**Moi :**

Voici la synthèse de la Discussion précédente : creat-organisation artefact - 4**. Continuons. Prends en compte que :** Projet Laravel avec PHP 7.3.29 et Bootstrap 5 disponible sur Github à l’adresse : <https://github.com/opolombanyholden/pngdi> Il suit une logique de développement séquentiel avec validation manuelle à chaque action. Ne passe à l'étape suivante qu’après ma validation explicite. Travail étape par étape. À 90 % de la limite de message, génère un fichier synthèse avec tout l’historique, code inclus. Avant de commencer une action, vérifie d’abord si des composants à prendre en compte n’existe pas déjà. Avant de commencer fait également un plan de structure du travail que tu comptes faire. Chaque étape unitaire doit ne doit pas dépasser la limite permise pour une discussion. Avant de commencer vérfie d’abord si les nouveaux composants ou fichiers à créer n’existe pas déjà dans le projet les repository github ou en local N’oublie pas que le projet est bien avancé et qu’il faut toujours prendre connaissance des fichiers existant relié à ce que tu veux créer avant de comment toute action afin d’éviter des erreurs, des reprises inutiles, des pertes de données ou encore des conflits. Les Cahiers de charge complets, la structuration du projet, et certaines synthèses des discussions précédentes sont partagés ci-join : la Discussion creat-organisation artefact - 3 ; fichier 02 - cahier des chargesv20juin2025; fichier 03 - cahier des charges gestion des dossiers \_ 23Juin2025; fichier 04 - Structuration du projet; fichier STRUCTURE DE LA BASE DE DONNÉE ; la copie du fichier original create.blade copy.php ; fichier en cours de traitement : create.blade.php Considère tous les détails.

Je comprends parfaitement. Je vais continuer le travail sur le fichier create.blade.php après la Section J (Étape 9 - Soumission finale) qui a été validée.

## 📋 ANALYSE DE L'ÉTAT ACTUEL

### Fichiers et sections déjà complétés :

* ✅ **Sections A à J** : Les 9 étapes sont complètes dans create.blade.php
* ✅ Structure HTML des formulaires pour chaque étape
* ✅ Modals d'aide et de confirmation
* ✅ Styles CSS de base intégrés

### Ce qu'il reste à faire :

1. **JavaScript principal** pour la gestion des étapes et validations
2. **Styles CSS additionnels** spécifiques aux types d'organisation
3. **Scripts de validation** et gestion des données
4. **Intégration avec le backend Laravel**

## 🎯 PLAN DE STRUCTURE - FINALISATION

### Sections à créer :

1. **Section K** : JavaScript principal de navigation et gestion des étapes
2. **Section L** : JavaScript de validation et gestion des formulaires
3. **Section M** : JavaScript des fonctionnalités spécifiques (upload, adhérents, etc.)
4. **Section N** : Styles CSS additionnels
5. **Section O** : Scripts d'initialisation et configuration

Je vais commencer par la **Section K : JavaScript principal de navigation**.

## ✅ ****SECTION K : JAVASCRIPT PRINCIPAL DE NAVIGATION****

<!-- ========== DÉBUT SECTION K - JAVASCRIPT PRINCIPAL ========== -->

<script>

// Variables globales

let currentStep = 1;

const totalSteps = 9;

let selectedOrgType = '';

let fondateursCount = 0;

let adherentsData = [];

let documentsUploaded = {};

let formData = new FormData();

// Configuration des types d'organisation

const orgTypeConfig = {

association: {

minFondateurs: 3,

minAdherents: 10,

label: 'Association',

color: 'success'

},

ong: {

minFondateurs: 5,

minAdherents: 15,

label: 'ONG',

color: 'info'

},

parti\_politique: {

minFondateurs: 3,

minAdherents: 50,

label: 'Parti Politique',

color: 'warning'

},

confession\_religieuse: {

minFondateurs: 3,

minAdherents: 10,

label: 'Confession Religieuse',

color: 'purple'

}

};

// Initialisation au chargement de la page

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

updateStepDisplay();

initializeEventListeners();

loadSavedData();

});

// Fonction de navigation entre étapes

function changeStep(direction) {

// Sauvegarder les données actuelles

saveCurrentStepData();

if (direction === 1) {

// Validation avant de passer à l'étape suivante

if (!validateCurrentStep()) {

return false;

}

if (currentStep < totalSteps) {

currentStep++;

// Actions spéciales selon l'étape

if (currentStep === 2) {

loadGuideContent();

