrorField) {

firstErrorField = field;

}

isValid = isValid && fieldValid;

}

});

// Vérifier les checkboxes d'engagement

const engagement = document.getElementById('demandeur\_engagement');

const responsabilite = document.getElementById('demandeur\_responsabilite');

if (!engagement || !engagement.checked) {

showFieldError(engagement, 'Veuillez cocher l\'engagement de véracité');

if (!firstErrorField) firstErrorField = engagement;

isValid = false;

}

if (!responsabilite || !responsabilite.checked) {

showFieldError(responsabilite, 'Veuillez accepter la responsabilité légale');

if (!firstErrorField) firstErrorField = responsabilite;

isValid = false;

}

// Scroll vers le premier champ en erreur

if (!isValid && firstErrorField) {

firstErrorField.scrollIntoView({ behavior: 'smooth', block: 'center' });

setTimeout(() => {

firstErrorField.focus();

}, 300);

}

return isValid;

}

/\*\*

\* Validation étape 4 : Informations organisation

\*/

function validateStep4() {

const requiredFields = [

'org\_nom', 'org\_objet', 'org\_date\_creation', 'org\_telephone'

];

let isValid = true;

requiredFields.forEach(fieldId => {

const field = document.getElementById(fieldId);

if (field && !validateField(field)) {

isValid = false;

}

});

// Validation spéciale pour org\_objet (minimum 50 caractères)

const orgObjet = document.getElementById('org\_objet');

if (orgObjet) {

const objetText = orgObjet.value.trim();

if (objetText.length < 50) {

showFieldError(orgObjet, `L'objet social doit contenir au moins 50 caractères (${objetText.length}/50)`);

isValid = false;

} else {

clearFieldError(orgObjet);

}

}

return isValid;

}

/\*\*

\* Validation étape 5 : Coordonnées

\*/

function validateStep5() {

const requiredFields = [

'org\_adresse\_complete', 'org\_province', 'org\_prefecture', 'org\_zone\_type'

];

let isValid = true;

requiredFields.forEach(fieldId => {

const field = document.getElementById(fieldId);

if (field && !validateField(field)) {

isValid = false;

}

});

return isValid;

}

/\*\*

\* Validation étape 6 : Fondateurs

\*/

function validateStep6() {

if (!OrganisationApp.selectedOrgType) return false;

const requirements = OrganisationApp.config.orgRequirements[OrganisationApp.selectedOrgType];

const minFondateurs = requirements ? requirements.minFondateurs : 3;

if (OrganisationApp.fondateurs.length < minFondateurs) {

showNotification(`Minimum ${minFondateurs} fondateurs requis pour ce type d'organisation`, 'warning');

return false;

}

return true;

}

/\*\*

\* Validation étape 7 : Adhérents

\*/

function validateStep7() {

if (!OrganisationApp.selectedOrgType) return false;

const requirements = OrganisationApp.config.orgRequirements[OrganisationApp.selectedOrgType];

const minAdherents = requirements ? requirements.minAdherents : 10;

if (OrganisationApp.adherents.length < minAdherents) {

showNotification(`Minimum ${minAdherents} adhérents requis pour ce type d'organisation`, 'warning');

return false;

}

return true;

}

/\*\*

\* Validation étape 8 : Documents

\*/

function validateStep8() {

if (!OrganisationApp.selectedOrgType) return false;

const requirements = OrganisationApp.config.orgRequirements[OrganisationApp.selectedOrgType];

const requiredDocs = requirements ? requirements.documents : [];

for (const doc of requiredDocs) {

if (!OrganisationApp.documents[doc]) {

showNotification(`Document requis manquant : ${getDocumentLabel(doc)}`, 'warning');

return false;

}

}

return true;

}

/\*\*

\* Validation étape 9 : Déclarations finales

\*/

function validateStep9() {

const declarations = ['declaration\_veracite', 'declaration\_conformite', 'declaration\_autorisation'];

// Ajouter la déclaration spécifique pour parti politique

if (OrganisationApp.selectedOrgType === 'parti\_politique') {

declarations.push('declaration\_exclusivite\_parti');

}

for (const declId of declarations) {

const decl = document.getElementById(declId);

if (!decl || !decl.checked) {

showFieldError(decl, 'Toutes les déclarations sont obligatoires');

return false;

}

}

return true;

}

/\*\*

\* Validation d'un champ individuel

\*/

function validateField(field) {

if (!field) return false;

const value = field.value.trim();

const fieldName = field.name || field.id;

// Champs obligatoires

if (field.hasAttribute('required') && !value) {

showFieldError(field, 'Ce champ est obligatoire');

return false;

}

// Validation spécifique selon le type de champ

switch (fieldName) {

case 'demandeur\_nip':

case 'fondateur\_nip':

case 'adherent\_nip':

return validateNIP(field, value);

case 'demandeur\_email':

case 'org\_email':

case 'fondateur\_email':

return validateEmail(field, value);

case 'demandeur\_telephone':

case 'org\_telephone':

case 'fondateur\_telephone':

case 'adherent\_telephone':

return validatePhone(field, value);

case 'demandeur\_date\_naissance':

return validateBirthDate(field, value);

case 'org\_nom':

return validateOrganizationName(field, value);

case 'org\_objet':

return validateOrgObjet(field, value);

default:

return validateGenericField(field, value);

}

}

/\*\*

\* Validation NIP gabonais (corrigée - sans checksum strict)

\*/

function validateNIP(field, value) {

if (!value) {

showFieldError(field, 'Le NIP est obligatoire');

return false;

}

// Validation de base seulement

if (!OrganisationApp.config.nip.pattern.test(value)) {

showFieldError(field, 'Le NIP doit contenir exactement 13 chiffres');

return false;

}

// Éviter les séquences évidentes

if (value === '1111111111111' || value === '1234567890123' || value === '0000000000000') {

showFieldError(field, 'NIP invalide');

return false;

}

// Validation réussie

clearFieldError(field);

updateNIPValidationIcon('valid');

return true;

}

/\*\*

\* Validation email

\*/

function validateEmail(field, value) {

if (!value && !field.hasAttribute('required')) return true;

if (!value) {

showFieldError(field, 'L\'email est obligatoire');

return false;

}

const emailPattern = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;

if (!emailPattern.test(value)) {

showFieldError(field, 'Format d\'email invalide');

return false;

}

clearFieldError(field);

return true;

}

/\*\*

\* Validation téléphone gabonais

\*/

function validatePhone(field, value) {

if (!value) {

showFieldError(field, 'Le téléphone est obligatoire');

return false;

}

// Nettoyer le numéro (enlever espaces et caractères spéciaux)

const cleanNumber = value.replace(/\s+/g, '').replace(/[^0-9]/g, '');

if (!OrganisationApp.config.phone.pattern.test(cleanNumber)) {

showFieldError(field, 'Format de téléphone gabonais invalide (8-9 chiffres)');

return false;

}

// Vérifier les préfixes valides

const prefix = cleanNumber.substring(0, 2);

if (!OrganisationApp.config.phone.prefixes.includes(prefix)) {

showFieldError(field, 'Préfixe téléphonique gabonais invalide');

return false;

}

clearFieldError(field);

return true;

}

/\*\*

\* Validation date de naissance

\*/

function validateBirthDate(field, value) {

if (!value) {

showFieldError(field, 'La date de naissance est obligatoire');

return false;

}

const birthDate = new Date(value);

const today = new Date();

const age = today.getFullYear() - birthDate.getFullYear();

if (age < 18) {

showFieldError(field, 'Vous devez être majeur (18 ans minimum)');

return false;

}

if (age > 100) {

showFieldError(field, 'Date de naissance invalide');

return false;

}

clearFieldError(field);

return true;

}

/\*\*

\* Validation nom organisation

\*/

function validateOrganizationName(field, value) {

if (!value) {

showFieldError(field, 'Le nom de l\'organisation est obligatoire');

return false;

}

if (value.length < 5) {

showFieldError(field, 'Le nom doit contenir au moins 5 caractères');

return false;

}

clearFieldError(field);

return true;

}

/\*\*

\* Validation objet social (minimum 50 caractères)

\*/

function validateOrgObjet(field, value) {

if (!value) {

showFieldError(field, 'L\'objet social est obligatoire');

return false;

}

const minLength = 50;

if (value.length < minLength) {

showFieldError(field, `L'objet social doit contenir au moins ${minLength} caractères (${value.length}/${minLength})`);

// Ajouter un compteur visuel

let counterDiv = field.parentNode.querySelector('.char-counter');

if (!counterDiv) {

counterDiv = document.createElement('div');

counterDiv.className = 'char-counter small text-muted mt-1';

field.parentNode.appendChild(counterDiv);

}

counterDiv.textContent = `${value.length}/${minLength} caractères`;

counterDiv.style.color = value.length < minLength ? '#dc3545' : '#28a745';

return false;

}

clearFieldError(field);

return true;

}

/\*\*

\* Validation générique

\*/

function validateGenericField(field, value) {

// Longueur minimale si spécifiée

if (field.hasAttribute('minlength')) {

const minLength = parseInt(field.getAttribute('minlength'));

if (value.length < minLength) {

showFieldError(field, `Minimum ${minLength} caractères requis`);

return false;

}

}

// Longueur maximale si spécifiée

if (field.hasAttribute('maxlength')) {

const maxLength = parseInt(field.getAttribute('maxlength'));

if (value.length > maxLength) {

showFieldError(field, `Maximum ${maxLength} caractères autorisés`);

return false;

}

}

clearFieldError(field);

return true;

}

/\*\*

\* Afficher une erreur sur un champ

\*/

function showFieldError(field, message) {

if (field) {

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

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

// Trouver ou créer l'élément d'erreur

let errorDiv = field.parentNode.querySelector('.invalid-feedback');

if (!errorDiv) {

errorDiv = document.createElement('div');

errorDiv.className = 'invalid-feedback';

field.parentNode.appendChild(errorDiv);

}

errorDiv.textContent = message;

// Animation d'erreur

field.style.animation = 'shake 0.5s ease-in-out';

setTimeout(() => {

field.style.animation = '';

}, 500);

} else {

// Message d'erreur générale

showNotification(message, 'danger');

}

// Sauvegarder l'erreur

if (field) {

OrganisationApp.validationErrors[field.name || field.id] = message;

}

}

/\*\*

\* Effacer l'erreur d'un champ

\*/

function clearFieldError(field) {

if (field) {

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

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

const errorDiv = field.parentNode.querySelector('.invalid-feedback');

if (errorDiv) {

errorDiv.textContent = '';

}

// Supprimer l'erreur du cache

delete OrganisationApp.validationErrors[field.name || field.id];

}

}

/\*\*

\* Mise à jour de l'icône de validation NIP

\*/

function updateNIPValidationIcon(status) {

const loading = document.getElementById('nip-loading');

const valid = document.getElementById('nip-valid');

const invalid = document.getElementById('nip-invalid');

// Masquer toutes les icônes

[loading, valid, invalid].forEach(icon => {

if (icon) icon.classList.add('d-none');

});

// Afficher l'icône appropriée

switch (status) {

case 'loading':

if (loading) loading.classList.remove('d-none');

break;

case 'valid':

if (valid) valid.classList.remove('d-none');

break;

case 'invalid':

if (invalid) invalid.classList.remove('d-none');

break;

}

}

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

// 5. GESTION FONDATEURS ET ADHÉRENTS

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

/\*\*

\* Mise à jour des exigences selon le type d'organisation

\*/

function updateOrganizationRequirements() {

if (!OrganisationApp.selectedOrgType) return;

const requirements = OrganisationApp.config.orgRequirements[OrganisationApp.selectedOrgType];

if (requirements) {

// Mettre à jour les exigences fondateurs

updateFoundersRequirements();

// Mettre à jour les exigences adhérents

updateMembersRequirements();

}

}

/\*\*

\* Mise à jour des exigences fondateurs

\*/

function updateFoundersRequirements() {

const requirementsDiv = document.getElementById('fondateurs\_requirements');

const minSpan = document.getElementById('min\_fondateurs');

if (!OrganisationApp.selectedOrgType) return;

const requirements = OrganisationApp.config.orgRequirements[OrganisationApp.selectedOrgType];

if (requirements && minSpan) {

minSpan.textContent = requirements.minFondateurs;

}

}

/\*\*

\* Mise à jour des exigences adhérents

\*/

function updateMembersRequirements() {

const requirementsDiv = document.getElementById('adherents\_requirements');

const minSpan = document.getElementById('min\_adherents');

if (!OrganisationApp.selectedOrgType) return;

const requirements = OrganisationApp.config.orgRequirements[OrganisationApp.selectedOrgType];

if (requirements && minSpan) {

minSpan.textContent = requirements.minAdherents;

}

}

/\*\*

\* Ajouter un fondateur

\*/

function addFondateur() {

const fondateur = {

civilite: document.getElementById('fondateur\_civilite').value,

nom: document.getElementById('fondateur\_nom').value,

prenom: document.getElementById('fondateur\_prenom').value,

nip: document.getElementById('fondateur\_nip').value,

fonction: document.getElementById('fondateur\_fonction').value,

telephone: document.getElementById('fondateur\_telephone').value,

email: document.getElementById('fondateur\_email').value

};

// Validation

if (!fondateur.nom || !fondateur.prenom || !fondateur.nip) {

showNotification('Veuillez remplir tous les champs obligatoires', 'warning');

return;

}

if (!validateNIP(document.getElementById('fondateur\_nip'), fondateur.nip)) {

return;

}

// Vérifier doublons

if (OrganisationApp.fondateurs.some(f => f.nip === fondateur.nip)) {

showNotification('Ce NIP existe déjà dans la liste des fondateurs', 'warning');

return;

}

// Ajouter à la liste

OrganisationApp.fondateurs.push(fondateur);

updateFoundersList();

clearFounderForm();

showNotification('Fondateur ajouté avec succès', 'success');

}

/\*\*

\* Mettre à jour la liste des fondateurs

\*/

function updateFoundersList() {

const listContainer = document.getElementById('fondateurs\_list');

const countSpan = document.getElementById('fondateurs\_count');

if (!listContainer) return;

if (OrganisationApp.fondateurs.length === 0) {

listContainer.innerHTML = `

<div class="text-center py-4 text-muted">

<i class="fas fa-users fa-3x mb-3"></i>

<p>Aucun fondateur ajouté</p>

</div>

`;

} else {

listContainer.innerHTML = OrganisationApp.fondateurs.map((fondateur, index) => `

<div class="d-flex justify-content-between align-items-center border-bottom py-2">

<div>

<strong>${fondateur.civilite} ${fondateur.nom} ${fondateur.prenom}</strong>

<br>

<small class="text-muted">NIP: ${fondateur.nip} | ${fondateur.fonction}</small>

${fondateur.telephone ? `<br><small class="text-muted">Tél: ${fondateur.telephone}</small>` : ''}

</div>

<button type="button" class="btn btn-sm btn-outline-danger" onclick="removeFondateur(${index})">

<i class="fas fa-trash"></i>

</button>

</div>

`).join('');

}

if (countSpan) {

countSpan.textContent = `${OrganisationApp.fondateurs.length} fondateur(s)`;

}

}

/\*\*

\* Supprimer un fondateur

\*/

function removeFondateur(index) {

if (confirm('Êtes-vous sûr de vouloir supprimer ce fondateur ?')) {

OrganisationApp.fondateurs.splice(index, 1);

updateFoundersList();

showNotification('Fondateur supprimé', 'info');

}

}

/\*\*

\* Vider le formulaire fondateur

\*/

function clearFounderForm() {

['fondateur\_civilite', 'fondateur\_nom', 'fondateur\_prenom', 'fondateur\_nip',

'fondateur\_fonction', 'fondateur\_telephone', 'fondateur\_email'].forEach(id => {

const field = document.getElementById(id);

if (field) {

if (field.tagName === 'SELECT') {

field.selectedIndex = 0;

} else {

field.value = '';

}

}

});

}

/\*\*

\* Ajouter un adhérent avec validation profession exclue

\*/

function addAdherent() {

const adherent = {

civilite: document.getElementById('adherent\_civilite').value,

nom: document.getElementById('adherent\_nom').value,

prenom: document.getElementById('adherent\_prenom').value,

nip: document.getElementById('adherent\_nip').value,

telephone: document.getElementById('adherent\_telephone').value,

profession: document.getElementById('adherent\_profession').value

};

// Validation

if (!adherent.nom || !adherent.prenom || !adherent.nip) {

showNotification('Veuillez remplir tous les champs obligatoires', 'warning');

return;

}

if (!validateNIP(document.getElementById('adherent\_nip'), adherent.nip)) {

return;

}

// Vérification profession exclue pour parti politique

if (OrganisationApp.selectedOrgType === 'parti\_politique' && adherent.profession) {

if (OrganisationApp.config.professionsExcluesParti.includes(adherent.profession)) {

if (!confirm(`⚠️ ATTENTION: La profession "${adherent.profession}" est normalement exclue des partis politiques selon la législation gabonaise.\n\nVoulez-vous tout de même ajouter cet adhérent ? (Il sera marqué avec une anomalie critique)`)) {

return;

