/\*\*

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

\* PNGDI - Formulaire Création Organisation - VERSION FINALE PROPRE

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

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

\* MISE À JOUR: Gestion complète anomalies importation adhérents

\* Date: 28/06/2025 - Version 1.1

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

\*/

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

// 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 gabonais

nip: {

length: 13,

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

strictValidation: false,

allowTestValues: true

},

// Configuration téléphone gabonais

phone: {

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

minLength: 8,

maxLength: 9,

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

},

// 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']

}

}

},

// Données de l'application

cache: new Map(),

formData: {},

validationErrors: {},

fondateurs: [],

adherents: [],

documents: {},

rapportAnomaliesAdherents: null, // NOUVEAU: rapport d'anomalies

// Timers

timers: {

autoSave: null,

validation: {}

}

};

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

// 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();

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 les éléments de progression

updateProgressBar();

updateStepIndicators();

// Mettre à jour le numéro d'étape

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

if (currentStepNumber) {

currentStepNumber.textContent = OrganisationApp.currentStep;

}

}

/\*\*

\* 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>

`,

'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. SYSTÈME DE VALIDATION COMPLET

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

/\*\*

\* 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;

}

});

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'];

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);

default:

return validateGenericField(field, value);

}

}

/\*\*

\* Validation NIP gabonais

\*/

function validateNIP(field, value) {

if (!value) {

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

return false;

}

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;

}

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

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 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) {

updateFoundersRequirements();

updateMembersRequirements();

}

}

/\*\*

\* Mise à jour des exigences fondateurs

\*/

function updateFoundersRequirements() {

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 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

\*/

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;

}

// 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();

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 simple

const itemsPerPage = 10;

const currentPage = 1;

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>Actions</th>

</tr>

</thead>

<tbody>

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

const globalIndex = startIndex + index;

const hasAnomalies = adherent.anomalies && adherent.anomalies.length > 0;

const rowClass = hasAnomalies ? 'table-warning' : '';

const title = hasAnomalies ? `${adherent.anomalies.length} anomalie(s) détectée(s)` : '';

return `

<tr class="${rowClass}" title="${title}">

<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>

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

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

</button>

${hasAnomalies ? `<span class="badge bg-warning ms-1">${adherent.anomalies.length} anomalie(s)</span>` : ''}

</td>

</tr>

`;

}).join('')}

</tbody>

</table>

</div>

`;

}

if (countSpan) {

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 = '';

}

}

});

}

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

// 6. IMPORTATION ADHÉRENTS AVEC GESTION ANOMALIES

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

/\*\*

\* Gestion de l'importation du fichier Excel/CSV avec anomalies

\*/

async function handleAdherentFileImport(fileInput) {

const file = fileInput.files[0];

if (!file) return;

console.log('📁 Début importation avec gestion anomalies:', file.name);

if (!validateAdherentFile(file)) {

clearFileInput();

return;

}

try {

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

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`);

// Validation avec gestion des anomalies

const validationResult = await validateAdherentsImportWithAnomalies(adherentsData);

// Traiter selon les résultats

await processImportResultWithAnomalies(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',

'application/vnd.ms-excel',

'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')) {

const csvText = e.target.result;

data = parseAdherentCSV(csvText);

} else {

reject(new Error('Import Excel non encore implémenté. Utilisez un fichier CSV.'));

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 CSV des adhérents

\*/

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');

}

// Amélioration du parsing pour gérer les guillemets et virgules

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 +