} else if (currentStep === 3) {

// Pré-remplir avec les données du compte utilisateur si disponibles

prefillDemandeurInfo();

} else if (currentStep === 4) {

// Afficher les sections selon le type d'organisation

showOrgTypeSpecificFields();

} else if (currentStep === 5) {

// Initialiser la géolocalisation

initializeGeolocation();

} else if (currentStep === 6) {

// Ajouter le premier fondateur

if (fondateursCount === 0) {

addFondateur();

}

} else if (currentStep === 8) {

// Générer les champs CNI pour les fondateurs

generateCNIFields();

// Afficher les documents spécifiques

showSpecificDocuments();

} else if (currentStep === 9) {

// Vérifier la complétude et générer le récapitulatif

checkCompleteness();

generateFullSummary();

}

}

} else {

if (currentStep > 1) {

currentStep--;

}

}

updateStepDisplay();

scrollToTop();

}

// Mise à jour de l'affichage

function updateStepDisplay() {

// Masquer toutes les étapes

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

content.style.display = 'none';

});

// Afficher l'étape actuelle

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

if (currentStepElement) {

currentStepElement.style.display = 'block';

}

// Mettre à jour la barre de progression

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

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

if (progressBar) {

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

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

}

// Mettre à jour le numéro d'étape

document.getElementById('currentStepNumber').textContent = currentStep;

// Mettre à jour les indicateurs d'étapes

updateStepIndicators();

// Gérer les boutons de navigation

updateNavigationButtons();

}

// Mise à jour des indicateurs d'étapes

function updateStepIndicators() {

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

const stepNumber = index + 1;

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

if (stepNumber === currentStep) {

indicator.classList.add('active');

} else if (stepNumber < currentStep) {

indicator.classList.add('completed');

// Changer l'icône en check pour les étapes complétées

const icon = indicator.querySelector('.step-icon');

if (icon && !icon.classList.contains('fa-check')) {

icon.className = 'fas fa-check step-icon text-success';

}

}

});

}

// 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 = currentStep === 1 ? 'none' : 'inline-block';

}

// Boutons suivant et soumettre

if (currentStep === 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');

}

}

// Initialisation des écouteurs d'événements

function initializeEventListeners() {

// Type d'organisation (Étape 1)

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

card.addEventListener('click', function() {

selectOrganizationType(this);

});

});

// Géolocalisation (Étape 5)

const geoBtn = document.getElementById('getCurrentLocation');

if (geoBtn) {

geoBtn.addEventListener('click', getCurrentLocation);

}

// Zone type change

const zoneTypeSelect = document.getElementById('org\_zone\_type');

if (zoneTypeSelect) {

zoneTypeSelect.addEventListener('change', toggleZoneFields);

}

// Province change pour charger les départements

const provinceSelect = document.getElementById('org\_province');

if (provinceSelect) {

provinceSelect.addEventListener('change', loadDepartements);

}

// Ajout de fondateur

const addFondateurBtn = document.getElementById('ajouterFondateur');

if (addFondateurBtn) {

addFondateurBtn.addEventListener('click', addFondateur);

}

// Compteurs de caractères

initializeCharCounters();

// Validation en temps réel

initializeRealTimeValidation();

// Auto-sauvegarde

setInterval(autoSaveProgress, 30000); // Toutes les 30 secondes

}

// Sélection du type d'organisation

function selectOrganizationType(card) {

// Retirer la sélection précédente

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

c.classList.remove('active', 'border-success', 'border-3');

});

// Appliquer la nouvelle sélection

card.classList.add('active', 'border-success', 'border-3');

// Cocher le radio button

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

if (radio) {

radio.checked = true;

selectedOrgType = radio.value;

document.getElementById('organizationType').value = selectedOrgType;

}

// Afficher les infos de sélection

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

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

if (selectedInfo && selectedTypeName) {

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

selectedTypeName.textContent = orgTypeConfig[selectedOrgType]?.label || selectedOrgType;

}

// Afficher les contraintes

showTypeConstraints();

}

// Afficher les contraintes du type sélectionné

function showTypeConstraints() {

const constraintsDiv = document.getElementById('typeConstraints');

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

if (!constraintsDiv || !constraintContent || !selectedOrgType) return;

const config = orgTypeConfig[selectedOrgType];

let content = `

<ul class="mb-0">

<li>Nombre minimum de fondateurs : <strong>${config.minFondateurs}</strong></li>

<li>Nombre minimum d'adhérents : <strong>${config.minAdherents}</strong></li>

`;