}

// Marquer avec anomalie mais permettre l'ajout

adherent.hasAnomalies = true;

adherent.anomalies = [{

type: 'profession\_exclue\_parti',

level: 'critique',

message: 'Profession exclue pour parti politique',

details: `La profession "${adherent.profession}" est interdite pour les membres de partis politiques`

}];

}

}

// Vérifier doublons

if (OrganisationApp.adherents.some(a => a.nip === adherent.nip)) {

showNotification('Ce NIP existe déjà dans la liste des adhérents', 'warning');

return;

}

// Vérifier si ce NIP est déjà dans les fondateurs

if (OrganisationApp.fondateurs.some(f => f.nip === adherent.nip)) {

showNotification('Ce NIP existe déjà dans la liste des fondateurs', 'warning');

return;

}

// Ajouter à la liste

OrganisationApp.adherents.push(adherent);

updateAdherentsList();

clearAdherentForm();

// Message spécial si anomalie

if (adherent.hasAnomalies) {

showNotification('Adhérent ajouté avec anomalie critique (profession exclue)', 'warning');

} else {

showNotification('Adhérent ajouté avec succès', 'success');

}

}

/\*\*

\* Mettre à jour la liste des adhérents

\*/

function updateAdherentsList() {

const listContainer = document.getElementById('adherents\_list');

const countSpan = document.getElementById('adherents\_count');

if (!listContainer) return;

if (OrganisationApp.adherents.length === 0) {

listContainer.innerHTML = `

<div class="text-center py-4 text-muted">

<i class="fas fa-user-plus fa-3x mb-3"></i>

<p>Aucun adhérent ajouté</p>

</div>

`;

} else {

// Générer table avec pagination si plus de 10 adhérents

const itemsPerPage = 10;

const totalPages = Math.ceil(OrganisationApp.adherents.length / itemsPerPage);

const currentPage = 1; // Pour l'instant, page simple

const startIndex = (currentPage - 1) \* itemsPerPage;

const endIndex = startIndex + itemsPerPage;

const pageAdherents = OrganisationApp.adherents.slice(startIndex, endIndex);

listContainer.innerHTML = `

<div class="table-responsive">

<table class="table table-striped">

<thead>

<tr>

<th>Civilité</th>

<th>Nom complet</th>

<th>NIP</th>

<th>Téléphone</th>

<th>Profession</th>

<th>Statut</th>

<th>Actions</th>

</tr>

</thead>

<tbody>

${pageAdherents.map((adherent, index) => `

<tr ${adherent.hasAnomalies ? 'class="table-warning"' : ''}>

<td>${adherent.civilite}</td>

<td><strong>${adherent.nom} ${adherent.prenom}</strong></td>

<td><code>${adherent.nip}</code></td>

<td>${adherent.telephone || '-'}</td>

<td>${adherent.profession || '-'}</td>

<td>

${adherent.hasAnomalies ?

`<span class="badge bg-danger" title="${adherent.anomalies[0]?.details || 'Anomalie détectée'}">Anomalie</span>` :

`<span class="badge bg-success">Valide</span>`

}

</td>

<td>

<button type="button" class="btn btn-sm btn-outline-danger" onclick="removeAdherent(${startIndex + index})">

<i class="fas fa-trash"></i>

</button>

</td>

</tr>

`).join('')}

</tbody>

</table>

</div>

${totalPages > 1 ? `

<nav>

<ul class="pagination pagination-sm justify-content-center">

<!-- Pagination à implémenter si nécessaire -->

</ul>

</nav>

` : ''}

`;

}

if (countSpan) {

// Compter les adhérents valides et avec anomalies

const valides = OrganisationApp.adherents.filter(a => !a.hasAnomalies).length;

const anomalies = OrganisationApp.adherents.filter(a => a.hasAnomalies).length;

if (anomalies > 0) {

countSpan.innerHTML = `${OrganisationApp.adherents.length} adhérent(s) <small class="text-muted">(${valides} valides, ${anomalies} anomalies)</small>`;

} else {

countSpan.textContent = `${OrganisationApp.adherents.length} adhérent(s)`;

}

}

}

/\*\*

\* Supprimer un adhérent

\*/

function removeAdherent(index) {

if (confirm('Êtes-vous sûr de vouloir supprimer cet adhérent ?')) {

OrganisationApp.adherents.splice(index, 1);

updateAdherentsList();

showNotification('Adhérent supprimé', 'info');

}

}

/\*\*

\* Vider le formulaire adhérent

\*/

function clearAdherentForm() {

['adherent\_civilite', 'adherent\_nom', 'adherent\_prenom', 'adherent\_nip',

'adherent\_telephone', 'adherent\_profession'].forEach(id => {

const field = document.getElementById(id);

if (field) {

if (field.tagName === 'SELECT') {

field.selectedIndex = 0;

} else {

field.value = '';

}

}

});

}

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

// 5.1 IMPORTATION FICHIER ADHÉRENTS - VERSION COMPLÈTE

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

/\*\*

\* Gestion de l'importation du fichier Excel/CSV des adhérents

\*/

async function handleAdherentFileImport(fileInput) {

const file = fileInput.files[0];

if (!file) return;

console.log('📁 Début importation fichier adhérents:', file.name);

// Validation du fichier

if (!validateAdherentFile(file)) {

clearFileInput();

return;

}

try {

showNotification('📁 Analyse du fichier en cours...', 'info');

// Lire le fichier Excel/CSV

const adherentsData = await readAdherentFile(file);

if (!adherentsData || adherentsData.length === 0) {

showNotification('❌ Le fichier est vide ou invalide', 'danger');

clearFileInput();

return;

}

console.log(`📊 ${adherentsData.length} lignes détectées dans le fichier`);

// Valider et traiter les données avec système d'anomalies

const validationResult = await validateAdherentsImport(adherentsData);

// Traiter selon les résultats de validation

await processImportResult(validationResult);

} catch (error) {

console.error('❌ Erreur importation adhérents:', error);

showNotification('❌ Erreur lors de l\'importation: ' + error.message, 'danger');

clearFileInput();

}

}

/\*\*

\* Validation du fichier (format, taille)

\*/

function validateAdherentFile(file) {

// Vérifier la taille (max 5MB)

if (file.size > 5 \* 1024 \* 1024) {

showNotification('❌ Le fichier ne peut pas dépasser 5MB', 'danger');

return false;

}

// Vérifier le format

const allowedTypes = [

'application/vnd.openxmlformats-officedocument.spreadsheetml.sheet', // .xlsx

'application/vnd.ms-excel', // .xls

'text/csv'

];

if (!allowedTypes.includes(file.type) && !file.name.match(/\.(xlsx|xls|csv)$/i)) {

showNotification('❌ Format de fichier non autorisé. Utilisez Excel (.xlsx, .xls) ou CSV', 'danger');

return false;

}

return true;

}

/\*\*

\* Lecture du fichier Excel/CSV

\*/

async function readAdherentFile(file) {

return new Promise((resolve, reject) => {

const reader = new FileReader();

reader.onload = function(e) {

try {

let data = [];

if (file.type === 'text/csv' || file.name.endsWith('.csv')) {

// Traitement CSV

const csvText = e.target.result;

data = parseAdherentCSV(csvText);

} else {

// Traitement Excel avec XLSX.js si disponible

try {

if (typeof XLSX !== 'undefined') {

// Si XLSX est disponible

const workbook = XLSX.read(e.target.result, { type: 'array' });

data = parseAdherentExcel(workbook);

} else {

// Si XLSX n'est pas disponible, simuler avec FileReader HTML5

data = parseExcelFallback(e.target.result, file.name);

}

} catch (excelError) {

console.warn('Erreur parsing Excel:', excelError);

reject(new Error('Erreur lors de la lecture du fichier Excel. Utilisez un fichier CSV ou vérifiez le format.'));

return;

}

}

resolve(data);

} catch (error) {

reject(error);

}

};

reader.onerror = () => reject(new Error('Erreur lecture fichier'));

if (file.type === 'text/csv' || file.name.endsWith('.csv')) {

reader.readAsText(file);

} else {

reader.readAsArrayBuffer(file);

}

});

}

/\*\*

\* Parsing Excel avec XLSX.js

\*/

function parseAdherentExcel(workbook) {

const sheetName = workbook.SheetNames[0];

const worksheet = workbook.Sheets[sheetName];

// Convertir en JSON

const jsonData = XLSX.utils.sheet\_to\_json(worksheet, { header: 1 });

if (jsonData.length < 2) {

throw new Error('Le fichier Excel doit contenir au moins un en-tête et une ligne de données');

}

const headers = jsonData[0].map(h => h.toString().trim().toLowerCase());

const data = [];

// Vérifier les colonnes requises

const requiredColumns = ['civilité', 'nom', 'prenom', 'nip'];

const missingColumns = requiredColumns.filter(col =>

!headers.some(h => h.includes(col.replace('é', 'e')) || h.includes(col))

);

if (missingColumns.length > 0) {

throw new Error(`Colonnes manquantes: ${missingColumns.join(', ')}`);

}

for (let i = 1; i < jsonData.length; i++) {

const values = jsonData[i];

if (!values || values.length < 3) continue; // Ignorer les lignes vides

const row = {};

headers.forEach((header, index) => {

row[header] = values[index] ? values[index].toString().trim() : '';

});

// Mapper vers notre format standard

const adherent = {

civilite: row.civilité || row.civilite || 'M',

nom: row.nom,

prenom: row.prenom || row.prénom,

nip: row.nip,

telephone: row.telephone || row.téléphone || '',

profession: row.profession || '',

lineNumber: i + 1

};

// Valider que les champs obligatoires sont présents

if (adherent.nom && adherent.prenom && adherent.nip) {

data.push(adherent);

}

}

return data;

}

/\*\*

\* Fallback pour Excel sans XLSX.js (méthode alternative)

\*/

function parseExcelFallback(arrayBuffer, fileName) {

// Pour l'instant, rejeter avec un message d'aide

throw new Error(

`Pour importer des fichiers Excel (.xlsx, .xls), veuillez :\n` +

`1. Convertir votre fichier en CSV, ou\n` +

`2. Installer la bibliothèque XLSX.js\n\n` +

`En attendant, utilisez un fichier CSV avec les colonnes : Civilité,Nom,Prenom,NIP,Telephone,Profession`

);

}

/\*\*

\* Parsing CSV des adhérents - VERSION AMÉLIORÉE

\*/

function parseAdherentCSV(csvText) {

const lines = csvText.split('\n').filter(line => line.trim());

if (lines.length < 2) {

throw new Error('Le fichier CSV doit contenir au moins un en-tête et une ligne de données');

}

// Améliorer le parsing pour gérer les guillemets et virgules dans les champs

const parseCSVLine = (line) => {

const result = [];

let current = '';

let inQuotes = false;

for (let i = 0; i < line.length; i++) {

const char = line[i];

const nextChar = line[i + 1];

if (char === '"') {

if (inQuotes && nextChar === '"') {

current += '"';

i++; // Skip next quote

} else {

inQuotes = !inQuotes;

}

} else if (char === ',' && !inQuotes) {

result.push(current.trim());

current = '';

} else {

current += char;

}

}

result.push(current.trim());

return result;

};

const headers = parseCSVLine(lines[0]).map(h => h.toLowerCase().replace(/"/g, ''));

const data = [];

// Vérifier les colonnes requises avec plus de flexibilité

const findColumn = (searchTerms) => {

return headers.findIndex(h =>

searchTerms.some(term => h.includes(term))

);

};

const civiliteIndex = findColumn(['civilité', 'civilite', 'civ']);

const nomIndex = findColumn(['nom', 'name', 'lastname']);

const prenomIndex = findColumn(['prenom', 'prénom', 'firstname']);

const nipIndex = findColumn(['nip', 'id', 'numero']);

const telephoneIndex = findColumn(['telephone', 'téléphone', 'phone', 'tel']);

const professionIndex = findColumn(['profession', 'metier', 'job']);

if (nomIndex === -1 || prenomIndex === -1 || nipIndex === -1) {

throw new Error('Colonnes obligatoires manquantes : Nom, Prénom, NIP');

}

for (let i = 1; i < lines.length; i++) {

const values = parseCSVLine(lines[i]);

if (values.length < 3) continue; // Ignorer les lignes insuffisantes

const adherent = {

civilite: civiliteIndex !== -1 ? (values[civiliteIndex] || 'M') : 'M',

nom: values[nomIndex] || '',

prenom: values[prenomIndex] || '',

nip: values[nipIndex] || '',

telephone: telephoneIndex !== -1 ? (values[telephoneIndex] || '') : '',

profession: professionIndex !== -1 ? (values[professionIndex] || '') : '',

lineNumber: i + 1

};

// Valider que les champs obligatoires sont présents

if (adherent.nom && adherent.prenom && adherent.nip) {

data.push(adherent);

}

}

return data;

}

/\*\*

\* Validation complète des données d'importation avec système d'anomalies

\*/

async function validateAdherentsImport(adherentsData) {

console.log('📋 Validation avec gestion anomalies - Version 1.2');

const result = {

originalCount: adherentsData.length,

adherentsValides: [],

adherentsAvecAnomalies: [],

adherentsTotal: [], // Tous les adhérents (valides + anomalies)

duplicatesInFile: [],

existingMembers: [],

invalidEntries: [],

finalValidCount: 0,

finalAnomaliesCount: 0,

canProceed: true, // Toujours true maintenant si minimum atteint

messages: [],

qualiteGlobale: 'excellent'

};

// Réinitialiser le rapport d'anomalies

OrganisationApp.rapportAnomalies = {

enabled: false,

adherentsValides: 0,

adherentsAvecAnomalies: 0,

anomalies: [],

statistiques: { critique: 0, majeure: 0, mineure: 0 },

genereAt: null,

version: '1.2'

};

// Obtenir les exigences selon le type d'organisation

const requirements = OrganisationApp.config.orgRequirements[OrganisationApp.selectedOrgType];

const minRequired = requirements ? requirements.minAdherents : 10;

console.log(`📊 Validation import: ${adherentsData.length} adhérents, minimum requis: ${minRequired}`);

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

// ÉTAPE 1 : Détection doublons NIP dans le fichier

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

const seenNips = new Set();

const processedAdherents = [];

adherentsData.forEach((adherent, index) => {

const nip = adherent.nip?.trim();

// Créer un ID unique pour chaque adhérent

adherent.id = `adherent\_${Date.now()}\_${index}`;

adherent.hasAnomalies = false;

adherent.anomalies = [];

// Validation NIP de base

if (!nip) {

const anomalie = createAnomalie(adherent, 'champs\_incomplets', 'NIP manquant');

if (anomalie) {

adherent.anomalies.push(anomalie);

adherent.hasAnomalies = true;

}

} else if (!OrganisationApp.config.nip.pattern.test(nip)) {

const anomalie = createAnomalie(adherent, 'nip\_invalide', `Format NIP incorrect: ${nip}`);

if (anomalie) {

adherent.anomalies.push(anomalie);

adherent.hasAnomalies = true;

}

} else if (seenNips.has(nip)) {

// Doublon dans le fichier

const anomalie = createAnomalie(adherent, 'doublon\_fichier', `NIP ${nip} déjà présent ligne précédente`);

if (anomalie) {

adherent.anomalies.push(anomalie);

adherent.hasAnomalies = true;

}

result.duplicatesInFile.push({ ...adherent, nip: nip });

} else {

seenNips.add(nip);

}

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

// VALIDATION AUTRES CHAMPS AVEC ANOMALIES

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

// Validation nom/prénom

if (!adherent.nom || !adherent.prenom) {

const anomalie = createAnomalie(adherent, 'champs\_incomplets', 'Nom ou prénom manquant');

if (anomalie) {

adherent.anomalies.push(anomalie);

adherent.hasAnomalies = true;

}

}

// Validation téléphone (si présent)

if (adherent.telephone && !OrganisationApp.config.phone.pattern.test(adherent.telephone.replace(/\s+/g, ''))) {

const anomalie = createAnomalie(adherent, 'telephone\_invalide', `Format téléphone incorrect: ${adherent.telephone}`);

if (anomalie) {

adherent.anomalies.push(anomalie);

adherent.hasAnomalies = true;

}

}

// Validation email (si présent)

if (adherent.email && !/^[^\s@]+@[^\s@]+\.[^\s@]+$/.test(adherent.email)) {

const anomalie = createAnomalie(adherent, 'email\_invalide', `Format email incorrect: ${adherent.email}`);

if (anomalie) {

adherent.anomalies.push(anomalie);

adherent.hasAnomalies = true;

}

}

// Validation profession exclue pour parti politique

if (OrganisationApp.selectedOrgType === 'parti\_politique' && adherent.profession) {

if (OrganisationApp.config.professionsExcluesParti.includes(adherent.profession)) {

const anomalie = createAnomalie(adherent, 'profession\_exclue\_parti',

`Profession "${adherent.profession}" interdite pour parti politique`);

if (anomalie) {

adherent.anomalies.push(anomalie);

adherent.hasAnomalies = true;

