<!-- ========== DÉBUT SECTION O - CONFIGURATION BACKEND ========== -->

<script>

/\* ================================================

SECTION O : INTÉGRATION BACKEND ET OPTIMISATIONS

================================================ \*/

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

// CONFIGURATION LARAVEL ET AXIOS

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

// Configuration CSRF et Headers

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

// Token CSRF depuis meta tag

const csrfToken = document.querySelector('meta[name="csrf-token"]');

if (csrfToken) {

window.axios.defaults.headers.common['X-CSRF-TOKEN'] = csrfToken.getAttribute('content');

}

// Headers par défaut

window.axios.defaults.headers.common['X-Requested-With'] = 'XMLHttpRequest';

window.axios.defaults.headers.common['Accept'] = 'application/json';

// Timeout global

window.axios.defaults.timeout = 30000; // 30 secondes

// Intercepteur de réponse pour gestion d'erreurs

window.axios.interceptors.response.use(

response => response,

error => {

handleAxiosError(error);

return Promise.reject(error);

}

);

});

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

// GESTION DES ERREURS SERVEUR

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

function handleAxiosError(error) {

console.error('Erreur API:', error);

if (error.response) {

const status = error.response.status;

const data = error.response.data;

switch(status) {

case 400:

showNotification('Données invalides: ' + (data.message || 'Erreur de validation'), 'danger');

break;

case 401:

showNotification('Session expirée. Veuillez vous reconnecter.', 'warning');

setTimeout(() => {

window.location.href = '/login';

}, 2000);

break;

case 403:

showNotification('Accès refusé. Permissions insuffisantes.', 'danger');

break;

case 404:

showNotification('Ressource non trouvée', 'danger');

break;

case 413:

showNotification('Fichier trop volumineux (max 5MB par fichier)', 'danger');

break;

case 422:

if (data.errors) {

displayValidationErrors(data.errors);

} else {

showNotification('Erreur de validation: ' + (data.message || 'Données invalides'), 'danger');

}

break;

case 429:

showNotification('Trop de tentatives. Veuillez patienter quelques minutes.', 'warning');

break;

case 500:

showNotification('Erreur serveur interne. Veuillez réessayer plus tard.', 'danger');

break;

case 503:

showNotification('Service temporairement indisponible. Maintenance en cours.', 'warning');

break;

default:

showNotification('Erreur de connexion (' + status + ')', 'danger');

}

} else if (error.request) {

showNotification('Impossible de contacter le serveur. Vérifiez votre connexion internet.', 'danger');

} else {

showNotification('Erreur inattendue: ' + error.message, 'danger');

}

}

// Afficher les erreurs de validation Laravel

function displayValidationErrors(errors) {

Object.keys(errors).forEach(field => {

const fieldElement = document.querySelector(`[name="${field}"]`);

if (fieldElement) {

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

// Ajouter le message d'erreur

let feedbackElement = fieldElement.parentElement.querySelector('.invalid-feedback');

if (!feedbackElement) {

feedbackElement = document.createElement('div');

feedbackElement.className = 'invalid-feedback';

fieldElement.parentElement.appendChild(feedbackElement);

}

feedbackElement.textContent = errors[field][0];

}

});

// Notification générale

showNotification('Veuillez corriger les erreurs dans le formulaire', 'danger');

}

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

// VÉRIFICATIONS TEMPS RÉEL AVEC BACKEND

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

// Vérification NIP en temps réel

async function verifyNIPWithServer(nip, callback) {

if (!nip || nip.length !== 13) {

callback(false, 'NIP invalide (13 chiffres requis)');

return;

}

try {

showNotification('Vérification du NIP...', 'info', 1000);

const response = await window.axios.post('/api/verify-nip', {

nip: nip,

organization\_type: selectedOrgType

});

const data = response.data;

callback(data.valid, data.message, data.details);

} catch (error) {

console.error('Erreur vérification NIP:', error);

callback(false, 'Erreur de vérification. Réessayez plus tard.');

}

}

// Vérification unicité nom organisation

async function verifyOrganizationName(name, callback) {

if (!name || name.length < 3) {

callback(false, 'Nom trop court');

return;

}

try {

const response = await window.axios.post('/api/verify-organization-name', {

name: name,

type: selectedOrgType

});

const data = response.data;

callback(data.available, data.message);