// Contraintes spécifiques

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

content += `<li class="text-danger"><strong>Un adhérent ne peut être membre que d'un seul parti politique</strong></li>`;

content += `<li>Présence obligatoire dans au moins <strong>3 provinces</strong></li>`;

} else if (selectedOrgType === 'confession\_religieuse') {

content += `<li>Lieu de culte obligatoire avec justificatif</li>`;

content += `<li>Au moins un responsable religieux qualifié parmi les fondateurs</li>`;

} else if (selectedOrgType === 'ong') {

content += `<li>Projet social détaillé obligatoire</li>`;

content += `<li>Budget prévisionnel sur 3 ans requis</li>`;

}

content += `</ul>`;

constraintContent.innerHTML = content;

constraintsDiv.style.display = 'block';

}

// Chargement du contenu du guide (Étape 2)

function loadGuideContent() {

if (!selectedOrgType) return;

// Afficher le bon guide

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

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

});

const selectedGuide = document.getElementById(`guide-${selectedOrgType}`);

if (selectedGuide) {

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

}

// Mettre à jour le titre

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

if (titleElement) {

titleElement.textContent = orgTypeConfig[selectedOrgType]?.label || selectedOrgType;

}

}

// Sauvegarde des données de l'étape actuelle

function saveCurrentStepData() {

const stepData = {};

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

if (!currentStepElement) return;

// Récupérer tous les champs de l'étape

currentStepElement.querySelectorAll('input, select, textarea').forEach(field => {

if (field.name && field.type !== 'file') {

if (field.type === 'checkbox' || field.type === 'radio') {

if (field.checked) {

stepData[field.name] = field.value;

}

} else {

stepData[field.name] = field.value;

}

}

});

// Sauvegarder dans le localStorage

const savedData = JSON.parse(localStorage.getItem('organizationFormData') || '{}');

savedData['step' + currentStep] = stepData;

savedData.currentStep = currentStep;

savedData.selectedOrgType = selectedOrgType;

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

}

// Chargement des données sauvegardées

function loadSavedData() {

const savedData = JSON.parse(localStorage.getItem('organizationFormData') || '{}');

if (savedData.selectedOrgType) {

selectedOrgType = savedData.selectedOrgType;

// Sélectionner le type d'organisation

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

if (card) {

selectOrganizationType(card);

}

}

// Restaurer les données de chaque étape

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

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

if (stepData) {

restoreStepData(i, stepData);

}

}

}

// Restauration des données d'une étape

function restoreStepData(stepNumber, data) {

const stepElement = document.getElementById('step' + stepNumber);

if (!stepElement) return;

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

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

if (field) {

if (field.type === 'checkbox' || field.type === 'radio') {

field.checked = field.value === data[fieldName];

} else {

field.value = data[fieldName];

}

}

});

}

// Scroll vers le haut de la page

function scrollToTop() {

window.scrollTo({

top: 0,

behavior: 'smooth'

});

}

// Auto-sauvegarde

function autoSaveProgress() {

saveCurrentStepData();

showNotification('Progression sauvegardée automatiquement', 'success', 2000);

}

// Afficher une notification

function showNotification(message, type = 'info', duration = 3000) {

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

notification.className = `alert alert-${type} alert-dismissible fade show position-fixed`;

notification.style.cssText = 'top: 20px; right: 20px; z-index: 9999; min-width: 300px;';

notification.innerHTML = `

${message}

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

`;

document.body.appendChild(notification);

setTimeout(() => {

notification.remove();

}, duration);

}

</script>

<!-- ========== FIN SECTION K - JAVASCRIPT PRINCIPAL ========== -->

## 📋 CONTENU DE LA SECTION K CRÉÉE

J'ai créé la Section K avec le JavaScript principal pour la navigation et la gestion des étapes :

### 1. Variables globales

* ✅ Gestion de l'étape courante et du total
* ✅ Type d'organisation sélectionné
* ✅ Compteurs et données
* ✅ Configuration des types d'organisation

### 2. Navigation entre étapes

* ✅ Fonction changeStep() avec validation
* ✅ Actions spécifiques par étape
* ✅ Sauvegarde automatique des données

### 3. Mise à jour de l'interface

* ✅ Affichage/masquage des étapes
* ✅ Barre de progression
* ✅ Indicateurs d'étapes avec icônes
* ✅ Gestion des boutons de navigation

### 4. Gestion des événements

* ✅ Sélection du type d'organisation
* ✅ Écouteurs pour tous les éléments interactifs
* ✅ Auto-sauvegarde toutes les 30 secondes

### 5. Fonctionnalités de sauvegarde

* ✅ Sauvegarde dans localStorage
* ✅ Restauration des données au chargement
* ✅ Gestion par étape

### 6. Utilitaires

* ✅ Notifications temporaires
* ✅ Scroll fluide
* ✅ Configuration centralisée

Cette section gère toute la logique de navigation et l'état global du formulaire.