}

}

}

// Validation format données générales

if (adherent.nom && adherent.nom.length < 2) {

const anomalie = createAnomalie(adherent, 'format\_donnees', 'Nom trop court');

if (anomalie) {

adherent.anomalies.push(anomalie);

adherent.hasAnomalies = true;

}

}

processedAdherents.push(adherent);

});

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

// ÉTAPE 2 : Vérification doublons avec fondateurs/adhérents existants

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

const foundersNips = OrganisationApp.fondateurs.map(f => f.nip);

const adherentsNips = OrganisationApp.adherents.map(a => a.nip);

processedAdherents.forEach(adherent => {

if (foundersNips.includes(adherent.nip)) {

const anomalie = createAnomalie(adherent, 'doublon\_fichier', 'NIP déjà présent dans les fondateurs');

if (anomalie) {

adherent.anomalies.push(anomalie);

adherent.hasAnomalies = true;

}

}

if (adherentsNips.includes(adherent.nip)) {

const anomalie = createAnomalie(adherent, 'doublon\_fichier', 'NIP déjà présent dans les adhérents');

if (anomalie) {

adherent.anomalies.push(anomalie);

adherent.hasAnomalies = true;

}

}

});

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

// ÉTAPE 3 : Vérification membres existants via API

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

const nipsToCheck = processedAdherents

.filter(a => a.nip && OrganisationApp.config.nip.pattern.test(a.nip))

.map(a => a.nip);

const existingMembersNips = await checkExistingMembersAPI(nipsToCheck);

processedAdherents.forEach(adherent => {

if (existingMembersNips.includes(adherent.nip)) {

const anomalie = createAnomalie(adherent, 'membre\_existant', 'Déjà membre actif d\'une autre organisation');

if (anomalie) {

adherent.anomalies.push(anomalie);

adherent.hasAnomalies = true;

}

result.existingMembers.push(adherent);

}

});

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

// ÉTAPE 4 : CLASSIFICATION FINALE

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

processedAdherents.forEach(adherent => {

if (adherent.hasAnomalies) {

// Ajouter toutes les anomalies au rapport global

adherent.anomalies.forEach(anomalie => {

addAnomalieToReport(anomalie);

});

// Marquer comme adhérent avec anomalies

adherent.status = 'anomalie';

adherent.statusLabel = 'Anomalie détectée';

adherent.statusColor = 'warning';

// Déterminer le niveau de gravité le plus élevé

const niveaux = adherent.anomalies.map(a => a.level);

if (niveaux.includes('critique')) {

adherent.priorityLevel = 'critique';

adherent.statusColor = 'danger';

} else if (niveaux.includes('majeure')) {

adherent.priorityLevel = 'majeure';

adherent.statusColor = 'warning';

} else {

adherent.priorityLevel = 'mineure';

adherent.statusColor = 'info';

}

result.adherentsAvecAnomalies.push(adherent);

} else {

// Adhérent valide

adherent.status = 'valide';

adherent.statusLabel = 'Valide';

adherent.statusColor = 'success';

adherent.priorityLevel = null;

result.adherentsValides.push(adherent);

}

// Tous les adhérents sont conservés

result.adherentsTotal.push(adherent);

});

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

// MISE À JOUR STATISTIQUES FINALES

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

result.finalValidCount = result.adherentsValides.length;

result.finalAnomaliesCount = result.adherentsAvecAnomalies.length;

// Mettre à jour le rapport d'anomalies

if (result.finalAnomaliesCount > 0) {

OrganisationApp.rapportAnomalies.enabled = true;

OrganisationApp.rapportAnomalies.adherentsValides = result.finalValidCount;

OrganisationApp.rapportAnomalies.adherentsAvecAnomalies = result.finalAnomaliesCount;

OrganisationApp.rapportAnomalies.genereAt = new Date().toISOString();

}

// Déterminer la qualité globale

result.qualiteGlobale = getQualiteStatut();

// Toujours permettre l'importation si minimum atteint

const totalAdherents = result.finalValidCount + result.finalAnomaliesCount;

result.canProceed = totalAdherents >= minRequired;

// Générer les messages selon les nouveaux critères

result.messages = generateImportMessagesWithAnomalies(result, minRequired);

console.log('📊 Résultat validation avec anomalies:', {

total: totalAdherents,

valides: result.finalValidCount,

anomalies: result.finalAnomaliesCount,

qualite: result.qualiteGlobale,

canProceed: result.canProceed

});

return result;

}

/\*\*

\* Génération des messages avec gestion des anomalies

\*/

function generateImportMessagesWithAnomalies(result, minRequired) {

const messages = [];

const totalAdherents = result.finalValidCount + result.finalAnomaliesCount;

// Message principal selon le résultat

if (result.canProceed) {

if (result.finalAnomaliesCount === 0) {

messages.push({

type: 'success',

title: '✅ Importation parfaite',

content: `${result.finalValidCount} adhérents valides détectés. Aucune anomalie trouvée. Minimum requis: ${minRequired}`

});

} else {

messages.push({

type: 'warning',

title: '⚠️ Importation avec anomalies',

content: `${totalAdherents} adhérents détectés (${result.finalValidCount} valides + ${result.finalAnomaliesCount} avec anomalies). Un rapport sera généré. Minimum requis: ${minRequired}`

});

}

} else {

messages.push({

type: 'danger',

title: '❌ Importation impossible',

content: `Seulement ${totalAdherents} adhérents détectés, minimum requis: ${minRequired}.`

});

}

// Message sur la qualité globale

const qualiteMessages = {

'excellent': { type: 'success', message: '🌟 Excellente qualité des données' },

'bon': { type: 'info', message: '👍 Bonne qualité des données' },

'moyen': { type: 'warning', message: '⚠️ Qualité moyenne des données' },

'faible': { type: 'danger', message: '❌ Qualité faible des données' }

};

if (qualiteMessages[result.qualiteGlobale]) {

const qMsg = qualiteMessages[result.qualiteGlobale];

messages.push({

type: qMsg.type,

title: 'Évaluation qualité',

content: qMsg.message

});

}

// Messages spécifiques pour les anomalies

if (result.finalAnomaliesCount > 0) {

const stats = OrganisationApp.rapportAnomalies.statistiques;

messages.push({

type: 'info',

title: '📋 Rapport d\'anomalies généré',

content: `${result.finalAnomaliesCount} adhérent(s) avec anomalies : ${stats.critique} critique(s), ${stats.majeure} majeure(s), ${stats.mineure} mineure(s)`,

details: result.adherentsAvecAnomalies.map(a =>

`Ligne ${a.lineNumber}: ${a.nom} ${a.prenom} (${a.anomalies.length} anomalie(s) ${a.priorityLevel})`

)

});

}

return messages;

}

/\*\*

\* Vérification des membres existants via API

\*/

async function checkExistingMembersAPI(nips) {

if (nips.length === 0) return [];

try {

const response = await fetch('/api/organisations/check-existing-members', {

method: 'POST',

headers: {

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

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

'Accept': 'application/json'

},

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

});

if (response.ok) {

const data = await response.json();

return data.existing\_nips || [];

} else if (response.status === 404) {

console.warn('API check membres existants non trouvée, import sans vérification');

return [];

} else {

console.warn('Erreur API check membres existants:', response.status);

return [];

}

} catch (error) {

console.error('Erreur vérification membres existants:', error);

return [];

}

}

/\*\*

\* Traitement du résultat d'importation

\*/

async function processImportResult(validationResult) {

const { canProceed, adherentsTotal, messages, originalCount, finalValidCount, finalAnomaliesCount } = validationResult;

// Afficher tous les messages de validation

messages.forEach(message => {

showDetailedImportNotification(message);

});

if (!canProceed) {

showNotification('❌ Importation annulée: critères non remplis', 'danger');

clearFileInput();

return;

}

// Message de confirmation avec anomalies

const totalImport = finalValidCount + finalAnomaliesCount;

let confirmMsg = `Importation de ${totalImport} adhérents sur ${originalCount} lignes analysées :\n`;

confirmMsg += `• ${finalValidCount} adhérents valides\n`;

if (finalAnomaliesCount > 0) {

confirmMsg += `• ${finalAnomaliesCount} adhérents avec anomalies (seront conservés)\n`;

confirmMsg += `\n⚠️ Un rapport d'anomalies sera généré automatiquement.\n`;

}

confirmMsg += `\nConfirmez-vous l'importation ?`;

if (!confirm(confirmMsg)) {

showNotification('❌ Importation annulée par l\'utilisateur', 'info');

clearFileInput();

return;

}

// Ajouter TOUS les adhérents (valides + anomalies)

adherentsTotal.forEach(adherent => {

OrganisationApp.adherents.push(adherent);

});

// Mettre à jour l'affichage avec les nouveaux statuts

updateAdherentsList();

// Message de succès détaillé

let successDetails = [`🎉 Importation réussie !`];

successDetails.push(`📊 ${finalValidCount} adhérents valides ajoutés`);

if (finalAnomaliesCount > 0) {

successDetails.push(`⚠️ ${finalAnomaliesCount} avec anomalies conservés`);

successDetails.push(`📋 Rapport d'anomalies généré automatiquement`);

}

successDetails.push(`📁 Total: ${totalImport} entrées traitées`);

showNotification(successDetails.join('\n'), 'success', 10000);

// Vider le champ fichier et sauvegarder

clearFileInput();

autoSave();

console.log('✅ Import terminé v1.2:', {

valides: finalValidCount,

anomalies: finalAnomaliesCount,

total: totalImport,

rapportGenere: OrganisationApp.rapportAnomalies.enabled

});

}

/\*\*

\* Affichage de notification détaillée pour l'import

\*/

function showDetailedImportNotification(message) {

const hasDetails = message.details && message.details.length > 0;

const detailsId = `details-${Date.now()}-${Math.random().toString(36).substr(2, 9)}`;

let notificationContent = `

<div class="d-flex align-items-start">

<div class="flex-grow-1">

<strong>${message.title}</strong>

<div class="mt-1">${message.content}</div>

${hasDetails ? `

<button class="btn btn-sm btn-outline-secondary mt-2" type="button"

onclick="toggleImportDetails('${detailsId}')">

<i class="fas fa-chevron-down me-1"></i>Voir détails (${message.details.length})

</button>

<div id="${detailsId}" class="mt-2 d-none small">

<div class="bg-light p-2 rounded" style="max-height: 150px; overflow-y: auto;">

${message.details.map(detail => `• ${detail}`).join('<br>')}

</div>

</div>

` : ''}

</div>

</div>

`;

// Créer notification personnalisée

showCustomNotification(notificationContent, message.type, hasDetails ? 15000 : 8000);

}

/\*\*

\* Basculer l'affichage des détails d'import

\*/

function toggleImportDetails(detailsId) {

const detailsElement = document.getElementById(detailsId);

const button = event.target.closest('button');

if (detailsElement && button) {

if (detailsElement.classList.contains('d-none')) {

detailsElement.classList.remove('d-none');

button.innerHTML = '<i class="fas fa-chevron-up me-1"></i>Masquer détails';

} else {

detailsElement.classList.add('d-none');

button.innerHTML = '<i class="fas fa-chevron-down me-1"></i>Voir détails';

}

}

}

/\*\*

\* Vider le champ fichier

\*/

function clearFileInput() {

const fileInput = document.getElementById('adherents\_file');

if (fileInput) {

fileInput.value = '';

}

}

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

// 6. GESTION DOCUMENTS

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

/\*\*

\* Mise à jour des documents requis

\*/

function updateDocumentsRequirements() {

const container = document.getElementById('documents\_container');

if (!container || !OrganisationApp.selectedOrgType) return;

const requirements = OrganisationApp.config.orgRequirements[OrganisationApp.selectedOrgType];

if (!requirements) return;

const documentsHTML = requirements.documents.map(doc => `

<div class="card border-0 shadow-sm mb-3">

<div class="card-header bg-info text-white">

<h6 class="mb-0">

<i class="fas fa-file-alt me-2"></i>

${getDocumentLabel(doc)}

<span class="badge bg-light text-dark ms-2">Obligatoire</span>

</h6>

</div>

<div class="card-body">

<div class="row">

<div class="col-md-8">

<input type="file"

class="form-control"

id="doc\_${doc}"

name="documents[${doc}]"

accept=".pdf,.jpg,.jpeg,.png"

onchange="handleDocumentUpload('${doc}', this)">

<div class="form-text">

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

Formats acceptés : PDF, JPG, PNG (max 5MB)

</div>

</div>

<div class="col-md-4">

<div id="status\_${doc}" class="text-muted">

<i class="fas fa-clock me-1"></i>En attente

</div>

<div class="progress mt-2 d-none" id="progress\_container\_${doc}">

<div class="progress-bar" id="progress\_${doc}" style="width: 0%"></div>

</div>

<div id="preview\_${doc}" class="mt-2 d-none"></div>

</div>

</div>

</div>

</div>

`).join('');

container.innerHTML = documentsHTML;

}

/\*\*

\* Obtenir le label d'un document

\*/

function getDocumentLabel(doc) {

const labels = {

'statuts': 'Statuts de l\'organisation',

'pv\_ag': 'Procès-verbal de l\'assemblée générale constitutive',

'liste\_fondateurs': 'Liste des membres fondateurs',

'justif\_siege': 'Justificatif du siège social',

'projet\_social': 'Projet social détaillé',

'budget\_previsionnel': 'Budget prévisionnel',

'programme\_politique': 'Programme politique',

'liste\_50\_adherents': 'Liste de 50 adhérents minimum',

'expose\_doctrine': 'Exposé de la doctrine religieuse',

'justif\_lieu\_culte': 'Justificatif du lieu de culte'

};

return labels[doc] || doc;

}

/\*\*

\* Gestion upload document

\*/

function handleDocumentUpload(docType, fileInput) {

const file = fileInput.files[0];

if (!file) return;

// Validation

if (file.size > 5 \* 1024 \* 1024) {

showNotification('Le fichier ne peut pas dépasser 5MB', 'danger');

fileInput.value = '';

return;

}

const allowedTypes = ['application/pdf', 'image/jpeg', 'image/png'];

if (!allowedTypes.includes(file.type)) {

showNotification('Type de fichier non autorisé. Utilisez PDF, JPG ou PNG.', 'danger');

fileInput.value = '';

return;

}

// Simuler upload (remplacer par vraie logique)

const statusElement = document.getElementById(`status\_${docType}`);

const progressContainer = document.getElementById(`progress\_container\_${docType}`);

const progressBar = document.getElementById(`progress\_${docType}`);

const previewContainer = document.getElementById(`preview\_${docType}`);

if (statusElement) {

statusElement.innerHTML = '<i class="fas fa-spinner fa-spin me-1 text-primary"></i>Upload en cours...';

}

if (progressContainer) {

progressContainer.classList.remove('d-none');

}

// Simulation progress

let progress = 0;

const interval = setInterval(() => {

progress += Math.random() \* 20;

if (progress > 100) progress = 100;

if (progressBar) {

progressBar.style.width = progress + '%';

progressBar.textContent = Math.round(progress) + '%';

}

if (progress >= 100) {

clearInterval(interval);

// Marquer comme uploadé

OrganisationApp.documents[docType] = {

file: file,

uploaded: true,

uploadedAt: new Date(),

fileName: file.name,

fileSize: file.size,

fileType: file.type

};

if (statusElement) {

statusElement.innerHTML = '<i class="fas fa-check text-success me-1"></i>Uploadé avec succès';

}

// Générer preview pour les images

if (file.type.startsWith('image/') && previewContainer) {

const reader = new FileReader();

reader.onload = function(e) {

previewContainer.innerHTML = `

<img src="${e.target.result}" class="img-thumbnail" style="max-height: 100px; cursor: pointer;"

onclick="openImageModal('${e.target.result}')" />

`;

previewContainer.classList.remove('d-none');

};

reader.readAsDataURL(file);

}

setTimeout(() => {

if (progressContainer) {

progressContainer.classList.add('d-none');

}

}, 2000);

showNotification(`Document "${getDocumentLabel(docType)}" uploadé avec succès`, 'success');

}

}, 200);

}

/\*\*

\* Ouvrir une image en modal

\*/

function openImageModal(imageSrc) {

const modalHtml = `

<div class="modal fade" id="imageModal" tabindex="-1">

<div class="modal-dialog modal-lg modal-dialog-centered">

<div class="modal-content">

<div class="modal-header">

<h5 class="modal-title">Aperçu du document</h5>

<button type="button" class="btn-close" data-bs-dismiss="modal"></button>

</div>

<div class="modal-body text-center">

<img src="${imageSrc}" class="img-fluid" style="max-height: 70vh;" />

</div>

<div class="modal-footer">

<button type="button" class="btn btn-secondary" data-bs-dismiss="modal">

Fermer

</button>

</div>

</div>

</div>

</div>

`;

