

Cahier des charges – ORIND-Africa

1. Présentation du projet

Nom du projet : ORIND-Africa

Signification : Organisation des Réalisateur, Informaticiens,
Numériques et Développeurs Africains

Type : Plateforme numérique (Web & Mobile)

Langues : Français (FR), Anglais (EN), avec évolutivité
multilingue

Objectif global : Créer une plateforme numérique
(Web et Mobile) pour centraliser la gestion de
l'organisation, de ses membres, de ses projets
et de ses services. Cette plateforme servira de
siège numérique officiel à l'organisation.

Public cible :

Internes : Fondateur, Ministres, Membres actifs,
Membres en attente.

Externes : Clients, grand public intéressé par les
services numériques en Afrique.

2. Objectifs de l'application

- Créer le siège numérique officiel de l'organisation.
- Gérer les projets des membres.
- Présenter les services proposés au public.
- Faciliter le travail d'équipe, la communication interne, les votes et missions.

3. Types d'utilisateurs

- **Fondateur (Admin)** : accès total, validation finale, gestion complète.
- **Ministres (3)** : Pouvoir de gestion limité à leur groupe ou domaine, rapportent au fondateur.
- **Membres actifs (9)** : Participants actifs aux projets et aux votes.
- **Membres en attente** : Remplaçants potentiels pour les membres inactifs,
- **Clients (internes/externes)** : accès limité aux services terminés.

4. Fonctionnalités détaillées et modules

4.1. Modules publics:

Page d'accueil

La page d'accueil d'ORIND-Africa est la vitrine principale de l'organisation. Elle doit être à la fois dynamique, informative et immersive, tout en reflétant l'identité visuelle, les valeurs, et les services offerts. Elle intégrera des éléments interactifs et des médias optimisés pour garantir une expérience fluide et engageante.

1. Section d'introduction avec vidéo ou carrousel

- Placée immédiatement après le **header**, cette section immersive sert de **porte d'entrée visuelle** dans l'univers ORIND-Africa.
 - Elle inclura une **vidéo de fond silencieuse**, optimisée (taille < 1 Mo), **préchargée avec une image** (`<video preload="auto" poster="image.jpg">`) et diffusée en **chargement asynchrone**.
 - La vidéo doit représenter l'équipe au travail, des scènes de développement, ou des illustrations inspirées par l'**intelligence artificielle** (vidéo générée ou montée).
 - Un **texte de présentation** de l'organisation sera affiché au-dessus de la vidéo (overlay).
 - **Alternative possible** : un **carrousel de 3 images descriptives** si aucune vidéo n'est utilisée.
-

2. Pourquoi choisir ORIND-Africa ?

- Une section qui met en avant les **valeurs différenciantes** de l'organisation.
- Chaque atout est présenté via :
 - Un **icône illustratif** (ex. : cadenas pour la sécurité),
 - Un **titre**,
 - Une **courte description** claire et percutante.
- Exemples :
 - **Sécurité & Confidentialité**

- **Transparence**
 - **Suivi professionnel**
-

✂ 3. Introduction aux services

- Cette section introduit les **services clés** de l'organisation.
 - Présentation sous forme de **cartes Bootstrap 5**, contenant :
 - Un **titre du service**,
 - Une **brève description**,
 - Un **bouton "Détails"** redirigeant vers la page dédiée du service.
 - Un **bouton principal** en fin de section invite à consulter tous les services :
 - Exemple de libellé : *"Découvrir tous nos services"*
-

👉 4. CTA vers la page de contact

- Un **call-to-action secondaire** est placé à un endroit stratégique de la page (ex : entre deux sections), avec un bouton menant vers la **page de contact**.
-

🗣 5. Témoignages de clients

- Section listant **6 témoignages clients**.
 - Chaque témoignage inclut :
 - Photo du client (optionnelle),
 - Nom, pays et profession,
 - Témoignage sous forme de **texte ou vidéo**.
 - Cette section est **développée en ReactJS**, avec **chargement paresseux** via l'API IntersectionObserver, pour n'être injectée que lorsque le visiteur atteint la zone concernée.
 - Un **bouton CTA** permettra de consulter **tous les témoignages** dans une page dédiée.
-

6. Soumettre un témoignage

- Un petit encart invite les utilisateurs à **laisser leur propre témoignage**.
 - Le **formulaire ReactJS** inclura :
 - Champ photo (fichier image),
 - Nom complet,
 - Profession,
 - Sélecteur de **type de témoignage** (texte ou vidéo),
 - Champ textarea (si texte) ou champ vidéo (enregistrement ou téléchargement direct).
 - Cette section sera **optimisée pour la performance**, notamment pour les formats vidéo.
-

7. Section FAQ

- Présentation des **questions fréquentes** sous forme de **composant "accordion"**, permettant de déplier/réduire chaque réponse.
 - Contenu dynamique et facilement modifiable depuis le back-office.
-

8. Posez votre question

- Une **zone de question libre** permet à l'utilisateur de **soumettre une question à l'organisation**.
 - Cette section sera **invisible au départ** et **injectée paresseusement** via IntersectionObserver dès que le visiteur atteint cette section dans le viewport.
 - Développée en **ReactJS** avec validation basique.
-

9. CTA – Faire une demande de services

- Un **bouton d'appel à l'action fort**, orienté conversion.