} catch (error) {

console.error('Erreur vérification nom:', error);

callback(false, 'Erreur de vérification');

}

}

// Vérification doublons adhérents

async function verifyMembersWithServer(members, callback) {

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

callback({ valid: [], conflicts: [], errors: [] });

return;

}

try {

showNotification('Vérification des adhérents...', 'info', 2000);

const response = await window.axios.post('/api/verify-members', {

members: members,

organization\_type: selectedOrgType

});

callback(response.data);

} catch (error) {

console.error('Erreur vérification adhérents:', error);

callback({ valid: [], conflicts: members, errors: ['Erreur de vérification'] });

}

}

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

// UPLOAD OPTIMISÉ DES FICHIERS

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

// Upload avec progress bar

async function uploadDocumentWithProgress(file, docType, progressCallback) {

const formData = new FormData();

formData.append('document', file);

formData.append('type', docType);

formData.append('organization\_type', selectedOrgType);

try {

const response = await window.axios.post('/api/upload-document', formData, {

headers: {

'Content-Type': 'multipart/form-data'

},

onUploadProgress: (progressEvent) => {

const progress = Math.round((progressEvent.loaded \* 100) / progressEvent.total);

if (progressCallback) {

progressCallback(progress);

}

}

});

return response.data;

} catch (error) {

console.error('Erreur upload:', error);

throw error;

}

}

// Upload multiple avec gestion d'erreurs

async function uploadMultipleDocuments(documents, progressCallback) {

const results = [];

const total = Object.keys(documents).length;

let completed = 0;

for (const [docType, file] of Object.entries(documents)) {

try {

const result = await uploadDocumentWithProgress(file, docType, (progress) => {

const globalProgress = Math.round(((completed + (progress / 100)) / total) \* 100);

if (progressCallback) {

progressCallback(globalProgress, docType, progress);

}

});

results.push({ docType, success: true, data: result });

completed++;

} catch (error) {

results.push({ docType, success: false, error: error.message });

completed++;

}

}

return results;

}

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

// SAUVEGARDE AUTOMATIQUE SERVEUR

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

// Sauvegarde brouillon sur serveur

async function saveDraftToServer() {

const formData = collectAllFormData();

try {

showNotification('Sauvegarde en cours...', 'info');

const response = await window.axios.post('/api/save-draft', {

data: formData,

step: currentStep

});

// Stocker l'ID du brouillon

if (response.data.draft\_id) {

localStorage.setItem('draftId', response.data.draft\_id);

}

showNotification('Brouillon sauvegardé', 'success', 2000);

return response.data;

} catch (error) {

console.error('Erreur sauvegarde:', error);

showNotification('Erreur de sauvegarde. Utilisation du cache local.', 'warning');

// Fallback vers localStorage

localStorage.setItem('organizationFormData', JSON.stringify(formData));

}

}

// Charger brouillon depuis serveur

async function loadDraftFromServer() {

const draftId = localStorage.getItem('draftId');

if (!draftId) return null;

try {

const response = await window.axios.get(`/api/load-draft/${draftId}`);

return response.data;

} catch (error) {

console.error('Erreur chargement brouillon:', error);

// Fallback vers localStorage

const localData = localStorage.getItem('organizationFormData');

return localData ? JSON.parse(localData) : null;

}

}

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

// SOUMISSION FINALE OPTIMISÉE

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

// Soumission avec retry automatique

async function submitOrganizationWithRetry(maxRetries = 3) {

let lastError;

for (let attempt = 1; attempt <= maxRetries; attempt++) {

try {

showNotification(`Soumission en cours... (tentative ${attempt}/${maxRetries})`, 'info');

const result = await submitOrganizationToServer();

return result;

} catch (error) {

lastError = error;

console.error(`Tentative ${attempt} échouée:`, error);

if (attempt < maxRetries) {

const delay = Math.pow(2, attempt) \* 1000; // Backoff exponentiel

showNotification(`Tentative ${attempt} échouée. Nouvelle tentative dans ${delay/1000}s...`, 'warning');

await new Promise(resolve => setTimeout(resolve, delay));

}

}

}

// Toutes les tentatives ont échoué

throw lastError;