// Supprimer l'ancienne modal si elle existe

const existingModal = document.getElementById('imageModal');

if (existingModal) {

existingModal.remove();

}

// Ajouter et afficher la nouvelle modal

document.body.insertAdjacentHTML('beforeend', modalHtml);

const modal = new bootstrap.Modal(document.getElementById('imageModal'));

modal.show();

// Nettoyer après fermeture

modal.\_element.addEventListener('hidden.bs.modal', () => {

modal.\_element.remove();

});

}

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

// 7. GÉNÉRATION RÉCAPITULATIF

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

/\*\*

\* Générer le récapitulatif final

\*/

function generateRecap() {

console.log('📋 Génération récapitulatif avec rapport d\'anomalies - Version 1.2');

const container = document.getElementById('recap\_content');

if (!container) return;

const formData = collectFormData();

const recapHTML = `

<div class="row">

<div class="col-md-6">

<h6 class="text-primary mb-3">

<i class="fas fa-building me-2"></i>

Informations de l'organisation

</h6>

<table class="table table-sm table-borderless">

<tr><td><strong>Type :</strong></td><td>${getOrganizationTypeLabel(OrganisationApp.selectedOrgType)}</td></tr>

<tr><td><strong>Nom :</strong></td><td>${formData.org\_nom || 'Non renseigné'}</td></tr>

<tr><td><strong>Sigle :</strong></td><td>${formData.org\_sigle || 'Aucun'}</td></tr>

<tr><td><strong>Téléphone :</strong></td><td>${formData.org\_telephone || 'Non renseigné'}</td></tr>

<tr><td><strong>Email :</strong></td><td>${formData.org\_email || 'Non renseigné'}</td></tr>

<tr><td><strong>Province :</strong></td><td>${formData.org\_province || 'Non renseigné'}</td></tr>

</table>

<h6 class="text-success mb-3 mt-4">

<i class="fas fa-users me-2"></i>

Composition

</h6>

${generateCompositionWithQuality(formData)}

<div class="mt-3">

<small class="text-muted">

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

Exigences : ${OrganisationApp.config.orgRequirements[OrganisationApp.selectedOrgType]?.minFondateurs || 3} fondateurs min,

${OrganisationApp.config.orgRequirements[OrganisationApp.selectedOrgType]?.minAdherents || 10} adhérents min

</small>

</div>

</div>

<div class="col-md-6">

<h6 class="text-info mb-3">

<i class="fas fa-user me-2"></i>

Demandeur principal

</h6>

<table class="table table-sm table-borderless">

<tr><td><strong>Nom :</strong></td><td>${formData.demandeur\_civilite || ''} ${formData.demandeur\_nom || ''} ${formData.demandeur\_prenom || ''}</td></tr>

<tr><td><strong>NIP :</strong></td><td><code>${formData.demandeur\_nip || 'Non renseigné'}</code></td></tr>

<tr><td><strong>Téléphone :</strong></td><td>${formData.demandeur\_telephone || 'Non renseigné'}</td></tr>

<tr><td><strong>Email :</strong></td><td>${formData.demandeur\_email || 'Non renseigné'}</td></tr>

<tr><td><strong>Rôle :</strong></td><td><span class="badge bg-primary">${formData.demandeur\_role || 'Non renseigné'}</span></td></tr>

</table>

<h6 class="text-warning mb-3 mt-4">

<i class="fas fa-file-alt me-2"></i>

Documents (${Object.keys(OrganisationApp.documents).length})

</h6>

<div>

${Object.keys(OrganisationApp.documents).length > 0 ?

Object.keys(OrganisationApp.documents).map(doc => `

<div class="d-flex align-items-center mb-1">

<i class="fas fa-check text-success me-2"></i>

<small>${getDocumentLabel(doc)}</small>

</div>

`).join('') :

'<small class="text-muted">Aucun document uploadé</small>'

}

</div>

</div>

</div>

<!-- Section rapport d'anomalies conditionnelle -->

${generateAnomaliesRecapSection()}

<!-- Statut de validation mis à jour -->

<div class="row mt-4">

<div class="col-12">

${generateValidationStatusWithQuality()}

</div>

</div>

<!-- Section spéciale pour parti politique -->

${OrganisationApp.selectedOrgType === 'parti\_politique' ? generatePartiPolitiqueSection() : ''}

`;

container.innerHTML = recapHTML;

// Mettre à jour les statistiques si rapport d'anomalies actif

if (OrganisationApp.rapportAnomalies.enabled) {

updateRapportStatistiques();

}

}

/\*\*

\* Générer la composition avec indicateur de qualité

\*/

function generateCompositionWithQuality(formData) {

const totalAdherents = OrganisationApp.adherents.length;

const adherentsValides = OrganisationApp.rapportAnomalies.enabled ?

OrganisationApp.rapportAnomalies.adherentsValides : totalAdherents;

const adherentsAnomalies = OrganisationApp.rapportAnomalies.enabled ?

OrganisationApp.rapportAnomalies.adherentsAvecAnomalies : 0;

const qualiteStatut = getQualiteStatut();

const qualiteBadge = getQualiteBadgeClass(qualiteStatut);

const qualiteLabel = getQualiteLabel(qualiteStatut);

return `

<table class="table table-sm table-borderless">

<tr>

<td><strong>Fondateurs :</strong></td>

<td>

<span class="badge bg-success">${OrganisationApp.fondateurs.length}</span>

</td>

</tr>

<tr>

<td><strong>Adhérents :</strong></td>

<td>

<span class="badge bg-primary">${totalAdherents}</span>

${adherentsAnomalies > 0 ? `

<small class="text-muted d-block mt-1">

<i class="fas fa-check text-success me-1"></i>${adherentsValides} valides

<i class="fas fa-exclamation-triangle text-warning ms-2 me-1"></i>${adherentsAnomalies} anomalies

</small>

` : ''}

</td>

</tr>

<tr>

<td><strong>Qualité :</strong></td>

<td>

<span class="badge ${qualiteBadge}">${qualiteLabel}</span>

</td>

</tr>

</table>

`;

}

/\*\*

\* Générer le statut de validation avec qualité

\*/

function generateValidationStatusWithQuality() {

const qualiteStatut = getQualiteStatut();

const isQualityGood = ['excellent', 'bon'].includes(qualiteStatut);

return `

<div class="card border-0 ${isQualityGood ? 'bg-light' : 'bg-warning-subtle'}">

<div class="card-body">

<div class="row align-items-center">

<div class="col-md-8">

<h6 class="text-dark mb-3">

<i class="fas fa-clipboard-check me-2"></i>

Statut de validation ${OrganisationApp.rapportAnomalies.enabled ? '& qualité' : ''}

</h6>

<div class="row">

<div class="col-md-3">

<div class="text-center">

<i class="fas fa-check-circle fa-2x ${validateStep1() ? 'text-success' : 'text-muted'}"></i>

<div class="small mt-1">Type organisation</div>

</div>

</div>

<div class="col-md-3">

<div class="text-center">

<i class="fas fa-check-circle fa-2x ${validateStep3() ? 'text-success' : 'text-muted'}"></i>

<div class="small mt-1">Demandeur</div>

</div>

</div>

<div class="col-md-3">

<div class="text-center">

<i class="fas fa-check-circle fa-2x ${validateStep6() ? 'text-success' : 'text-muted'}"></i>

<div class="small mt-1">Fondateurs</div>

</div>

</div>

<div class="col-md-3">

<div class="text-center">

<i class="fas fa-check-circle fa-2x ${validateStep7() ? 'text-success' : 'text-muted'}"></i>

<div class="small mt-1">Adhérents</div>

</div>

</div>

</div>

</div>

${OrganisationApp.rapportAnomalies.enabled ? `

<div class="col-md-4">

<div class="text-center">

<div class="mb-2">

<span class="badge ${getQualiteBadgeClass(qualiteStatut)} fs-6">

${getQualiteLabel(qualiteStatut)}

</span>

</div>

<small class="text-muted">

${OrganisationApp.rapportAnomalies.adherentsAvecAnomalies} anomalie(s) détectée(s)

</small>

</div>

</div>

` : ''}

</div>

${!isQualityGood && OrganisationApp.rapportAnomalies.enabled ? `

<div class="alert alert-warning mt-3 mb-0">

<i class="fas fa-exclamation-triangle me-2"></i>

<strong>Attention :</strong> Des anomalies ont été détectées dans votre dossier.

Un rapport détaillé sera transmis avec votre demande pour faciliter le traitement.

</div>

` : ''}

</div>

</div>

`;

}

/\*\*

\* Générer la section spéciale pour parti politique

\*/

function generatePartiPolitiqueSection() {

const professionsExclues = OrganisationApp.adherents.filter(a =>

a.hasAnomalies && a.anomalies.some(an => an.type === 'profession\_exclue\_parti')

);

if (professionsExclues.length === 0) {

return `

<div class="alert alert-success mt-4">

<h6><i class="fas fa-shield-alt me-2"></i>Conformité Parti Politique</h6>

<p class="mb-0">✅ Aucune profession exclue détectée. Votre parti politique respecte les exigences légales gabonaises.</p>

</div>

`;

}

return `

<div class="alert alert-danger mt-4">

<h6><i class="fas fa-exclamation-triangle me-2"></i>Attention - Professions Exclues Détectées</h6>

<p><strong>${professionsExclues.length} membre(s)</strong> avec des professions normalement exclues pour les partis politiques :</p>

<ul class="mb-2">

${professionsExclues.map(p => `

<li><strong>${p.nom} ${p.prenom}</strong> - ${p.profession}</li>

`).join('')}

</ul>

<p class="mb-0"><small class="text-muted">

Ces membres ont été conservés avec une anomalie critique. Une régularisation sera nécessaire.

</small></p>

</div>

`;

}

/\*\*

\* Générer la section anomalies pour le récapitulatif

\*/

function generateAnomaliesRecapSection() {

if (!OrganisationApp.rapportAnomalies.enabled || OrganisationApp.rapportAnomalies.anomalies.length === 0) {

return '';

}

const stats = OrganisationApp.rapportAnomalies.statistiques;

const total = OrganisationApp.rapportAnomalies.adherentsAvecAnomalies;

return `

<div class="card border-warning shadow-sm mt-4">

<div class="card-header bg-warning text-dark">

<h6 class="mb-0">

<i class="fas fa-exclamation-triangle me-2"></i>

Rapport d'anomalies détectées

</h6>

</div>

<div class="card-body">

<div class="row">

<div class="col-md-6">

<p class="mb-2">

<strong>${total} adhérent(s)</strong> présentent des anomalies nécessitant une attention :

</p>

<ul class="list-unstyled">

${stats.critique > 0 ? `<li><span class="badge bg-danger me-2">${stats.critique}</span>Critique(s) - Action immédiate</li>` : ''}

${stats.majeure > 0 ? `<li><span class="badge bg-warning me-2">${stats.majeure}</span>Majeure(s) - Sous 48h</li>` : ''}

${stats.mineure > 0 ? `<li><span class="badge bg-info me-2">${stats.mineure}</span>Mineure(s) - Recommandée</li>` : ''}

</ul>

</div>

<div class="col-md-6 text-end">

<button type="button" class="btn btn-outline-primary me-2" onclick="previewRapportAnomalies()">

<i class="fas fa-eye me-1"></i>Prévisualiser

</button>

<button type="button" class="btn btn-success" onclick="downloadRapportAnomalies()">

<i class="fas fa-download me-1"></i>Télécharger

</button>

</div>

</div>

<div class="mt-3">

<small class="text-muted">

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

Ce rapport sera automatiquement transmis avec votre dossier pour faciliter le traitement.

</small>

</div>

</div>

</div>

`;

}

/\*\*

\* Mettre à jour les statistiques du rapport

\*/

function updateRapportStatistiques() {

if (!OrganisationApp.rapportAnomalies.enabled) return;

// Recalculer les statistiques en temps réel

OrganisationApp.rapportAnomalies.adherentsValides = OrganisationApp.adherents.filter(a => !a.hasAnomalies).length;

OrganisationApp.rapportAnomalies.adherentsAvecAnomalies = OrganisationApp.adherents.filter(a => a.hasAnomalies).length;

console.log('📊 Statistiques rapport mises à jour:', OrganisationApp.rapportAnomalies);

}

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

// 8. RAPPORT D'ANOMALIES COMPLET

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

/\*\*

\* Générer le rapport d'anomalies complet

\*/

function generateRapportAnomalies() {

console.log('📋 Génération du rapport d\'anomalies - Version 1.2');

if (!OrganisationApp.rapportAnomalies.enabled || OrganisationApp.rapportAnomalies.anomalies.length === 0) {

console.log('ℹ️ Aucune anomalie détectée, pas de rapport à générer');

return null;

}

const rapport = {

metadata: generateRapportMetadata(),

organisation: generateRapportOrganisationInfo(),

statistiques: generateRapportStatistiques(),

anomalies: generateRapportAnomaliesDetail(),

recommandations: getRecommandationsAnomalies(),

signature: generateRapportSignature()

};

console.log('✅ Rapport d\'anomalies généré avec succès');

return rapport;

}

/\*\*

\* Générer les métadonnées du rapport

\*/

function generateRapportMetadata() {

return {

titre: 'Rapport d\'Anomalies - Importation Adhérents',

version: OrganisationApp.rapportAnomalies.version,

genereAt: OrganisationApp.rapportAnomalies.genereAt || new Date().toISOString(),

generePar: 'Système PNGDI',

typeDocument: 'RAPPORT\_ANOMALIES\_ADHERENTS',

format: 'JSON/HTML',

encodage: 'UTF-8',

langue: 'fr-GA'

};

}

/\*\*

\* Générer les informations de l'organisation pour le rapport

\*/

function generateRapportOrganisationInfo() {

const formData = collectFormData();

return {

typeOrganisation: OrganisationApp.selectedOrgType,

typeLabel: getOrganizationTypeLabel(OrganisationApp.selectedOrgType),

nomOrganisation: formData.org\_nom || 'Non renseigné',

sigleOrganisation: formData.org\_sigle || null,

demandeurPrincipal: {

nom: `${formData.demandeur\_civilite || ''} ${formData.demandeur\_nom || ''} ${formData.demandeur\_prenom || ''}`.trim(),

nip: formData.demandeur\_nip || 'Non renseigné',

email: formData.demandeur\_email || 'Non renseigné',

telephone: formData.demandeur\_telephone || 'Non renseigné',

role: formData.demandeur\_role || 'Non renseigné'

},

exigencesMinimales: {

fondateursMin: OrganisationApp.config.orgRequirements[OrganisationApp.selectedOrgType]?.minFondateurs || 3,

adherentsMin: OrganisationApp.config.orgRequirements[OrganisationApp.selectedOrgType]?.minAdherents || 10

}

};

}

/\*\*

\* Générer les statistiques détaillées

\*/

function generateRapportStatistiques() {

const totalAdherents = OrganisationApp.adherents.length;

const totalAnomalies = OrganisationApp.rapportAnomalies.anomalies.length;

const adherentsAvecAnomalies = OrganisationApp.rapportAnomalies.adherentsAvecAnomalies;

const adherentsValides = OrganisationApp.rapportAnomalies.adherentsValides;

// Statistiques par niveau d'anomalie

const statsNiveaux = OrganisationApp.rapportAnomalies.statistiques;

// Statistiques par type d'anomalie

const statsTypes = {};

OrganisationApp.rapportAnomalies.anomalies.forEach(anomalie => {

if (!statsTypes[anomalie.type]) {

statsTypes[anomalie.type] = {

count: 0,

label: anomalie.label,

level: anomalie.level

};

}

statsTypes[anomalie.type].count++;

});

// Calcul des pourcentages

const pourcentageValides = totalAdherents > 0 ? ((adherentsValides / totalAdherents) \* 100).toFixed(1) : 0;

const pourcentageAnomalies = totalAdherents > 0 ? ((adherentsAvecAnomalies / totalAdherents) \* 100).toFixed(1) : 0;

return {

resume: {

totalAdherentsImportes: totalAdherents,

adherentsValides: adherentsValides,

adherentsAvecAnomalies: adherentsAvecAnomalies,

totalAnomaliesDetectees: totalAnomalies,

pourcentageValides: parseFloat(pourcentageValides),

pourcentageAnomalies: parseFloat(pourcentageAnomalies),

qualiteGlobale: getQualiteStatut()

},

parNiveau: {

critique: statsNiveaux.critique,

majeure: statsNiveaux.majeure,

mineure: statsNiveaux.mineure

},

parType: statsTypes,

evaluation: {

statutQualite: getQualiteStatut(),

niveauRisque: statsNiveaux.critique > 0 ? 'ÉLEVÉ' :

statsNiveaux.majeure > 0 ? 'MOYEN' : 'FAIBLE',

actionRequise: statsNiveaux.critique > 0 ? 'IMMÉDIATE' :

statsNiveaux.majeure > 0 ? 'SOUS 48H' : 'OPTIONNELLE'