**Validez-vous cette Section K pour que je passe à la Section L (JavaScript de validation) ?**

Moi :

ou dois je placé la section K dans le fichier ?

Excellente question ! La Section K (JavaScript principal) doit être placée **à la fin du fichier create.blade.php**, après toutes les sections HTML mais **avant la fermeture de la section content**.

Voici où la placer exactement :

<!-- ========== FIN SECTION J - ÉTAPE 9 : SOUMISSION FINALE ========== -->

</form>

</div>

<!-- Boutons de navigation -->

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

<div class="d-flex justify-content-between">

<button type="button" class="btn btn-outline-secondary" id="prevBtn" onclick="changeStep(-1)" style="display: none;">

<i class="fas fa-arrow-left me-2"></i>Précédent

</button>

<div class="ms-auto">

<button type="button" class="btn btn-success" id="nextBtn" onclick="changeStep(1)">

Suivant <i class="fas fa-arrow-right ms-2"></i>

</button>

<button type="submit" class="btn btn-primary d-none" id="submitBtn">

<i class="fas fa-paper-plane me-2"></i>Soumettre la demande

</button>

</div>

</div>

</div>

</div>

</div>

</div>

</div>

<!-- Modal d'aide -->

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

<!-- ... contenu du modal ... -->

</div>

<!-- ========== PLACER LA SECTION K ICI ========== -->

<!-- ========== DÉBUT SECTION K - JAVASCRIPT PRINCIPAL ========== -->

<script>

// ... tout le code JavaScript de la Section K ...

</script>

<!-- ========== FIN SECTION K - JAVASCRIPT PRINCIPAL ========== -->

@endsection

**Structure de placement :**

1. Après tous les modals
2. Avant @endsection
3. Dans des balises <script> directement dans le blade

Cette position permet au JavaScript d'avoir accès à tous les éléments HTML du formulaire qui sont chargés avant.

**Puis-je continuer avec la Section L (JavaScript de validation) ?**

Moi :

suivant, mais n'oublie pas de faire la synthèse lorsque nous serons à 90% de la limite maximal autorisé par discussion. la synthèse doit permettre de continuer le travail dans une autre discussion de manière cohérente avec l'ensemble du projet

Parfait ! Je vais créer la Section L et je surveillerai la limite pour faire la synthèse à 90%.

## ✅ ****SECTION L : JAVASCRIPT DE VALIDATION****

<!-- ========== DÉBUT SECTION L - JAVASCRIPT DE VALIDATION ========== -->

<script>

// Validation de l'étape courante

function validateCurrentStep() {

let isValid = true;

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

if (!currentStepElement) return false;

// Réinitialiser les erreurs

currentStepElement.querySelectorAll('.is-invalid').forEach(field => {

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

});

switch(currentStep) {

case 1:

isValid = validateStep1();

break;

case 2:

isValid = validateStep2();

break;

case 3:

isValid = validateStep3();

break;

case 4:

isValid = validateStep4();

break;

case 5:

isValid = validateStep5();

break;

case 6:

isValid = validateStep6();

break;

case 7:

isValid = validateStep7();

break;

case 8:

isValid = validateStep8();

break;

case 9:

isValid = validateStep9();

break;

}

if (!isValid) {

showValidationErrors();

}

return isValid;

}

// Validation Étape 1 : Type d'organisation

function validateStep1() {

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

if (!typeSelected) {

showNotification('Veuillez sélectionner un type d\'organisation', 'danger');

return false;

}

selectedOrgType = typeSelected.value;

document.getElementById('organizationType').value = selectedOrgType;

return true;

}

// Validation Étape 2 : Guide

function validateStep2() {

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

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

showNotification('Veuillez confirmer avoir lu le guide', 'danger');

if (guideConfirm) {

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

}

return false;

}

return true;

}

// Validation Étape 3 : Informations demandeur

function validateStep3() {

const requiredFields = [

{ id: 'demandeur\_nip', message: 'Le NIP est obligatoire', validator: validateNIP },