}

// Soumission principale

async function submitOrganizationToServer() {

// Préparer les données

const organizationData = {

// Informations générales

type: selectedOrgType,

demandeur: collectDemandeurData(),

organization: collectOrganizationData(),

coordinates: collectCoordinatesData(),

// Membres

fondateurs: collectFondateursData(),

adherents: adherentsData,

// Métadonnées

creation\_steps: collectStepsMetadata(),

submission\_timestamp: new Date().toISOString()

};

// Étape 1: Créer l'organisation

const orgResponse = await window.axios.post('/organizations', organizationData);

const organizationId = orgResponse.data.id;

// Étape 2: Uploader les documents

if (Object.keys(documentsUploaded).length > 0) {

showNotification('Upload des documents...', 'info');

const uploadResults = await uploadMultipleDocuments(documentsUploaded, (progress, docType) => {

console.log(`Upload ${docType}: ${progress}%`);

});

// Associer les documents à l'organisation

const documentIds = uploadResults

.filter(result => result.success)

.map(result => result.data.id);

if (documentIds.length > 0) {

await window.axios.post(`/organizations/${organizationId}/documents`, {

document\_ids: documentIds

});

}

}

// Étape 3: Finaliser la soumission

const finalResponse = await window.axios.post(`/organizations/${organizationId}/submit`, {

final\_submission: true

});

return finalResponse.data;

}

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

// OPTIMISATIONS PERFORMANCE

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

// Debounce pour validations

function debounceValidation(func, delay = 500) {

let timeoutId;

return function (...args) {

clearTimeout(timeoutId);

timeoutId = setTimeout(() => func.apply(this, args), delay);

};

}

// Cache des réponses API

const apiCache = new Map();

function getCachedOrFetch(key, fetchFunction, ttl = 300000) { // 5 minutes

const cached = apiCache.get(key);

const now = Date.now();

if (cached && (now - cached.timestamp) < ttl) {

return Promise.resolve(cached.data);

}

return fetchFunction().then(data => {

apiCache.set(key, { data, timestamp: now });

return data;

});

}

// Compression des données avant envoi

function compressFormData(data) {

// Supprimer les champs vides

const cleaned = {};

Object.keys(data).forEach(key => {

if (data[key] !== null && data[key] !== undefined && data[key] !== '') {

cleaned[key] = data[key];

}

});

return cleaned;

}

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

// COLLECTE DE DONNÉES FINALES

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

function collectDemandeurData() {

return {

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

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

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

prenoms: document.getElementById('demandeur\_prenoms')?.value,

date\_naissance: document.getElementById('demandeur\_date\_naissance')?.value,

lieu\_naissance: document.getElementById('demandeur\_lieu\_naissance')?.value,

sexe: document.getElementById('demandeur\_sexe')?.value,

nationalite: document.getElementById('demandeur\_nationalite')?.value,

profession: document.getElementById('demandeur\_profession')?.value,

email: document.getElementById('demandeur\_email')?.value,

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

telephone\_2: document.getElementById('demandeur\_telephone\_2')?.value,

adresse: document.getElementById('demandeur\_adresse')?.value,

cni: {

numero: document.getElementById('demandeur\_cni\_numero')?.value,

date\_etablissement: document.getElementById('demandeur\_cni\_date\_etablissement')?.value,

date\_expiration: document.getElementById('demandeur\_cni\_date\_expiration')?.value,

lieu\_etablissement: document.getElementById('demandeur\_cni\_lieu\_etablissement')?.value

}

};

}

function collectOrganizationData() {

return {

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

sigle: document.getElementById('org\_sigle')?.value,

slogan: document.getElementById('org\_slogan')?.value,

objet: document.getElementById('org\_objet')?.value,

objectifs: document.getElementById('org\_objectifs')?.value,

secteur: document.getElementById('org\_secteur')?.value,

zone\_intervention: document.getElementById('org\_zone\_intervention')?.value,

public\_cible: document.getElementById('org\_public\_cible')?.value,

langues: Array.from(document.getElementById('org\_langues')?.selectedOptions || []).map(opt => opt.value),

activites: Array.from(document.querySelectorAll('.activite-checkbox:checked')).map(cb => cb.value)

};