}

};

}

/\*\*

\* Générer le détail des anomalies avec groupement

\*/

function generateRapportAnomaliesDetail() {

const anomalies = OrganisationApp.rapportAnomalies.anomalies;

// Grouper par niveau de gravité

const parNiveau = {

critique: anomalies.filter(a => a.level === 'critique'),

majeure: anomalies.filter(a => a.level === 'majeure'),

mineure: anomalies.filter(a => a.level === 'mineure')

};

// Grouper par type d'anomalie

const parType = {};

anomalies.forEach(anomalie => {

if (!parType[anomalie.type]) {

parType[anomalie.type] = [];

}

parType[anomalie.type].push(anomalie);

});

// Générer le détail formaté

const detailFormate = anomalies.map(anomalie => ({

id: anomalie.id,

adherent: {

nom: anomalie.adherentNom,

nip: anomalie.adherentNip,

ligne: anomalie.adherentLigne

},

anomalie: {

type: anomalie.type,

level: anomalie.level,

label: anomalie.label,

description: anomalie.description,

details: anomalie.details,

detecteAt: anomalie.detecteAt

},

resolution: {

priorite: anomalie.level === 'critique' ? 1 :

anomalie.level === 'majeure' ? 2 : 3,

actionSuggere: getActionSuggereePourAnomalie(anomalie.type),

delaiRecommande: anomalie.level === 'critique' ? '24h' :

anomalie.level === 'majeure' ? '72h' : '1 semaine'

}

}));

return {

total: anomalies.length,

parNiveau: parNiveau,

parType: parType,

detailComplet: detailFormate,

ordreTraitement: detailFormate.sort((a, b) => a.resolution.priorite - b.resolution.priorite)

};

}

/\*\*

\* Obtenir l'action suggérée pour un type d'anomalie

\*/

function getActionSuggereePourAnomalie(type) {

const actions = {

'nip\_invalide': 'Vérifier auprès des services d\'état civil',

'telephone\_invalide': 'Corriger le format du numéro de téléphone',

'email\_invalide': 'Corriger l\'adresse email',

'champs\_incomplets': 'Compléter les informations manquantes',

'membre\_existant': 'Contacter le membre pour régularisation',

'profession\_exclue\_parti': 'Exclure le membre ou changer le type d\'organisation',

'doublon\_fichier': 'Supprimer ou fusionner les doublons',

'format\_donnees': 'Vérifier et corriger le format des données'

};

return actions[type] || 'Vérifier et corriger les données';

}

/\*\*

\* Générer la signature du rapport

\*/

function generateRapportSignature() {

return {

systeme: 'PNGDI - Plateforme Nationale de Gestion des Déclarations d\'Intentions',

version: '1.2',

module: 'Import Adhérents avec Gestion Anomalies',

checksum: generateRapportChecksum(),

timestamp: Date.now(),

format: 'Rapport JSON structuré compatible email/inbox'

};

}

/\*\*

\* Générer un checksum simple pour le rapport

\*/

function generateRapportChecksum() {

const data = JSON.stringify({

anomalies: OrganisationApp.rapportAnomalies.anomalies.length,

timestamp: OrganisationApp.rapportAnomalies.genereAt,

version: OrganisationApp.rapportAnomalies.version

});

// Simple hash basé sur le contenu

let hash = 0;

for (let i = 0; i < data.length; i++) {

const char = data.charCodeAt(i);

hash = ((hash << 5) - hash) + char;

hash = hash & hash; // Convert to 32-bit integer

}

return Math.abs(hash).toString(16);

}

/\*\*

\* Prévisualiser le rapport d'anomalies en modal

\*/

function previewRapportAnomalies() {

console.log('👁️ Prévisualisation du rapport d\'anomalies');

const rapport = generateRapportAnomalies();

if (!rapport) {

showNotification('Aucun rapport d\'anomalies à prévisualiser', 'info');

return;

}

// Créer et afficher la modal

createRapportAnomaliesModal(rapport);

}

/\*\*

\* Créer la modal de prévisualisation du rapport

\*/

function createRapportAnomaliesModal(rapport) {

const modalId = 'rapportAnomaliesModal';

// Supprimer l'ancienne modal si elle existe

const existingModal = document.getElementById(modalId);

if (existingModal) {

existingModal.remove();

}

const stats = rapport.statistiques;

const anomalies = rapport.anomalies;

const modalHTML = `

<div class="modal fade" id="${modalId}" tabindex="-1" aria-labelledby="${modalId}Label" aria-hidden="true">

<div class="modal-dialog modal-xl modal-dialog-scrollable">

<div class="modal-content">

<div class="modal-header bg-primary text-white">

<h5 class="modal-title" id="${modalId}Label">

<i class="fas fa-exclamation-triangle me-2"></i>

${rapport.metadata.titre}

</h5>

<button type="button" class="btn-close btn-close-white" data-bs-dismiss="modal" aria-label="Close"></button>

</div>

<div class="modal-body">

${generateRapportModalContent(rapport)}

</div>

<div class="modal-footer bg-light">

<div class="d-flex justify-content-between w-100 align-items-center">

<div class="text-muted small">

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

Généré le ${new Date(rapport.metadata.genereAt).toLocaleDateString('fr-FR', {

year: 'numeric', month: 'long', day: 'numeric', hour: '2-digit', minute: '2-digit'

})}

</div>

<div>

<button type="button" class="btn btn-outline-secondary me-2" data-bs-dismiss="modal">

<i class="fas fa-times me-1"></i>Fermer

</button>

<button type="button" class="btn btn-success me-2" onclick="downloadRapportAnomalies()">

<i class="fas fa-download me-1"></i>Télécharger JSON

</button>

<button type="button" class="btn btn-primary" onclick="exportRapportHTML()">

<i class="fas fa-file-export me-1"></i>Exporter HTML

</button>

</div>

</div>

</div>

</div>

</div>

</div>

`;

// Ajouter la modal au DOM

document.body.insertAdjacentHTML('beforeend', modalHTML);

// Afficher la modal

const modal = new bootstrap.Modal(document.getElementById(modalId));

modal.show();

// Nettoyer après fermeture

modal.\_element.addEventListener('hidden.bs.modal', () => {

modal.\_element.remove();

});

}

/\*\*

\* Générer le contenu de la modal

\*/

function generateRapportModalContent(rapport) {

const stats = rapport.statistiques;

const anomalies = rapport.anomalies;

return `

<!-- En-tête du rapport -->

<div class="card border-0 shadow-sm mb-4">

<div class="card-header bg-gradient-primary text-white">

<div class="row align-items-center">

<div class="col-md-8">

<h6 class="mb-1">

<i class="fas fa-building me-2"></i>

${rapport.organisation.nomOrganisation}

</h6>

<small class="opacity-75">

${rapport.organisation.typeLabel} | Demandeur: ${rapport.organisation.demandeurPrincipal.nom}

</small>

</div>

<div class="col-md-4 text-end">

<span class="badge ${getQualiteBadgeClass(stats.resume.qualiteGlobale)} fs-6">

${getQualiteLabel(stats.resume.qualiteGlobale)}

</span>

</div>

</div>

</div>

</div>

<!-- Statistiques résumées -->

<div class="row mb-4">

<div class="col-md-3">

<div class="card border-0 bg-light h-100">

<div class="card-body text-center">

<h3 class="text-primary mb-1">${stats.resume.totalAdherentsImportes}</h3>

<small class="text-muted">Total importés</small>

</div>

</div>

</div>

<div class="col-md-3">

<div class="card border-0 bg-success-subtle h-100">

<div class="card-body text-center">

<h3 class="text-success mb-1">${stats.resume.adherentsValides}</h3>

<small class="text-muted">Valides (${stats.resume.pourcentageValides}%)</small>

</div>

</div>

</div>

<div class="col-md-3">

<div class="card border-0 bg-warning-subtle h-100">

<div class="card-body text-center">

<h3 class="text-warning mb-1">${stats.resume.adherentsAvecAnomalies}</h3>

<small class="text-muted">Avec anomalies (${stats.resume.pourcentageAnomalies}%)</small>

</div>

</div>

</div>

<div class="col-md-3">

<div class="card border-0 bg-danger-subtle h-100">

<div class="card-body text-center">

<h3 class="text-danger mb-1">${stats.resume.totalAnomaliesDetectees}</h3>

<small class="text-muted">Anomalies total</small>

</div>

</div>

</div>

</div>

<!-- Répartition par niveau -->

<div class="card border-0 shadow-sm mb-4">

<div class="card-header">

<h6 class="mb-0">

<i class="fas fa-chart-bar me-2"></i>

Répartition par niveau de gravité

</h6>

</div>

<div class="card-body">

<div class="row">

<div class="col-md-4">

<div class="d-flex align-items-center p-3 bg-danger-subtle rounded">

<div class="me-3">

<span class="badge bg-danger fs-6">${stats.parNiveau.critique}</span>

</div>

<div>

<strong class="text-danger">Critique</strong><br>

<small class="text-muted">Action immédiate</small>

</div>

</div>

</div>

<div class="col-md-4">

<div class="d-flex align-items-center p-3 bg-warning-subtle rounded">

<div class="me-3">

<span class="badge bg-warning fs-6">${stats.parNiveau.majeure}</span>

</div>

<div>

<strong class="text-warning">Majeure</strong><br>

<small class="text-muted">Sous 48h</small>

</div>

</div>

</div>

<div class="col-md-4">

<div class="d-flex align-items-center p-3 bg-info-subtle rounded">

<div class="me-3">

<span class="badge bg-info fs-6">${stats.parNiveau.mineure}</span>

</div>

<div>

<strong class="text-info">Mineure</strong><br>

<small class="text-muted">Recommandée</small>

</div>

</div>

</div>

</div>

</div>

</div>

<!-- Tableau des anomalies -->

<div class="card border-0 shadow-sm mb-4">

<div class="card-header d-flex justify-content-between align-items-center">

<h6 class="mb-0">

<i class="fas fa-list me-2"></i>

Détail des anomalies (${anomalies.total})

</h6>

<div class="btn-group" role="group">

<button type="button" class="btn btn-outline-secondary btn-sm active" onclick="filterAnomalies('all')">

Toutes

</button>

<button type="button" class="btn btn-outline-danger btn-sm" onclick="filterAnomalies('critique')">

Critiques

</button>

<button type="button" class="btn btn-outline-warning btn-sm" onclick="filterAnomalies('majeure')">

Majeures

</button>

<button type="button" class="btn btn-outline-info btn-sm" onclick="filterAnomalies('mineure')">

Mineures

</button>

</div>

</div>

<div class="card-body p-0">

<div class="table-responsive" style="max-height: 400px; overflow-y: auto;">

<table class="table table-hover mb-0" id="anomaliesTable">

<thead class="table-light sticky-top">

<tr>

<th>Adhérent</th>

<th>NIP</th>

<th>Ligne</th>

<th>Niveau</th>

<th>Anomalie</th>

<th>Action suggérée</th>

<th>Délai</th>

</tr>

</thead>

<tbody>

${generateAnomaliesTableRows(anomalies.ordreTraitement)}

</tbody>

</table>

</div>

</div>

</div>

<!-- Recommandations -->

${rapport.recommandations.length > 0 ? `

<div class="card border-0 shadow-sm">

<div class="card-header">

<h6 class="mb-0">

<i class="fas fa-lightbulb me-2"></i>

Recommandations (${rapport.recommandations.length})

</h6>

</div>

<div class="card-body">

${rapport.recommandations.map(rec => `

<div class="alert alert-${getRecommandationAlertClass(rec.type)} d-flex align-items-start">

<i class="fas ${getRecommandationIcon(rec.type)} me-3 mt-1"></i>

<div>

<strong>${rec.type.toUpperCase()} :</strong> ${rec.message}

</div>

</div>

`).join('')}

</div>

</div>

` : ''}

`;

}

/\*\*

\* Générer les lignes du tableau des anomalies

\*/

function generateAnomaliesTableRows(anomalies) {

return anomalies.map(anomalie => `

<tr class="anomalie-row anomalie-${anomalie.anomalie.level}" data-level="${anomalie.anomalie.level}">

<td>

<strong>${anomalie.adherent.nom}</strong>

</td>

<td>

<code class="small">${anomalie.adherent.nip}</code>

</td>

<td>

<span class="badge bg-secondary">${anomalie.adherent.ligne || 'N/A'}</span>

</td>

<td>

<span class="badge bg-${getLevelBadgeColor(anomalie.anomalie.level)}">

${anomalie.anomalie.level.toUpperCase()}

</span>

</td>

<td>

<div>

<strong class="small">${anomalie.anomalie.label}</strong>

<br>

<small class="text-muted">${anomalie.anomalie.description}</small>

${anomalie.anomalie.details ? `<br><small class="text-warning">Détails: ${anomalie.anomalie.details}</small>` : ''}

</div>

</td>

<td>

<small>${anomalie.resolution.actionSuggere}</small>

</td>

<td>

<span class="badge bg-outline-${getLevelBadgeColor(anomalie.anomalie.level)}">

${anomalie.resolution.delaiRecommande}

</span>

</td>

</tr>

`).join('');

}

/\*\*

\* Filtrer les anomalies par niveau

\*/

function filterAnomalies(level) {

const table = document.getElementById('anomaliesTable');

if (!table) return;

const rows = table.querySelectorAll('.anomalie-row');

const buttons = table.closest('.card').querySelectorAll('.btn-group .btn');

// Mettre à jour les boutons

buttons.forEach(btn => btn.classList.remove('active'));

event.target.classList.add('active');

// Filtrer les lignes

rows.forEach(row => {

if (level === 'all' || row.dataset.level === level) {

row.style.display = '';

} else {

row.style.display = 'none';

}

});

}

/\*\*

\* Télécharger le rapport au format JSON

\*/

function downloadRapportAnomalies() {

const rapport = generateRapportAnomalies();

if (!rapport) {

showNotification('Aucun rapport d\'anomalies à télécharger', 'info');

return;

}

const rapportJSON = JSON.stringify(rapport, null, 2);

const blob = new Blob([rapportJSON], { type: 'application/json;charset=utf-8;' });

const link = document.createElement('a');

if (link.download !== undefined) {

const url = URL.createObjectURL(blob);

const fileName = `rapport\_anomalies\_${rapport.organisation.nomOrganisation.replace(/[^a-z0-9]/gi, '\_')}\_${new Date().toISOString().split('T')[0]}.json`;

link.setAttribute('href', url);

link.setAttribute('download', fileName);

link.style.visibility = 'hidden';

document.body.appendChild(link);

link.click();

document.body.removeChild(link);

showNotification('Rapport d\'anomalies téléchargé avec succès', 'success');

}

}

/\*\*

\* Exporter le rapport en HTML

\*/

function exportRapportHTML() {

const htmlContent = generateRapportAnomaliesHTML();

if (!htmlContent) {

showNotification('Impossible d\'exporter le rapport HTML', 'danger');

return;

}

const blob = new Blob([htmlContent], { type: 'text/html;charset=utf-8;' });

const link = document.createElement('a');

if (link.download !== undefined) {

const url = URL.createObjectURL(blob);

const fileName = `rapport\_anomalies\_${OrganisationApp.formData.org\_nom ? OrganisationApp.formData.org\_nom.replace(/[^a-z0-9]/gi, '\_') : 'organisation'}\_${new Date().toISOString().split('T')[0]}.html`;

link.setAttribute('href', url);

link.setAttribute('download', fileName);

link.style.visibility = 'hidden';

document.body.appendChild(link);

link.click();

document.body.removeChild(link);

showNotification('Rapport HTML exporté avec succès', 'success');

}

}

/\*\*

\* Générer le rapport au format HTML pour email

\*/

function generateRapportAnomaliesHTML() {

const rapport = generateRapportAnomalies();

if (!rapport) return null;

const stats = rapport.statistiques;

const anomalies = rapport.anomalies;

return `

<!DOCTYPE html>

<html lang="fr">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>${rapport.metadata.titre}</title>

<style>

body { font-family: Arial, sans-serif; line-height: 1.6; color: #333; max-width: 800px; margin: 0 auto; padding: 20px; }

.header { background: #f8f9fa; padding: 20px; border-radius: 8px; margin-bottom: 20px; }

.stats { display: grid; grid-template-columns: repeat(auto-fit, minmax(200px, 1fr)); gap: 15px; margin: 20px 0; }

.stat-card { background: #fff; border: 1px solid #dee2e6; border-radius: 6px; padding: 15px; text-align: center; }

.badge { padding: 4px 8px; border-radius: 4px; font-size: 0.85em; font-weight: bold; }

.badge-danger { background: #dc3545; color: white; }

.badge-warning { background: #ffc107; color: #212529; }

.badge-info { background: #17a2b8; color: white; }

.badge-success { background: #28a745; color: white; }

.anomalie-item { border-left: 4px solid #dc3545; padding: 10px; margin: 10px 0; background: #f8f9fa; }