{ id: 'demandeur\_civilite', message: 'La civilité est obligatoire' },

{ id: 'demandeur\_nom', message: 'Le nom est obligatoire' },

{ id: 'demandeur\_prenoms', message: 'Les prénoms sont obligatoires' },

{ id: 'demandeur\_date\_naissance', message: 'La date de naissance est obligatoire', validator: validateAge },

{ id: 'demandeur\_lieu\_naissance', message: 'Le lieu de naissance est obligatoire' },

{ id: 'demandeur\_sexe', message: 'Le sexe est obligatoire' },

{ id: 'demandeur\_nationalite', message: 'La nationalité est obligatoire' },

{ id: 'demandeur\_profession', message: 'La profession est obligatoire' },

{ id: 'demandeur\_email', message: 'L\'email est obligatoire', validator: validateEmail },

{ id: 'demandeur\_telephone', message: 'Le téléphone est obligatoire', validator: validatePhone },

{ id: 'demandeur\_adresse', message: 'L\'adresse est obligatoire' },

{ id: 'demandeur\_cni\_numero', message: 'Le numéro de CNI est obligatoire' },

{ id: 'demandeur\_cni\_date\_etablissement', message: 'La date d\'établissement de la CNI est obligatoire' },

{ id: 'demandeur\_cni\_date\_expiration', message: 'La date d\'expiration de la CNI est obligatoire', validator: validateCNIExpiration },

{ id: 'demandeur\_cni\_lieu\_etablissement', message: 'Le lieu d\'établissement de la CNI est obligatoire' }

];

let isValid = true;

const errors = [];

requiredFields.forEach(field => {

const element = document.getElementById(field.id);

if (!element) return;

const value = element.value.trim();

if (!value) {

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

errors.push(field.message);

isValid = false;

} else if (field.validator && !field.validator(value, element)) {

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

isValid = false;

} else {

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

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

}

});

// Validation des checkboxes

const checkboxes = ['demandeur\_info\_exactes', 'demandeur\_contact\_autorise', 'demandeur\_responsabilite'];

checkboxes.forEach(id => {

const checkbox = document.getElementById(id);

if (checkbox && !checkbox.checked) {

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

errors.push('Toutes les déclarations doivent être cochées');

isValid = false;

}

});

if (!isValid && errors.length > 0) {

showNotification(errors[0], 'danger');

}

return isValid;

}

// Validation Étape 4 : Informations organisation

function validateStep4() {

const requiredFields = [

{ id: 'org\_nom', message: 'Le nom de l\'organisation est obligatoire' },

{ id: 'org\_objet', message: 'L\'objet social est obligatoire', minLength: 50 },

{ id: 'org\_objectifs', message: 'Les objectifs sont obligatoires', minLength: 50 },

{ id: 'org\_secteur', message: 'Le secteur d\'activité est obligatoire' },

{ id: 'org\_zone\_intervention', message: 'La zone d\'intervention est obligatoire' },

{ id: 'org\_public\_cible', message: 'Le public cible est obligatoire' }

];

let isValid = true;

requiredFields.forEach(field => {

const element = document.getElementById(field.id);

if (!element) return;

const value = element.value.trim();

if (!value) {

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

showNotification(field.message, 'danger');

isValid = false;

} else if (field.minLength && value.length < field.minLength) {

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

showNotification(`${field.message} (minimum ${field.minLength} caractères)`, 'danger');

isValid = false;

} else {

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

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

}

});

// Validation des langues (au moins une)

const langues = document.getElementById('org\_langues');

if (langues && langues.selectedOptions.length === 0) {

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

showNotification('Sélectionnez au moins une langue de travail', 'danger');

isValid = false;

}

// Validation des activités (maximum 5)

const activites = document.querySelectorAll('.activite-checkbox:checked');

if (activites.length === 0) {

showNotification('Sélectionnez au moins une activité principale', 'danger');

isValid = false;

} else if (activites.length > 5) {

showNotification('Maximum 5 activités principales autorisées', 'danger');

isValid = false;

}

// Validation des checkboxes

const checkboxes = ['org\_info\_valides', 'org\_unicite\_nom'];

checkboxes.forEach(id => {

const checkbox = document.getElementById(id);

if (checkbox && !checkbox.checked) {

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

isValid = false;

}

});

return isValid;