}

function collectCoordinatesData() {

return {

province: document.getElementById('org\_province')?.value,

departement: document.getElementById('org\_departement')?.value,

prefecture: document.getElementById('org\_prefecture')?.value,

sous\_prefecture: document.getElementById('org\_sous\_prefecture')?.value,

canton: document.getElementById('org\_canton')?.value,

regroupement: document.getElementById('org\_regroupement')?.value,

zone\_type: document.getElementById('org\_zone\_type')?.value,

ville\_commune: document.getElementById('org\_ville\_commune')?.value,

arrondissement: document.getElementById('org\_arrondissement')?.value,

quartier: document.getElementById('org\_quartier')?.value,

village: document.getElementById('org\_village')?.value,

lieu\_dit: document.getElementById('org\_lieu\_dit')?.value,

adresse\_complete: document.getElementById('org\_adresse\_complete')?.value,

points\_repere: document.getElementById('org\_points\_repere')?.value,

latitude: document.getElementById('org\_latitude')?.value,

longitude: document.getElementById('org\_longitude')?.value,

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

telephone\_2: document.getElementById('org\_telephone\_2')?.value,

email: document.getElementById('org\_email')?.value,

email\_2: document.getElementById('org\_email\_2')?.value,

site\_web: document.getElementById('org\_site\_web')?.value,

reseaux\_sociaux: document.getElementById('org\_reseaux\_sociaux')?.value

};

}

function collectStepsMetadata() {

return {

total\_steps: totalSteps,

completed\_steps: currentStep,

completion\_time: Date.now() - (window.formStartTime || Date.now()),

type\_selected: selectedOrgType,

validation\_errors: window.formValidationErrors || [],

user\_interactions: window.formInteractions || []

};

}

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

// GESTION DES ÉVÉNEMENTS FINAUX

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

// Initialisation au chargement

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

// Marquer le début du formulaire

window.formStartTime = Date.now();

window.formValidationErrors = [];

window.formInteractions = [];

// Charger brouillon existant

loadDraftFromServer().then(draftData => {

if (draftData) {

showNotification('Brouillon trouvé. Restauration...', 'info');

restoreFromDraft(draftData);

}

});

// Auto-sauvegarde toutes les 2 minutes

setInterval(() => {

if (currentStep > 1) {

saveDraftToServer();

}

}, 120000);

// Sauvegarde avant fermeture

window.addEventListener('beforeunload', function(e) {

if (currentStep > 1 && currentStep < totalSteps) {

saveDraftToServer();

}

});

});

// Fonction de soumission finale

async function confirmSubmission() {

const modal = bootstrap.Modal.getInstance(document.getElementById('confirmSubmitModal'));

modal.hide();

// Désactiver tous les boutons

document.querySelectorAll('button, input[type="submit"]').forEach(btn => {

btn.disabled = true;

});

try {

// Soumission avec retry

const result = await submitOrganizationWithRetry();

// Succès

document.getElementById('dossier-number').textContent = result.dossier\_number;

// Nettoyer les données locales

localStorage.removeItem('organizationFormData');

localStorage.removeItem('draftId');

// Afficher modal de succès

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

successModal.show();

// Redirection après 10 secondes

setTimeout(() => {

window.location.href = `/operator/dossiers/${result.id}`;

}, 10000);

} catch (error) {

console.error('Erreur soumission finale:', error);

// Réactiver les boutons

document.querySelectorAll('button, input[type="submit"]').forEach(btn => {

btn.disabled = false;

});

// Afficher erreur détaillée

let errorMessage = 'Erreur lors de la soumission. ';

if (error.response && error.response.data && error.response.data.message) {

errorMessage += error.response.data.message;

} else {

errorMessage += 'Veuillez réessayer ou contacter le support.';

}

showNotification(errorMessage, 'danger', 10000);

}

}

// Fonction de restauration depuis brouillon

function restoreFromDraft(draftData) {

if (!draftData || !draftData.data) return;

const data = draftData.data;

// Restaurer les données de base

if (data.selectedOrgType) {

selectedOrgType = data.selectedOrgType;

const typeCard = document.querySelector(`[data-type="${selectedOrgType}"]`);

if (typeCard) {

selectOrganizationType(typeCard);

}

}

if (data.currentStep) {

currentStep = data.currentStep;

updateStepDisplay();

}

// Restaurer les fondateurs si présents

if (data.fondateurs && data.fondateurs.length > 0) {

data.fondateurs.forEach((fondateur, index) => {

if (index === 0) {

addFondateur(); // Ajouter le premier

} else {

addFondateur(); // Ajouter les suivants

}

// Remplir les données

const card = document.querySelectorAll('.fondateur-card')[index];

if (card) {

Object.keys(fondateur).forEach(key => {

const field = card.querySelector(`[name\*="${key}"]`);

if (field) {

field.value = fondateur[key];