.anomalie-critique { border-left-color: #dc3545; }

.anomalie-majeure { border-left-color: #ffc107; }

.anomalie-mineure { border-left-color: #17a2b8; }

.table { width: 100%; border-collapse: collapse; margin: 15px 0; }

.table th, .table td { padding: 8px 12px; text-align: left; border-bottom: 1px solid #dee2e6; }

.table th { background: #f8f9fa; font-weight: bold; }

.recommandation { background: #e7f3ff; border: 1px solid #b3d9ff; border-radius: 6px; padding: 15px; margin: 10px 0; }

</style>

</head>

<body>

<div class="header">

<h1>${rapport.metadata.titre}</h1>

<p><strong>Organisation :</strong> ${rapport.organisation.nomOrganisation} (${rapport.organisation.typeLabel})</p>

<p><strong>Demandeur :</strong> ${rapport.organisation.demandeurPrincipal.nom}</p>

<p><strong>Généré le :</strong> ${new Date(rapport.metadata.genereAt).toLocaleDateString('fr-FR', {

year: 'numeric', month: 'long', day: 'numeric', hour: '2-digit', minute: '2-digit'

})}</p>

</div>

<div class="stats">

<div class="stat-card">

<h3>${stats.resume.totalAdherentsImportes}</h3>

<p>Adhérents importés</p>

</div>

<div class="stat-card">

<h3 style="color: #28a745;">${stats.resume.adherentsValides}</h3>

<p>Valides (${stats.resume.pourcentageValides}%)</p>

</div>

<div class="stat-card">

<h3 style="color: #dc3545;">${stats.resume.adherentsAvecAnomalies}</h3>

<p>Avec anomalies (${stats.resume.pourcentageAnomalies}%)</p>

</div>

<div class="stat-card">

<h3>${stats.resume.totalAnomaliesDetectees}</h3>

<p>Anomalies détectées</p>

</div>

</div>

<h2>📊 Répartition par niveau de gravité</h2>

<table class="table">

<thead>

<tr>

<th>Niveau</th>

<th>Nombre</th>

<th>Action requise</th>

</tr>

</thead>

<tbody>

<tr>

<td><span class="badge badge-danger">Critique</span></td>

<td>${stats.parNiveau.critique}</td>

<td>Correction immédiate</td>

</tr>

<tr>

<td><span class="badge badge-warning">Majeure</span></td>

<td>${stats.parNiveau.majeure}</td>

<td>Correction sous 48h</td>

</tr>

<tr>

<td><span class="badge badge-info">Mineure</span></td>

<td>${stats.parNiveau.mineure}</td>

<td>Correction recommandée</td>

</tr>

</tbody>

</table>

<h2>📋 Détail des anomalies par ordre de priorité</h2>

${anomalies.ordreTraitement.map(anomalie => `

<div class="anomalie-item anomalie-${anomalie.anomalie.level}">

<h4>${anomalie.adherent.nom} <span class="badge badge-${anomalie.anomalie.level === 'critique' ? 'danger' : anomalie.anomalie.level === 'majeure' ? 'warning' : 'info'}">${anomalie.anomalie.level.toUpperCase()}</span></h4>

<p><strong>NIP :</strong> ${anomalie.adherent.nip} | <strong>Ligne :</strong> ${anomalie.adherent.ligne || 'N/A'}</p>

<p><strong>Anomalie :</strong> ${anomalie.anomalie.label}</p>

<p><strong>Description :</strong> ${anomalie.anomalie.description}</p>

${anomalie.anomalie.details ? `<p><strong>Détails :</strong> ${anomalie.anomalie.details}</p>` : ''}

<p><strong>Action suggérée :</strong> ${anomalie.resolution.actionSuggere}</p>

<p><strong>Délai recommandé :</strong> ${anomalie.resolution.delaiRecommande}</p>

</div>

`).join('')}

<h2>💡 Recommandations</h2>

${rapport.recommandations.map(rec => `

<div class="recommandation">

<strong>${rec.type.toUpperCase()} :</strong> ${rec.message}

</div>

`).join('')}

<div class="header" style="margin-top: 30px; text-align: center; font-size: 0.9em; color: #666;">

<p>Rapport généré automatiquement par ${rapport.signature.systeme}</p>

<p>Version ${rapport.signature.version} | Checksum: ${rapport.signature.checksum}</p>

</div>

</body>

</html>

`;

}

/\*\*

\* Fonctions utilitaires pour l'affichage

\*/

function getLevelBadgeColor(level) {

const colors = {

'critique': 'danger',

'majeure': 'warning',

'mineure': 'info'

};

return colors[level] || 'secondary';

}

function getRecommandationAlertClass(type) {

const classes = {

'urgent': 'danger',

'important': 'warning',

'conseil': 'info'

};

return classes[type] || 'secondary';

}

function getRecommandationIcon(type) {

const icons = {

'urgent': 'fa-exclamation-triangle',

'important': 'fa-exclamation-circle',

'conseil': 'fa-lightbulb'

};

return icons[type] || 'fa-info-circle';

}

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

// 9. SAUVEGARDE ET COLLECTE DE DONNÉES

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

/\*\*

\* Sauvegarder les données de l'étape actuelle

\*/

function saveCurrentStepData() {

const stepData = collectStepData(OrganisationApp.currentStep);

OrganisationApp.formData[`step${OrganisationApp.currentStep}`] = stepData;

// Sauvegarder automatiquement

autoSave();

}

/\*\*

\* Collecter les données d'une étape

\*/

function collectStepData(stepNumber) {

const stepElement = document.getElementById(`step${stepNumber}`);

if (!stepElement) return {};

const data = {};

stepElement.querySelectorAll('input, select, textarea').forEach(field => {

if (!field.name && !field.id) return;

const key = field.name || field.id;

if (field.type === 'checkbox' || field.type === 'radio') {

if (field.checked) {

data[key] = field.value;

}

} else if (field.type !== 'file') {

data[key] = field.value;

}

});

return data;

}

/\*\*

\* Collecter toutes les données du formulaire

\*/

function collectFormData() {

const data = {};

// Parcourir toutes les étapes

for (let i = 1; i <= OrganisationApp.totalSteps; i++) {

const stepData = collectStepData(i);

Object.assign(data, stepData);

}

return data;

}

/\*\*

\* Sauvegarde automatique

\*/

function autoSave() {

try {

const saveData = {

formData: collectFormData(),

fondateurs: OrganisationApp.fondateurs,

adherents: OrganisationApp.adherents,

documents: Object.keys(OrganisationApp.documents),

// Inclure le rapport d'anomalies

rapportAnomaliesAdherents: OrganisationApp.rapportAnomalies.enabled ? {

enabled: true,

adherentsValides: OrganisationApp.rapportAnomalies.adherentsValides,

adherentsAvecAnomalies: OrganisationApp.rapportAnomalies.adherentsAvecAnomalies,

anomalies: OrganisationApp.rapportAnomalies.anomalies,

statistiques: OrganisationApp.rapportAnomalies.statistiques,

genereAt: OrganisationApp.rapportAnomalies.genereAt,

version: OrganisationApp.rapportAnomalies.version

} : { enabled: false },

currentStep: OrganisationApp.currentStep,

selectedOrgType: OrganisationApp.selectedOrgType,

timestamp: Date.now(),

version: '1.2'

};

localStorage.setItem('pngdi\_organisation\_draft', JSON.stringify(saveData));

updateSaveIndicator('success');

console.log('💾 Sauvegarde automatique v1.2 réussie');

} catch (error) {

console.error('Erreur sauvegarde:', error);

updateSaveIndicator('error');

}

}

/\*\*

\* Charger les données sauvegardées

\*/

function loadSavedData() {

try {

const saved = localStorage.getItem('pngdi\_organisation\_draft');

if (saved) {

const data = JSON.parse(saved);

// Vérifier que les données ne sont pas trop anciennes (7 jours)

if (Date.now() - data.timestamp < 7 \* 24 \* 60 \* 60 \* 1000) {

if (confirm('Des données sauvegardées ont été trouvées. Voulez-vous les restaurer ?')) {

restoreFormData(data);

showNotification('Données restaurées avec succès', 'success');

return true;

}

} else {

// Supprimer les anciennes données

localStorage.removeItem('pngdi\_organisation\_draft');

}

}

} catch (error) {

console.error('Erreur chargement données:', error);

}

return false;

}

/\*\*

\* Restaurer les données du formulaire

\*/

function restoreFormData(savedData) {

try {

// Restaurer les données existantes

OrganisationApp.currentStep = savedData.currentStep || 1;

OrganisationApp.selectedOrgType = savedData.selectedOrgType || '';

OrganisationApp.fondateurs = savedData.fondateurs || [];

OrganisationApp.adherents = savedData.adherents || [];

// Restaurer le rapport d'anomalies

if (savedData.rapportAnomaliesAdherents && savedData.rapportAnomaliesAdherents.enabled) {

OrganisationApp.rapportAnomalies = {

enabled: true,

adherentsValides: savedData.rapportAnomaliesAdherents.adherentsValides || 0,

adherentsAvecAnomalies: savedData.rapportAnomaliesAdherents.adherentsAvecAnomalies || 0,

anomalies: savedData.rapportAnomaliesAdherents.anomalies || [],

statistiques: savedData.rapportAnomaliesAdherents.statistiques || { critique: 0, majeure: 0, mineure: 0 },

genereAt: savedData.rapportAnomaliesAdherents.genereAt || null,

version: savedData.rapportAnomaliesAdherents.version || '1.2'

};

console.log('✅ Rapport d\'anomalies restauré:', OrganisationApp.rapportAnomalies);

}

// Restaurer les champs du formulaire

const formData = savedData.formData || {};

Object.keys(formData).forEach(key => {

const field = document.getElementById(key) || document.querySelector(`[name="${key}"]`);

if (field && field.type !== 'file') {

if (field.type === 'checkbox' || field.type === 'radio') {

field.checked = field.value === formData[key];

} else {

field.value = formData[key];

}

field.dispatchEvent(new Event('change', { bubbles: true }));

}

});

// Restaurer la sélection du type d'organisation

if (OrganisationApp.selectedOrgType) {

const typeCard = document.querySelector(`[data-type="${OrganisationApp.selectedOrgType}"]`);

if (typeCard) {

selectOrganizationType(typeCard);

}

}

// Mettre à jour l'affichage

updateStepDisplay();

updateFoundersList();

updateAdherentsList();

} catch (error) {

console.error('Erreur restauration données:', error);

showNotification('Erreur lors de la restauration des données', 'warning');

}

}

/\*\*

\* Mise à jour indicateur de sauvegarde

\*/

function updateSaveIndicator(status) {

const indicator = document.getElementById('save-indicator');

if (!indicator) return;

const messages = {

'saving': '<i class="fas fa-spinner fa-spin text-primary"></i> Sauvegarde...',

'success': '<i class="fas fa-check text-success"></i> Sauvegardé',

'error': '<i class="fas fa-times text-danger"></i> Erreur sauvegarde'

};

indicator.innerHTML = messages[status] || '';

if (status === 'success' || status === 'error') {

setTimeout(() => {

indicator.innerHTML = '';

}, 3000);

}

}

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

// 10. NOTIFICATIONS

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

/\*\*

\* Afficher une notification

\*/

function showNotification(message, type = 'info', duration = 5000) {

let container = document.getElementById('notification-container');

if (!container) {

container = document.createElement('div');

container.id = 'notification-container';

container.style.cssText = `

position: fixed;

top: 20px;

right: 20px;

z-index: 9999;

max-width: 400px;

`;

document.body.appendChild(container);

}

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

notification.className = `alert alert-${type} alert-dismissible fade show shadow-lg`;

notification.style.cssText = `

margin-bottom: 10px;

border: none;

border-radius: 12px;

animation: slideInRight 0.3s ease-out;

`;

const iconMap = {

'success': 'fa-check-circle',

'warning': 'fa-exclamation-triangle',

'danger': 'fa-times-circle',

'info': 'fa-info-circle'

};

notification.innerHTML = `

<i class="fas ${iconMap[type]} me-2"></i>

${message}

<button type="button" class="btn-close" data-bs-dismiss="alert"></button>

`;

container.appendChild(notification);

// Auto-suppression

setTimeout(() => {

if (notification.parentNode) {

notification.style.animation = 'slideOutRight 0.3s ease-out';

setTimeout(() => {

notification.remove();

}, 300);

}

}, duration);

}

/\*\*

\* Notification personnalisée (utilise showCustomNotification si elle existe, sinon showNotification)

\*/

function showCustomNotification(htmlContent, type = 'info', duration = 5000) {

// Si la fonction showCustomNotification n'existe pas, utiliser showNotification basique

if (typeof showCustomNotification === 'undefined') {

// Extraire le texte du HTML pour showNotification basique

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

tempDiv.innerHTML = htmlContent;

const textContent = tempDiv.textContent || tempDiv.innerText || '';

showNotification(textContent, type, duration);

return;

}

// Sinon utiliser la version complète

let container = document.getElementById('notification-container');

if (!container) {

container = document.createElement('div');

container.id = 'notification-container';

container.style.cssText = `

position: fixed;

top: 20px;

right: 20px;

z-index: 9999;

max-width: 500px;

`;

document.body.appendChild(container);

}

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

notification.className = `alert alert-${type} alert-dismissible fade show shadow-lg`;

notification.style.cssText = `

margin-bottom: 10px;

border: none;

border-radius: 12px;

animation: slideInRight 0.3s ease-out;

`;

notification.innerHTML = `

${htmlContent}

<button type="button" class="btn-close" data-bs-dismiss="alert" style="margin-top: -0.5rem;"></button>

`;

container.appendChild(notification);

// Auto-suppression

setTimeout(() => {

if (notification.parentNode) {

notification.style.animation = 'slideOutRight 0.3s ease-out';

setTimeout(() => {

notification.remove();

}, 300);

}

}, duration);

}

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

// 10.1. SYSTÈME DE DEBUG AVANCÉ

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

/\*\*

\* ✅ FONCTION AMÉLIORÉE : Afficher les erreurs détaillées du serveur

\*/

function showErrorModal(title, message, serverResponse = null, isDebug = true) {

// Construire le message d'erreur détaillé

let fullMessage = message;

if (isDebug && serverResponse) {

fullMessage += '\n\n' + '='.repeat(50);

fullMessage += '\n🔍 DEBUG - RÉPONSE SERVEUR DÉTAILLÉE :';

fullMessage += '\n' + '='.repeat(50);

// Afficher les détails de la réponse

if (typeof serverResponse === 'object') {

try {

// Si c'est un objet, l'afficher en JSON formaté

fullMessage += '\n📄 Contenu de la réponse :\n';

fullMessage += JSON.stringify(serverResponse, null, 2);

// Afficher les erreurs spécifiques si disponibles

if (serverResponse.errors) {

fullMessage += '\n\n🚨 Erreurs de validation :\n';

Object.keys(serverResponse.errors).forEach(field => {

fullMessage += `• ${field}: ${serverResponse.errors[field].join(', ')}\n`;

});

}

// Afficher les infos de debug si disponibles

if (serverResponse.debug) {

fullMessage += '\n\n🐛 Informations de debug :\n';

fullMessage += `Fichier: ${serverResponse.debug.file}\n`;

fullMessage += `Ligne: ${serverResponse.debug.line}\n`;

if (serverResponse.debug.json\_error) {

fullMessage += `Erreur JSON: ${serverResponse.debug.json\_error}\n`;

}

}

} catch (e) {

fullMessage += '\n📄 Réponse brute :\n' + String(serverResponse);

}

} else {

fullMessage += '\n📄 Réponse brute :\n' + String(serverResponse);

}

fullMessage += '\n' + '='.repeat(50);

}

// Créer et afficher le modal d'erreur amélioré

const modal = `

<div class="modal fade" id="errorDebugModal" tabindex="-1" role="dialog" style="z-index: 9999;">

<div class="modal-dialog modal-lg" role="document">

<div class="modal-content border-danger">

<div class="modal-header bg-danger text-white">

<h5 class="modal-title">

<i class="fas fa-exclamation-triangle me-2"></i>

${title}

</h5>

<button type="button" class="btn-close btn-close-white" data-bs-dismiss="modal"></button>

</div>

<div class="modal-body">

<div class="alert alert-danger">

<h6 class="alert-heading">

<i class="fas fa-bug me-2"></i>

Erreur de Soumission - Mode Debug

</h6>

<pre style="white-space: pre-wrap; font-family: monospace; font-size: 12px; max-height: 400px; overflow-y: auto; background: #f8f9fa; padding: 15px; border-radius: 5px;">${fullMessage}</pre>

</div>

<div class="mt-3">

<button type="button" class="btn btn-outline-primary btn-sm" onclick="copyDebugInfo()">

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

Copier les détails

</button>

<button type="button" class="btn btn-outline-secondary btn-sm" onclick="console.log('Debug Info:', ${JSON.stringify(serverResponse || {})})">

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

Afficher dans la console

</button>

</div>

</div>

<div class="modal-footer">

<button type="button" class="btn btn-secondary" data-bs-dismiss="modal">Fermer</button>