}

// Validation Étape 5 : Coordonnées

function validateStep5() {

const requiredFields = [

{ id: 'org\_province', message: 'La province est obligatoire' },

{ id: 'org\_prefecture', message: 'La préfecture est obligatoire' },

{ id: 'org\_zone\_type', message: 'Le type de zone est obligatoire' },

{ id: 'org\_adresse\_complete', message: 'L\'adresse complète est obligatoire' },

{ id: 'org\_telephone', message: 'Le téléphone est obligatoire', validator: validatePhone },

{ id: 'org\_email', message: 'L\'email est obligatoire', validator: validateEmail }

];

let isValid = true;

requiredFields.forEach(field => {

const element = document.getElementById(field.id);

if (!element) return;

const value = element.value.trim();

if (!value) {

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

showNotification(field.message, 'danger');

isValid = false;

} else if (field.validator && !field.validator(value, element)) {

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

isValid = false;

}

});

// Validation des coordonnées GPS si fournies

const latitude = document.getElementById('org\_latitude');

const longitude = document.getElementById('org\_longitude');

if (latitude && latitude.value && !validateLatitude(latitude.value)) {

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

showNotification('Latitude invalide pour le Gabon (entre -3 et 3)', 'danger');

isValid = false;

}

if (longitude && longitude.value && !validateLongitude(longitude.value)) {

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

showNotification('Longitude invalide pour le Gabon (entre 8 et 15)', 'danger');

isValid = false;

}

// Validation des checkboxes

const checkboxes = ['coordonnees\_exactes', 'justificatif\_disponible', 'acces\_autorise'];

checkboxes.forEach(id => {

const checkbox = document.getElementById(id);

if (checkbox && !checkbox.checked) {

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

isValid = false;

}

});

return isValid;

}

// Validation Étape 6 : Fondateurs

function validateStep6() {

const config = orgTypeConfig[selectedOrgType];

const minFondateurs = config ? config.minFondateurs : 3;

const fondateurs = document.querySelectorAll('.fondateur-card');

if (fondateurs.length < minFondateurs) {

showNotification(`Minimum ${minFondateurs} fondateurs requis pour ${config.label}`, 'danger');

return false;

}

let isValid = true;

const nips = new Set();

fondateurs.forEach((card, index) => {

const requiredFields = [

{ selector: '.fondateur-nip', message: 'NIP' },

{ selector: '.fondateur-nom', message: 'Nom' },

{ selector: '.fondateur-prenoms', message: 'Prénoms' },

{ selector: '.fondateur-fonction', message: 'Fonction' },

{ selector: '.fondateur-date-naissance', message: 'Date de naissance' },

{ selector: '.fondateur-lieu-naissance', message: 'Lieu de naissance' },

{ selector: '.fondateur-sexe', message: 'Sexe' },

{ selector: '.fondateur-nationalite', message: 'Nationalité' },

{ selector: '.fondateur-email', message: 'Email' },

{ selector: '.fondateur-telephone', message: 'Téléphone' },

{ selector: '.fondateur-adresse', message: 'Adresse' },

{ selector: '.fondateur-cni-numero', message: 'Numéro CNI' },

{ selector: '.fondateur-type-piece', message: 'Type de pièce' }

];

requiredFields.forEach(field => {

const element = card.querySelector(field.selector);

if (!element) return;

if (!element.value.trim()) {

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

if (isValid) {

showNotification(`${field.message} manquant pour le fondateur ${index + 1}`, 'danger');

}

isValid = false;

}

});

// Vérification des doublons NIP

const nipField = card.querySelector('.fondateur-nip');

if (nipField && nipField.value) {

if (nips.has(nipField.value)) {

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

showNotification(`NIP en doublon détecté : ${nipField.value}`, 'danger');

isValid = false;

} else {

nips.add(nipField.value);

}

}

});

// Validation des checkboxes

const checkboxes = ['fondateurs\_consentement', 'fondateurs\_documents', 'fondateurs\_engagement'];

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

checkboxes.push('fondateurs\_exclusivite');

}

checkboxes.forEach(id => {

const checkbox = document.getElementById(id);

if (checkbox && !checkbox.checked) {

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

isValid = false;

}

});

return isValid;

}

// Validation Étape 7 : Adhérents

function validateStep7() {

const config = orgTypeConfig[selectedOrgType];

const minAdherents = config ? config.minAdherents : 10;

const adherentsValides = adherentsData.filter(a => a.statut === 'valide').length;

if (adherentsValides < minAdherents) {

showNotification(`Minimum ${minAdherents} adhérents valides requis pour ${config.label}`, 'danger');

return false;