}

});

}

});

}

// Restaurer les adhérents

if (data.adherents && data.adherents.length > 0) {

adherentsData = data.adherents;

updateAdherentsDisplay();

updateAdherentsCounters();

}

// Restaurer les étapes

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

const stepData = data['step' + i];

if (stepData) {

restoreStepData(i, stepData);

}

}

showNotification(`Brouillon restauré (étape ${currentStep}/${totalSteps})`, 'success');

}

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

// ANALYTIQUES ET LOGGING

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

// Tracker les interactions utilisateur

function trackUserInteraction(action, details = {}) {

if (!window.formInteractions) {

window.formInteractions = [];

}

window.formInteractions.push({

timestamp: Date.now(),

action: action,

step: currentStep,

details: details

});

// Envoyer périodiquement au serveur pour analytics

if (window.formInteractions.length % 10 === 0) {

sendAnalytics();

}

}

// Envoyer les données d'analyse

async function sendAnalytics() {

if (!window.formInteractions || window.formInteractions.length === 0) return;

try {

await window.axios.post('/api/form-analytics', {

interactions: window.formInteractions,

session\_id: getSessionId(),

organization\_type: selectedOrgType

});

// Nettoyer après envoi

window.formInteractions = [];

} catch (error) {

console.error('Erreur envoi analytics:', error);

}

}

function getSessionId() {

let sessionId = sessionStorage.getItem('formSessionId');

if (!sessionId) {

sessionId = 'session\_' + Date.now() + '\_' + Math.random().toString(36).substr(2, 9);

sessionStorage.setItem('formSessionId', sessionId);

}

return sessionId;

}

// Hooks pour tracker les interactions

const originalChangeStep = window.changeStep;

window.changeStep = function(direction) {

trackUserInteraction('step\_change', { direction, from: currentStep, to: currentStep + direction });

return originalChangeStep(direction);

};

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

// FONCTIONS UTILITAIRES FINALES

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

// Vérification de la connectivité

function checkConnectivity() {

return new Promise((resolve) => {

const img = new Image();

img.onload = () => resolve(true);

img.onerror = () => resolve(false);

img.src = '/favicon.ico?' + Date.now();

});

}

// Notification de statut hors ligne

window.addEventListener('online', () => {

showNotification('Connexion rétablie', 'success');

});

window.addEventListener('offline', () => {

showNotification('Connexion perdue. Les données seront sauvegardées localement.', 'warning');

});

// Export pour tests

if (typeof module !== 'undefined' && module.exports) {

module.exports = {

verifyNIPWithServer,

verifyOrganizationName,

uploadDocumentWithProgress,

saveDraftToServer,

submitOrganizationWithRetry,

collectDemandeurData,

collectOrganizationData,

collectCoordinatesData

};

}

console.log('✅ Section O - Configuration Backend chargée avec succès');

</script>

<!-- Configuration Meta pour Laravel -->

<meta name="csrf-token" content="{{ csrf\_token() }}">

<meta name="api-base-url" content="{{ config('app.url') }}/api">

<meta name="max-file-size" content="{{ config('filesystems.max\_file\_size', 5242880) }}">

<meta name="supported-formats" content="pdf,jpg,jpeg,png">

<!-- Scripts Laravel requis -->

@if (config('app.env') === 'local')

<script>

// Mode développement - logging détaillé

window.APP\_DEBUG = true;

console.log('🔧 Mode développement activé');

</script>

@endif

<!-- Gestion des erreurs spécifiques Laravel -->

<script>

// Gestion des erreurs de validation Laravel

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

@if ($errors->any())

const laravelErrors = @json($errors->getMessages());

displayValidationErrors(laravelErrors);

showNotification('Erreurs de validation détectées', 'danger');

@endif

@if (session('success'))

showNotification("{{ session('success') }}", 'success');

@endif

@if (session('error'))

showNotification("{{ session('error') }}", 'danger');

@endif

@if (session('warning'))

showNotification("{{ session('warning') }}", 'warning');

@endif

});

</script>

<!-- ========== FIN SECTION O - CONFIGURATION BACKEND ========== -->