<button type="button" class="btn btn-primary" onclick="location.reload()">

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

Recharger la page

</button>

</div>

</div>

</div>

</div>

`;

// Supprimer le modal existant s'il y en a un

const existingModal = document.getElementById('errorDebugModal');

if (existingModal) {

existingModal.remove();

}

// Ajouter le nouveau modal

document.body.insertAdjacentHTML('beforeend', modal);

// Afficher le modal

const modalElement = new bootstrap.Modal(document.getElementById('errorDebugModal'));

modalElement.show();

// Stocker les infos de debug globalement pour la fonction de copie

window.debugInfo = fullMessage;

}

/\*\*

\* ✅ FONCTION : Copier les informations de debug

\*/

function copyDebugInfo() {

if (window.debugInfo) {

navigator.clipboard.writeText(window.debugInfo).then(() => {

showNotification('Informations de debug copiées dans le presse-papier !', 'success');

}).catch(() => {

// Fallback pour les navigateurs plus anciens

const textarea = document.createElement('textarea');

textarea.value = window.debugInfo;

document.body.appendChild(textarea);

textarea.select();

document.execCommand('copy');

document.body.removeChild(textarea);

showNotification('Informations de debug copiées !', 'success');

});

}

}

/\*\*

\* ✅ FONCTION : Analyser les données du formulaire pour diagnostic

\*/

function analyzeFormDataForDebug() {

const form = document.getElementById('organisationForm');

const formData = new FormData(form);

let totalSize = 0;

let fieldCount = 0;

let largestFields = [];

for (let [key, value] of formData.entries()) {

fieldCount++;

const size = new Blob([value]).size;

totalSize += size;

if (size > 1000) { // Champs > 1KB

largestFields.push({key, size, value: value.toString().substring(0, 50) + '...'});

}

}

console.log('=== ANALYSE FORMULAIRE DEBUG ===');

console.log('Nombre de champs:', fieldCount);

console.log('Taille totale:', (totalSize / 1024).toFixed(2) + ' KB');

console.log('Champs volumineux:', largestFields);

return {fieldCount, totalSize, largestFields};

}

// Exposer les nouvelles fonctions

window.showErrorModal = showErrorModal;

window.copyDebugInfo = copyDebugInfo;

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

// 11. SOUMISSION FINALE

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

/\*\*

\* Validation de toutes les étapes avant soumission

\*/

function validateAllSteps() {

for (let i = 1; i <= OrganisationApp.totalSteps; i++) {

if (!validateStep(i)) {

goToStep(i); // Aller à la première étape en erreur

showNotification(`Erreur à l'étape ${i}. Veuillez corriger avant de continuer.`, 'danger');

return false;

}

}

return true;

}

/\*\*

\* ✅ SOUMISSION FINALE CORRIGÉE - Avec redirection dossier\_id

\*/

async function submitForm() {

console.log('📤 Début de la soumission du formulaire avec rapport d\'anomalies...');

// Validation finale complète

if (!validateAllSteps()) {

showNotification('Veuillez corriger toutes les erreurs avant de soumettre', 'danger');

return false;

}

// Analyser les données du formulaire pour diagnostic

const analysis = analyzeFormDataForDebug();

if (analysis.fieldCount > 1000) {

console.warn('⚠️ Trop de champs:', analysis.fieldCount);

showNotification(`Attention: ${analysis.fieldCount} champs détectés (limite recommandée: 1000)`, 'warning');

}

if (analysis.totalSize > 50 \* 1024 \* 1024) { // 50MB

console.warn('⚠️ Taille importante:', (analysis.totalSize / 1024 / 1024).toFixed(2) + ' MB');

showNotification(`Attention: ${(analysis.totalSize / 1024 / 1024).toFixed(2)} MB de données à envoyer`, 'warning');

}

try {

// Afficher le loader

showGlobalLoader(true);

updateSaveIndicator('saving');

// Préparation des données

const formData = new FormData();

const data = collectFormData();

// Ajouter le token CSRF

const csrfToken = document.querySelector('meta[name="csrf-token"]')?.getAttribute('content');

if (csrfToken) {

formData.append('\_token', csrfToken);

}

// Ajouter les données de base

Object.keys(data).forEach(key => {

if (data[key] !== null && data[key] !== undefined) {

formData.append(key, data[key]);

}

});

// Ajouter les fondateurs et adhérents

formData.append('fondateurs', JSON.stringify(OrganisationApp.fondateurs));

formData.append('adherents', JSON.stringify(OrganisationApp.adherents));

// Ajouter le rapport d'anomalies si présent

if (OrganisationApp.rapportAnomalies.enabled) {

const rapport = generateRapportAnomalies();

const rapportHTML = generateRapportAnomaliesHTML();

formData.append('rapport\_anomalies\_json', JSON.stringify(rapport));

formData.append('rapport\_anomalies\_html', rapportHTML);

formData.append('has\_anomalies', 'true');

console.log('📋 Rapport d\'anomalies inclus dans la soumission');

} else {

formData.append('has\_anomalies', 'false');

}

// Ajouter les métadonnées

formData.append('selectedOrgType', OrganisationApp.selectedOrgType);

formData.append('totalFondateurs', OrganisationApp.fondateurs.length);

formData.append('totalAdherents', OrganisationApp.adherents.length);

formData.append('totalDocuments', Object.keys(OrganisationApp.documents).length);

formData.append('qualiteAdherents', getQualiteStatut());

// Ajouter les documents

Object.keys(OrganisationApp.documents).forEach(docType => {

const doc = OrganisationApp.documents[docType];

if (doc.file) {

formData.append(`documents[${docType}]`, doc.file);

}

});

console.log('📋 Données préparées pour soumission v1.2');

// Soumettre via fetch

const formElement = document.getElementById('organisationForm');

const response = await fetch(formElement.action, {

method: 'POST',

body: formData,

headers: {

'X-Requested-With': 'XMLHttpRequest'

}

});

console.log('📡 Réponse reçue du serveur:', response.status);

if (response.ok) {

const result = await response.json();

if (result.success) {

// ✅ CORRECTION PRINCIPALE : Gestion de la redirection avec dossier\_id

console.log('🎯 CORRECTION REDIRECTION - Analyse de la réponse serveur:', result);

// Succès avec message amélioré

let successMsg = '🎉 Dossier soumis avec succès !';

if (OrganisationApp.rapportAnomalies.enabled) {

successMsg += '\n📋 Le rapport d\'anomalies a été transmis automatiquement.';

}

showNotification(successMsg, 'success', 10000);

// Nettoyer les données sauvegardées

localStorage.removeItem('pngdi\_organisation\_draft');

// Désactiver le formulaire

const submitBtn = document.getElementById('submitBtn');

if (submitBtn) {

submitBtn.disabled = true;

submitBtn.innerHTML = '<i class="fas fa-check me-2"></i>Dossier soumis';

}

// ✅ REDIRECTION CORRIGÉE - Priorité aux données structurées

let redirectUrl = null;

// PRIORITÉ 1 : Utiliser redirect\_url si fourni explicitement

if (result.data && result.data.redirect\_url) {

redirectUrl = result.data.redirect\_url;

console.log('✅ REDIRECTION via result.data.redirect\_url:', redirectUrl);

// PRIORITÉ 2 : Construire avec dossier\_id si disponible

} else if (result.data && result.data.dossier\_id) {

redirectUrl = `/operator/dossiers/confirmation/${result.data.dossier\_id}`;

console.log('✅ REDIRECTION construite avec dossier\_id:', result.data.dossier\_id, '→', redirectUrl);

// PRIORITÉ 3 : Utiliser result.redirect si fourni

} else if (result.redirect) {

redirectUrl = result.redirect;

console.log('✅ REDIRECTION via result.redirect:', redirectUrl);

// ⚠️ VÉRIFICATION : S'assurer que result.redirect utilise dossier\_id

if (result.data && result.data.organisation\_id && redirectUrl.includes(result.data.organisation\_id)) {

console.warn('⚠️ ATTENTION: result.redirect semble utiliser organisation\_id au lieu de dossier\_id');

// Corriger automatiquement si dossier\_id est disponible

if (result.data.dossier\_id) {

redirectUrl = redirectUrl.replace(result.data.organisation\_id, result.data.dossier\_id);

console.log('🔧 CORRECTION AUTOMATIQUE: Remplacement organisation\_id par dossier\_id:', redirectUrl);

}

}

// PRIORITÉ 4 : URL par défaut

} else {

redirectUrl = '/operator/dossiers';

console.log('✅ REDIRECTION par défaut vers la liste des dossiers');

}

// ✅ LOG FINAL POUR DEBUG

console.log('📊 DONNÉES DE REDIRECTION FINALES:', {

redirectUrl: redirectUrl,

resultData: result.data,

resultRedirect: result.redirect,

organisationId: result.data?.organisation\_id,

dossierId: result.data?.dossier\_id,

timestamp: new Date().toISOString()

});

// Effectuer la redirection

setTimeout(() => {

console.log('🚀 REDIRECTION VERS:', redirectUrl);

window.location.href = redirectUrl;

}, 3000);

} else {

// Erreur métier avec debug détaillé

const errorMessage = result.message || 'Erreur lors de la soumission';

// Afficher d'abord la notification basique

showNotification(errorMessage, 'danger');

// Puis afficher le modal de debug avec tous les détails

showErrorModal(

'Erreur de Validation',

errorMessage,

{

success: result.success,

message: result.message,

errors: result.errors,

data: result.data,

debug: result.debug,

timestamp: new Date().toISOString(),

formAnalysis: analyzeFormDataForDebug()

},

true

);

if (result.errors) {

Object.keys(result.errors).forEach(field => {

const fieldElement = document.querySelector(`[name="${field}"]`) ||

document.getElementById(field);

if (fieldElement) {

showFieldError(fieldElement, result.errors[field][0]);

}

});

}

}

} else {

// Gestion améliorée des erreurs HTTP

let errorResponse = null;

try {

errorResponse = await response.json();

} catch (e) {

try {

errorResponse = await response.text();

} catch (e2) {

errorResponse = 'Impossible de lire la réponse du serveur';

}

}

const errorDetails = {

status: response.status,

statusText: response.statusText,

url: response.url,

headers: Object.fromEntries(response.headers.entries()),

response: errorResponse,

timestamp: new Date().toISOString()

};

// Afficher le modal de debug au lieu de juste lancer une erreur

showErrorModal(

`Erreur HTTP ${response.status}`,

`Le serveur a retourné une erreur : ${response.status} ${response.statusText}`,

errorDetails,

true

);

return; // Sortir de la fonction au lieu de throw

}

} catch (error) {

console.error('❌ Erreur soumission:', error);

// Gestion avancée des erreurs avec debug

let errorTitle = 'Erreur de Soumission';

let errorMessage = 'Une erreur est survenue lors de la soumission du formulaire.';

let serverResponse = null;

try {

// Analyser la réponse si c'est une erreur fetch

if (error.response) {

serverResponse = error.response;

errorMessage = error.message || 'Erreur de communication avec le serveur.';

} else if (error.message) {

errorMessage = error.message;

serverResponse = {

error: error.name,

message: error.message,

stack: error.stack,

timestamp: new Date().toISOString()

};

}

// Ajouter l'analyse du formulaire au debug

const analysisResult = analyzeFormDataForDebug();

if (serverResponse) {

serverResponse.formAnalysis = analysisResult;

} else {

serverResponse = { formAnalysis: analysisResult };

}

} catch (debugError) {

console.warn('Erreur lors de l\'analyse debug:', debugError);

serverResponse = {

originalError: error.toString(),

debugError: debugError.toString(),

timestamp: new Date().toISOString()

};

}

// Afficher le modal d'erreur avec tous les détails

showErrorModal(errorTitle, errorMessage, serverResponse, true);

} finally {

showGlobalLoader(false);

updateSaveIndicator('success');

}

}

/\*\*

\* Afficher/masquer le loader global

\*/

function showGlobalLoader(show) {

const loader = document.getElementById('global-loader');

if (loader) {

if (show) {

loader.classList.remove('d-none');

} else {

loader.classList.add('d-none');

}

}

}

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

// 12. UTILITAIRES AVANCÉS

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

/\*\*

\* Géolocalisation

\*/

function getCurrentLocation() {

const btn = document.getElementById('getLocationBtn');

if (!navigator.geolocation) {

showNotification('Géolocalisation non supportée par votre navigateur', 'warning');

return;

}

btn.innerHTML = '<i class="fas fa-spinner fa-spin me-2"></i>Localisation en cours...';

btn.disabled = true;

navigator.geolocation.getCurrentPosition(

(position) => {

const lat = position.coords.latitude.toFixed(7);

const lng = position.coords.longitude.toFixed(7);

// Vérifier si c'est au Gabon (limites approximatives)

if (lat >= -3.978 && lat <= 2.318 && lng >= 8.695 && lng <= 14.502) {

document.getElementById('org\_latitude').value = lat;

document.getElementById('org\_longitude').value = lng;

showNotification('Position obtenue avec succès', 'success');

} else {

showNotification('Position détectée hors du Gabon', 'warning');

}

btn.innerHTML = '<i class="fas fa-map-marker-alt me-2"></i>Obtenir ma position actuelle';

btn.disabled = false;

},

(error) => {

console.error('Erreur géolocalisation:', error);

showNotification('Impossible d\'obtenir votre position', 'danger');

btn.innerHTML = '<i class="fas fa-map-marker-alt me-2"></i>Obtenir ma position actuelle';

btn.disabled = false;

},

{

enableHighAccuracy: true,

timeout: 10000,

maximumAge: 600000 // 10 minutes

}

);

}

/\*\*

\* Gestion des départements selon la province

\*/

function updateDepartements() {

const province = document.getElementById('org\_province')?.value;

const departementSelect = document.getElementById('org\_departement');

if (!departementSelect || !province) return;

const departements = {

'Estuaire': ['Libreville', 'Komo-Mondah', 'Noya'],

'Haut-Ogooué': ['Franceville', 'Lékoko', 'Lemboumbi-Leyou', 'Mpassa', 'Plateaux'],

'Moyen-Ogooué': ['Lambaréné', 'Abanga-Bigné', 'Ogooué et des Lacs'],

'Ngounié': ['Mouila', 'Dola', 'Douya-Onoy', 'Lolo-Bouenguidi', 'Tsamba-Magotsi'],

'Nyanga': ['Tchibanga', 'Basse-Banio', 'Douigni', 'Haute-Banio', 'Mougoutsi', 'Ndolou'],

'Ogooué-Ivindo': ['Makokou', 'Ivindo', 'Lope', 'Mvoung', 'Zadie'],

'Ogooué-Lolo': ['Koulamoutou', 'Lolo', 'Lombo-Bouenguidi', 'Mulundu', 'Offoue-Onoye'],

'Ogooué-Maritime': ['Port-Gentil', 'Bendje', 'Etimboue', 'Komo-Kango'],

'Woleu-Ntem': ['Oyem', 'Haut-Como', 'Haut-Ntem', 'Ntem', 'Okano', 'Woleu']

};

const depts = departements[province] || [];

departementSelect.innerHTML = '<option value="">Sélectionnez un département</option>';

depts.forEach(dept => {

const option = document.createElement('option');

option.value = dept;

option.textContent = dept;

departementSelect.appendChild(option);

});

}

/\*\*

\* Télécharger le modèle Excel pour les adhérents

\*/

function downloadTemplate() {

// Créer un fichier CSV simple comme modèle

const csvContent = `Civilité,Nom,Prenom,NIP,Telephone,Profession

M,DUPONT,Jean,1234567890123,01234567,Ingénieur

Mme,MARTIN,Marie,1234567890124,01234568,Professeure

M,BERNARD,Paul,1234567890125,01234569,Commerçant`;

const blob = new Blob([csvContent], { type: 'text/csv;charset=utf-8;' });

const link = document.createElement('a');

if (link.download !== undefined) {

const url = URL.createObjectURL(blob);

link.setAttribute('href', url);

link.setAttribute('download', 'modele\_adherents\_pngdi.csv');

link.style.visibility = 'hidden';

document.body.appendChild(link);

link.click();

document.body.removeChild(link);

showNotification('Modèle téléchargé avec succès', 'success');

}

}

/\*\*

\* Gestion mode adhérents (manuel vs fichier)

\*/

function toggleAdherentMode(mode) {

const manuelSection = document.getElementById('adherent\_manuel\_section');

const fichierSection = document.getElementById('adherent\_fichier\_section');

if (mode === 'manuel') {

if (manuelSection) manuelSection.classList.remove('d-none');

if (fichierSection) fichierSection.classList.add('d-none');

} else {

if (manuelSection) manuelSection.classList.add('d-none');

if (fichierSection) fichierSection.classList.remove('d-none');

}

}

/\*\*

\* Démarrer la sauvegarde automatique

\*/

function startAutoSave() {

if (OrganisationApp.timers.autoSave) {

clearInterval(OrganisationApp.timers.autoSave);

}

OrganisationApp.timers.autoSave = setInterval(() => {

autoSave();

}, OrganisationApp.config.autoSaveInterval);

console.log('🔄 Auto-sauvegarde démarrée (30s)');

}

/\*\*

\* Arrêter la sauvegarde automatique

\*/

function stopAutoSave() {

if (OrganisationApp.timers.autoSave) {

clearInterval(OrganisationApp.timers.autoSave);

OrganisationApp.timers.autoSave = null;

}

}

// AJOUTER CETTE FONCTION de diagnostic

function analyzeFormData() {

const form = document.getElementById('organisationForm');

const formData = new FormData(form);

let totalSize = 0;

let fieldCount = 0;

let largestFields = [];

const fieldsByType = {};

for (let [key, value] of formData.entries()) {

fieldCount++;

const size = new Blob([value]).size;

totalSize += size;

// Catégoriser par type

if (value instanceof File) {

if (!fieldsByType.files) fieldsByType.files = [];

fieldsByType.files.push({key, size, name: value.name});

} else if (typeof value === 'string' && value.length > 100) {

if (!fieldsByType.longText) fieldsByType.longText = [];

fieldsByType.longText.push({key, size, preview: value.substring(0, 50) + '...'});

}

if (size > 1000) { // Champs > 1KB

largestFields.push({

key,

size,

type: value instanceof File ? 'file' : 'text',

preview: value instanceof File ? value.name : value.toString().substring(0, 50) + '...'