}

// Vérifier s'il y a des doublons internes non résolus

const doublonsInternes = adherentsData.filter(a => a.statut === 'doublon\_interne').length;

if (doublonsInternes > 0) {

showNotification(`${doublonsInternes} doublon(s) interne(s) à corriger`, 'danger');

return false;

}

// Validation des checkboxes

const checkboxes = ['adherents\_consent', 'adherents\_documents'];

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

checkboxes.push('adherents\_exclusivity');

}

let isValid = true;

checkboxes.forEach(id => {

const checkbox = document.getElementById(id);

if (checkbox && !checkbox.checked) {

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

isValid = false;

}

});

return isValid;

}

// Validation Étape 8 : Documents

function validateStep8() {

let isValid = true;

const missingDocs = [];

// Vérifier tous les documents obligatoires

document.querySelectorAll('.document-input[data-required="true"]').forEach(input => {

const docType = input.getAttribute('data-doc-type');

if (!input.files || input.files.length === 0) {

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

const label = input.closest('.document-upload-card')?.querySelector('h6')?.textContent || docType;

missingDocs.push(label);

isValid = false;

} else {

// Vérifier la taille du fichier (max 5MB)

const file = input.files[0];

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

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

showNotification(`${file.name} dépasse la taille maximale de 5MB`, 'danger');

isValid = false;

}

}

});

if (missingDocs.length > 0) {

showNotification(`Documents manquants : ${missingDocs[0]}`, 'danger');

}

// Validation des checkboxes

const checkboxes = ['documents\_authenticite', 'documents\_lisibilite', 'documents\_complets'];

checkboxes.forEach(id => {

const checkbox = document.getElementById(id);

if (checkbox && !checkbox.checked) {

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

isValid = false;

}

});

return isValid;

}

// Validation Étape 9 : Soumission

function validateStep9() {

// Vérifier la complétude globale

const completion = checkGlobalCompletion();

if (completion.percentage < 100) {

showNotification('Veuillez compléter toutes les sections avant de soumettre', 'danger');

return false;

}

// Validation des déclarations finales

const declarations = [

'declaration\_exactitude',

'declaration\_legalite',

'declaration\_documents',

'declaration\_consentement',

'declaration\_responsabilite',

'declaration\_donnees'

];

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

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

}

let isValid = true;

declarations.forEach(id => {

const checkbox = document.getElementById(id);

if (checkbox && !checkbox.checked) {

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

isValid = false;

}

});

if (!isValid) {

showNotification('Toutes les déclarations doivent être acceptées', 'danger');

}

return isValid;

}

// Validateurs spécifiques

function validateNIP(value) {

const nipRegex = /^[0-9]{13}$/;

if (!nipRegex.test(value)) {

showNotification('Le NIP doit contenir exactement 13 chiffres', 'danger');

return false;

}

return true;

}

function validateEmail(value) {

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

if (!emailRegex.test(value)) {

showNotification('Format d\'email invalide', 'danger');

return false;

}

return true;

}

function validatePhone(value) {

// Nettoyer le numéro

const cleaned = value.replace(/\s+/g, '').replace(/^\+241/, '');

const phoneRegex = /^[0-9]{8,9}$/;

if (!phoneRegex.test(cleaned)) {

showNotification('Format de téléphone invalide', 'danger');

return false;

}

return true;

}

function validateAge(dateString) {

const birthDate = new Date(dateString);

const today = new Date();

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

const monthDiff = today.getMonth() - birthDate.getMonth();

if (monthDiff < 0 || (monthDiff === 0 && today.getDate() < birthDate.getDate())) {

age--;

}

if (age < 18) {

showNotification('Vous devez être majeur (18 ans minimum)', 'danger');

return false;

}

return true;

}

function validateCNIExpiration(dateString) {

const expirationDate = new Date(dateString);

const today = new Date();

if (expirationDate <= today) {

showNotification('La CNI est expirée ou expire aujourd\'hui', 'danger');

return false;

}

return true;

}

function validateLatitude(value) {

const lat = parseFloat(value);

return !isNaN(lat) && lat >= -3 && lat <= 3;

}

function validateLongitude(value) {

const lng = parseFloat(value);

return !isNaN(lng) && lng >= 8 && lng <= 15;

}

// Validation en temps réel

function initializeRealTimeValidation() {

// NIP

document.querySelectorAll('input[name\*="nip"]').forEach(input => {

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

if (this.value && !validateNIP(this.value)) {

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

} else if (this.value) {

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

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

}

});