});

}

}

// Analyser les données spécifiques

const organisationData = {

fondateurs: OrganisationApp.fondateurs.length,

adherents: OrganisationApp.adherents.length,

documents: Object.keys(OrganisationApp.documents).length,

anomalies: OrganisationApp.rapportAnomalies.enabled ? OrganisationApp.rapportAnomalies.adherentsAvecAnomalies : 0

};

const analysis = {

fieldCount,

totalSize,

largestFields,

fieldsByType,

organisationData,

warnings: []

};

// Générer des avertissements

if (fieldCount > 1000) {

analysis.warnings.push(`Nombre de champs élevé: ${fieldCount} (limite recommandée: 1000)`);

}

if (totalSize > 50 \* 1024 \* 1024) { // 50MB

analysis.warnings.push(`Taille importante: ${(totalSize / 1024 / 1024).toFixed(2)} MB`);

}

if (largestFields.length > 10) {

analysis.warnings.push(`Nombreux champs volumineux: ${largestFields.length}`);

}

console.log('=== ANALYSE FORMULAIRE COMPLÈTE ===');

console.log('Nombre de champs:', fieldCount);

console.log('Taille totale:', (totalSize / 1024).toFixed(2) + ' KB');

console.log('Organisation:', organisationData);

console.log('Champs volumineux:', largestFields);

console.log('Avertissements:', analysis.warnings);

return analysis;

}

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

// 13. INITIALISATION COMPLÈTE

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

/\*\*

\* Initialisation complète de l'application

\*/

function initializeApplication() {

console.log('🚀 Initialisation complète PNGDI - Création Organisation');

// Initialiser l'affichage

updateStepDisplay();

updateNavigationButtons();

// Configurer les événements

setupEventListeners();

// Charger les données sauvegardées

loadSavedData();

// Démarrer l'auto-sauvegarde

startAutoSave();

console.log('✅ Application initialisée avec succès');

}

/\*\*

\* Configuration des événements

\*/

function setupEventListeners() {

// Événements pour les cartes d'organisation

document.querySelectorAll('.organization-type-card').forEach(card => {

card.addEventListener('click', function() {

selectOrganizationType(this);

});

// Accessibilité clavier

card.addEventListener('keydown', function(e) {

if (e.key === 'Enter' || e.key === ' ') {

e.preventDefault();

selectOrganizationType(this);

}

});

card.setAttribute('tabindex', '0');

card.setAttribute('role', 'button');

});

// Bouton géolocalisation

const geoBtn = document.getElementById('getLocationBtn');

if (geoBtn) {

geoBtn.addEventListener('click', getCurrentLocation);

}

// Boutons fondateurs et adhérents

const addFondateurBtn = document.getElementById('addFondateurBtn');

if (addFondateurBtn) {

addFondateurBtn.addEventListener('click', addFondateur);

}

const addAdherentBtn = document.getElementById('addAdherentBtn');

if (addAdherentBtn) {

addAdherentBtn.addEventListener('click', addAdherent);

}

// Bouton téléchargement modèle

const downloadBtn = document.getElementById('downloadTemplateBtn');

if (downloadBtn) {

downloadBtn.addEventListener('click', downloadTemplate);

}

// Mode adhérents

document.querySelectorAll('input[name="adherent\_mode"]').forEach(radio => {

radio.addEventListener('change', function() {

toggleAdherentMode(this.value);

});

});

// Province/département

const provinceSelect = document.getElementById('org\_province');

if (provinceSelect) {

provinceSelect.addEventListener('change', updateDepartements);

}

// Validation en temps réel avec débounce pour org\_objet

const orgObjetField = document.getElementById('org\_objet');

if (orgObjetField) {

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

const currentLength = e.target.value.trim().length;

const minLength = 50;

// Mettre à jour compteur en temps réel

let counterDiv = e.target.parentNode.querySelector('.char-counter');

if (!counterDiv) {

counterDiv = document.createElement('div');

counterDiv.className = 'char-counter small text-muted mt-1';

e.target.parentNode.appendChild(counterDiv);

}

counterDiv.textContent = `${currentLength}/${minLength} caractères`;

counterDiv.style.color = currentLength < minLength ? '#dc3545' : '#28a745';

// Validation différée

clearTimeout(e.target.validationTimeout);

e.target.validationTimeout = setTimeout(() => {

validateField(e.target);

}, OrganisationApp.config.validationDelay);

});

}

// Validation en temps réel pour autres champs

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

const validationSelector = 'input:not(#org\_objet), textarea:not(#org\_objet), select';

if (elementMatches(e.target, validationSelector)) {

clearTimeout(e.target.validationTimeout);

e.target.validationTimeout = setTimeout(() => {

validateField(e.target);

}, OrganisationApp.config.validationDelay);

}

});

// Configuration de l'importation fichier adhérents

initializeAdherentFileImport();

// Sauvegarde avant fermeture

window.addEventListener('beforeunload', function(e) {

autoSave();

});

// Raccourcis clavier

document.addEventListener('keydown', function(e) {

// Ctrl+S pour sauvegarder

if (e.ctrlKey && e.key === 's') {

e.preventDefault();

autoSave();

showNotification('Données sauvegardées', 'success');

}

// Flèches pour navigation (si pas dans un champ)

if (!elementMatches(e.target,'input, textarea, select')) {

if (e.key === 'ArrowRight' && e.ctrlKey) {

e.preventDefault();

changeStep(1);

} else if (e.key === 'ArrowLeft' && e.ctrlKey) {

e.preventDefault();

changeStep(-1);

}

}

});

// Bouton de soumission finale

const submitBtn = document.getElementById('submitBtn');

if (submitBtn) {

submitBtn.addEventListener('click', function(e) {

e.preventDefault();

submitForm();

});

}

}

/\*\*

\* Initialiser l'importation de fichier adhérents

\*/

function initializeAdherentFileImport() {

const fileInput = document.getElementById('adherents\_file');

if (fileInput) {

// Supprimer les anciens event listeners

fileInput.removeEventListener('change', handleAdherentFileImport);

// Ajouter le nouvel event listener

fileInput.addEventListener('change', function() {

if (this.files.length > 0) {

handleAdherentFileImport(this);

}

});

console.log('✅ Événement importation fichier adhérents configuré');

}

}

/\*\*

\* Basculer les déclarations selon le type d'organisation

\*/

function toggleDeclarationParti() {

const typeOrganisation = document.querySelector('input[name="type\_organisation"]:checked');

const declarationParti = document.getElementById('declaration\_parti\_politique');

if (typeOrganisation && typeOrganisation.value === 'parti\_politique') {

if (declarationParti) {

declarationParti.classList.remove('d-none');

const checkbox = document.getElementById('declaration\_exclusivite\_parti');

if (checkbox) {

checkbox.required = true;

}

}

} else {

if (declarationParti) {

declarationParti.classList.add('d-none');

const checkbox = document.getElementById('declaration\_exclusivite\_parti');

if (checkbox) {

checkbox.required = false;

checkbox.checked = false;

}

}

}

}

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

// 14. FONCTIONS GLOBALES EXPOSÉES

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

// Exposer les fonctions principales pour compatibilité avec le HTML

window.changeStep = changeStep;

window.selectOrganizationType = selectOrganizationType;

window.addFondateur = addFondateur;

window.removeFondateur = removeFondateur;

window.addAdherent = addAdherent;

window.removeAdherent = removeAdherent;

window.handleDocumentUpload = handleDocumentUpload;

window.getCurrentLocation = getCurrentLocation;

window.openImageModal = openImageModal;

window.handleAdherentFileImport = handleAdherentFileImport;

window.toggleImportDetails = toggleImportDetails;

window.previewRapportAnomalies = previewRapportAnomalies;

window.downloadRapportAnomalies = downloadRapportAnomalies;

window.exportRapportHTML = exportRapportHTML;

window.filterAnomalies = filterAnomalies;

window.downloadTemplate = downloadTemplate;

window.toggleAdherentMode = toggleAdherentMode;

window.updateDepartements = updateDepartements;

window.toggleDeclarationParti = toggleDeclarationParti;

window.submitForm = submitForm;

/\*\*

\* Vérification de l'intégrité du système d'anomalies

\*/

function verifyAnomaliesSystem() {

const checks = {

configurationAnomalies: !!OrganisationApp.config.anomalies,

rapportAnomaliesStructure: !!OrganisationApp.rapportAnomalies,

fonctionsUtilitaires: typeof createAnomalie === 'function' && typeof addAnomalieToReport === 'function',

fonctionsGeneration: typeof generateRapportAnomalies === 'function',

fonctionsInterface: typeof previewRapportAnomalies === 'function',

integrationRecapitulatif: typeof generateAnomaliesRecapSection === 'function'

};

const allChecksPass = Object.values(checks).every(check => check === true);

console.log('🔍 Vérification intégrité système anomalies:', checks);

console.log(allChecksPass ? '✅ Système d\'anomalies opérationnel' : '❌ Problème détecté dans le système');

return allChecksPass;

}

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

// 15. INITIALISATION AU CHARGEMENT DOM

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

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

// Vérifier que nous sommes sur la bonne page

if (document.getElementById('organisationForm')) {

initializeApplication();

// Vérifier l'intégrité du système d'anomalies

verifyAnomaliesSystem();

// Configurer les événements spéciaux

setupSpecialEventListeners();

}

// Ajouter les styles pour les animations de notifications

if (!document.getElementById('notification-styles')) {

const styles = document.createElement('style');

styles.id = 'notification-styles';

styles.textContent = `

@keyframes slideInRight {

from { transform: translateX(100%); opacity: 0; }

to { transform: translateX(0); opacity: 1; }

}

@keyframes slideOutRight {

from { transform: translateX(0); opacity: 1; }

to { transform: translateX(100%); opacity: 0; }

}

@keyframes shake {

0%, 100% { transform: translateX(0); }

10%, 30%, 50%, 70%, 90% { transform: translateX(-5px); }

20%, 40%, 60%, 80% { transform: translateX(5px); }

}

.table-warning {

background-color: rgba(255, 243, 205, 0.3) !important;

}

.badge.bg-warning {

font-size: 0.75em;

}

.char-counter {

transition: color 0.3s ease;

}

.organization-type-card {

transition: all 0.3s ease;

cursor: pointer;

}

.organization-type-card:hover {

transform: translateY(-2px);

box-shadow: 0 4px 8px rgba(0,0,0,0.1);

}

.organization-type-card.active {

border-color: #0d6efd !important;

background-color: rgba(13, 110, 253, 0.05);

}

.step-indicator {

transition: all 0.3s ease;

cursor: pointer;

}

.step-indicator:hover {

transform: scale(1.1);

}

.step-indicator.active {

background-color: #0d6efd !important;

color: white !important;

}

.step-indicator.completed {

background-color: #198754 !important;

color: white !important;

}

.anomalie-row {

transition: all 0.3s ease;

}

.anomalie-critique {

border-left: 3px solid #dc3545;

}

.anomalie-majeure {

border-left: 3px solid #ffc107;

}

.anomalie-mineure {

border-left: 3px solid #17a2b8;

}

`;

document.head.appendChild(styles);

}

});

/\*\*

\* Configuration des événements spéciaux

\*/

function setupSpecialEventListeners() {

// Événement pour basculer les déclarations parti politique

document.addEventListener('change', function(e) {

if (elementMatches(e.target,'input[name="type\_organisation"]')) {

toggleDeclarationParti();

}

});

// Gestion des navigation avec les touches

document.addEventListener('keydown', function(e) {

// Échapper pour fermer les modals

if (e.key === 'Escape') {

const modals = document.querySelectorAll('.modal.show');

modals.forEach(modal => {

const modalInstance = bootstrap.Modal.getInstance(modal);

if (modalInstance) {

modalInstance.hide();

}

});

}

// Entrée pour valider les étapes

if (e.key === 'Enter' && e.ctrlKey) {

e.preventDefault();

if (OrganisationApp.currentStep < OrganisationApp.totalSteps) {

changeStep(1);

} else {

submitForm();

}

}

});

// Auto-focus sur le premier champ de chaque étape

const observer = new MutationObserver(function(mutations) {

mutations.forEach(function(mutation) {

if (mutation.type === 'attributes' && mutation.attributeName === 'style') {

const target = mutation.target;

if (target.classList.contains('step-content') && target.style.display === 'block') {

setTimeout(() => {

const firstInput = target.querySelector('input:not([type="hidden"]):not([type="radio"]):not([type="checkbox"]), select, textarea');

if (firstInput && !firstInput.disabled) {

firstInput.focus();

}

}, 100);

}

}

});

});

// Observer les changements d'affichage des étapes

document.querySelectorAll('.step-content').forEach(step => {

observer.observe(step, { attributes: true, attributeFilter: ['style'] });

});

// Gestion des tooltips Bootstrap si disponible

if (typeof bootstrap !== 'undefined' && bootstrap.Tooltip) {

const tooltipTriggerList = [].slice.call(document.querySelectorAll('[data-bs-toggle="tooltip"]'));

tooltipTriggerList.map(function (tooltipTriggerEl) {

return new bootstrap.Tooltip(tooltipTriggerEl);

});

}

// Validation automatique à la perte de focus

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

if (elementMatches(e.target,'input[required], select[required], textarea[required]')) {

validateField(e.target);

}

}, true);

// Nettoyage automatique des erreurs lors de la saisie

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

if (e.target.classList.contains('is-invalid')) {

// Nettoyer l'erreur après 1 seconde de saisie continue

clearTimeout(e.target.cleanupTimeout);

e.target.cleanupTimeout = setTimeout(() => {

if (e.target.value.trim()) {

clearFieldError(e.target);

}

}, 1000);

}

});

// Prévenír la soumission accidentelle avec Entrée

document.addEventListener('keydown', function(e) {

if (e.key === 'Enter' && elementMatches(e.target,'input:not([type="submit"]):not([type="button"])')) {

e.preventDefault();

// Aller au champ suivant

const form = e.target.closest('form');

if (form) {

const formElements = Array.from(form.querySelectorAll('input, select, textarea, button'));

const currentIndex = formElements.indexOf(e.target);

const nextElement = formElements[currentIndex + 1];

if (nextElement && !nextElement.disabled) {

nextElement.focus();

}

}

}

});

console.log('✅ Événements spéciaux configurés');

}

/\*\*

\* Message de fin de chargement

\*/

console.log(`

🎉 ========================================================================

PNGDI - Formulaire Création Organisation - CHARGEMENT TERMINÉ

========================================================================

✅ Version: 1.2 - Système d'anomalies intégré

✅ 9 étapes complètes avec validation

✅ Import Excel/CSV avec détection d'anomalies

✅ Rapport d'anomalies automatique

✅ Sauvegarde automatique toutes les 30s

✅ Validation temps réel

✅ Interface responsive Bootstrap 5

✅ 22 professions exclues pour partis politiques

✅ Gestion complète des documents

✅ Géolocalisation intégrée

✅ Raccourcis clavier

🚀 Prêt pour production !

📋 Système révolutionnaire de conservation totale des anomalies

🇬🇦 Conformité législation gabonaise

Développé pour l'excellence du service public gabonais

========================================================================

`);

// Vérification finale de l'intégrité au chargement

setTimeout(() => {

const integrityCheck = verifyAnomaliesSystem();

if (integrityCheck) {

console.log('🎯 Système opérationnel - Toutes les fonctionnalités disponibles');

} else {

console.warn('⚠️ Problème d\'intégrité détecté - Certaines fonctionnalités peuvent être limitées');

}

}, 1000);

// Fin du fichier JavaScript complet