});

// Email

document.querySelectorAll('input[type="email"]').forEach(input => {

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

if (this.value && !validateEmail(this.value)) {

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

} else if (this.value) {

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

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

}

});

});

// Téléphone

document.querySelectorAll('input[type="tel"]').forEach(input => {

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

formatPhoneNumber(this);

});

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

if (this.value && !validatePhone(this.value)) {

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

} else if (this.value) {

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

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

}

});

});

}

// Formatage du numéro de téléphone

function formatPhoneNumber(input) {

let value = input.value.replace(/\D/g, '');

// Si commence par 241, c'est déjà avec l'indicatif

if (value.startsWith('241')) {

value = value.substring(3);

}

// Formater en XX XX XX XX

if (value.length > 0) {

const formatted = value.match(/.{1,2}/g)?.join(' ') || value;

input.value = formatted.substring(0, 11); // Limite à 8 chiffres + 3 espaces

}

}

// Afficher les erreurs de validation

function showValidationErrors() {

const firstError = document.querySelector('.is-invalid');

if (firstError) {

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

firstError.focus();

}

}

// Vérification de la complétude globale

function checkGlobalCompletion() {

const sections = {

infos: checkInfosCompletion(),

fondateurs: checkFondateursCompletion(),

adherents: checkAdherentsCompletion(),

documents: checkDocumentsCompletion()

};

const total = Object.values(sections).reduce((sum, val) => sum + val, 0);

const percentage = Math.round(total / 4);

return {

percentage,

sections,

isComplete: percentage === 100

};

}

function checkInfosCompletion() {

// Vérifier étapes 1 à 5

const savedData = JSON.parse(localStorage.getItem('organizationFormData') || '{}');

let completed = 0;

for (let i = 1; i <= 5; i++) {

if (savedData['step' + i] && Object.keys(savedData['step' + i]).length > 0) {

completed++;

}

}

return (completed / 5) \* 100;

}

function checkFondateursCompletion() {

const config = orgTypeConfig[selectedOrgType];

const minFondateurs = config ? config.minFondateurs : 3;

const fondateurs = document.querySelectorAll('.fondateur-card');

if (fondateurs.length >= minFondateurs) {

return 100;

}

return (fondateurs.length / minFondateurs) \* 100;

}

function checkAdherentsCompletion() {

const config = orgTypeConfig[selectedOrgType];

const minAdherents = config ? config.minAdherents : 10;

const adherentsValides = adherentsData.filter(a => a.statut === 'valide').length;

if (adherentsValides >= minAdherents) {

return 100;

}

return (adherentsValides / minAdherents) \* 100;

}

function checkDocumentsCompletion() {

const requiredDocs = document.querySelectorAll('.document-input[data-required="true"]');

let uploaded = 0;

requiredDocs.forEach(input => {

if (input.files && input.files.length > 0) {

uploaded++;

}

});

if (requiredDocs.length === 0) return 0;

return (uploaded / requiredDocs.length) \* 100;

}

</script>

<!-- ========== FIN SECTION L - JAVASCRIPT DE VALIDATION ========== -->

## 📋 CONTENU DE LA SECTION L CRÉÉE

J'ai créé la Section L avec tout le système de validation JavaScript :

### 1. Validation par étape

* ✅ Fonction principale validateCurrentStep()
* ✅ Validation spécifique pour chacune des 9 étapes
* ✅ Messages d'erreur contextuels

### 2. Validateurs spécifiques

* ✅ NIP (13 chiffres)
* ✅ Email (format valide)
* ✅ Téléphone (format gabonais)
* ✅ Âge (minimum 18 ans)
* ✅ CNI (non expirée)
* ✅ Coordonnées GPS (Gabon)

### 3. Validation en temps réel

* ✅ Formatage automatique des téléphones
* ✅ Validation au blur des champs
* ✅ Classes visuelles (is-valid/is-invalid)

### 4. Gestion des erreurs

* ✅ Scroll vers la première erreur
* ✅ Focus automatique
* ✅ Messages de notification

### 5. Vérification de complétude

* ✅ Calcul du pourcentage global
* ✅ Vérification par section
* ✅ Statut pour la soumission finale

Cette section gère toute la logique de validation du formulaire avec des règles spécifiques selon le type d'organisation.

**Note :** Nous approchons de la limite. Souhaitez-vous que je fasse la synthèse maintenant ou que je continue avec une section de plus ?