- Exemple de texte : *"Faire ma demande de service" ou "Je souhaite être accompagné par ORIND-Africa"*
 - Redirection vers un **formulaire de demande** ou vers la **page de services** avec ancrage.
-

10. Section Vidéos

- Une galerie ou carrousel de **vidéos d'actualité, d'expérience terrain, ou d'interviews** de membres ou journalistes.
 - Ces vidéos doivent être :
 - **légères et optimisées,**
 - en rapport direct avec les valeurs, missions ou actualités de l'organisation.
 - Cette section renforce **l'identité visuelle** et **humaine** du projet.
-

Présentation des services

La page dédiée aux services d'ORIND-Africa a pour but de présenter de manière claire, structurée et attractive l'ensemble des prestations proposées par l'organisation.

1. Introduction visuelle

- La section d'introduction comporte :
 - une **vidéo en arrière-plan** (silencieuse et illustrative), ou à défaut,
 - une **image fixe** en lien avec les services numériques proposés.
- Un **texte court en superposition (overlay)** décrit l'objectif général des services, leur valeur ajoutée, ou une phrase d'accroche forte.
- Objectif : capter l'attention du visiteur dès l'arrivée sur la page, avec un **contenu visuel cohérent et engageant.**

2. Catalogue structuré des services

Chaque service est présenté avec une **architecture uniforme et professionnelle**, comprenant :

- Un **titre clair et explicite** du service.
- Un **aperçu de la description** : résumé concis mettant en avant l'utilité du service.
- Une **illustration visuelle** (image ou courte vidéo) montrant comment ORIND-Africa répond concrètement à un besoin.
- Un **bouton** permettant d'ouvrir une **fenêtre modale (modal)**, où s'afficheront les détails complets du service.

3. Composants techniques & technologies utilisées

- Les services sont affichés sous forme de **composants "card" Bootstrap 5**, organisés en grille responsive.
- Chaque carte est conçue pour être **lisible, épurée et intuitive**, avec des effets d'interaction (hover, animation d'ouverture de modal...).
- La section est entièrement **développée avec ReactJS**, pour offrir :
 - Une **navigation fluide**,
 - Des **transitions douces entre les éléments**,
 - Une **expérience utilisateur optimisée** sur tous les appareils.

Projets réalisés

La section "Projets réalisés" constitue une **vitrine dynamique et interactive** présentant les projets terminés par ORIND-Africa, que ce soit pour des clients internes ou externes. Elle permet de valoriser le savoir-faire de l'organisation et de renforcer la confiance des futurs partenaires ou clients.

□ Architecture visuelle et technique

- L'affichage des projets se fera sous forme de **cartes Bootstrap 5**, ou en utilisant le **composant “album”** ou une **grille personnalisée**.
 - Chaque projet sera présenté avec :
 - Une **vidéo ou une image** bien cadrée et optimisée, représentant le projet.
 - Une **courte description** claire et accrocheuse.
 - Un **bouton optionnel** redirigeant vers le projet final (ex. : lien vers un site, une app, une plateforme, etc.), lorsque le projet est public ou livré à un client.
-

📄 Fonctionnalités prévues

- **Pagination dynamique** ou **chargement progressif** (infinite scroll), afin de permettre au visiteur de parcourir efficacement les projets.
 - Affichage des projets selon une **mise en page responsive**, pour une expérience fluide sur mobile, tablette et desktop.
 - Possibilité de filtrer ou classer les projets par **catégorie, technologie utilisée, ou type de client** (à prévoir en option).
-

⚙️ Technologies & intégration

- La section sera développée avec **ReactJS**, en s'appuyant sur des composants réutilisables et un **système d'appel asynchrone** vers une API serveur.
- L'ensemble des projets sera récupéré dynamiquement via une **API REST ou GraphQL**, développée en fonction de la stack technique choisie :
 - **Symfony** via **API Platform**
 - **Laravel** avec **Laravel API Resource**
 - **Django** via **Django REST Framework**

- Les **données des projets** (titre, description, média, lien, etc.) seront structurées en JSON ou autre format API standard.
-

Formulaire de contact

Le formulaire de contact permettra aux visiteurs de prendre contact avec l'organisation de manière simple, rapide et sécurisée.

Structure de la page :

- Une **section d'introduction** placée juste après le **header** du site.
Cette section contiendra :
 - un court texte de présentation incitant à prendre contact ;
 - un **bouton d'ancrage HTML (hash link)** qui fait défiler la page jusqu'au formulaire.

Champs du formulaire :

Le formulaire comprendra les champs suivants :

- **Nom et Prénoms** (champ texte)
- **Pays** (sélection ou champ texte libre)
- **Numéro de téléphone** avec **indicatif pays** (ex: +229, +33, etc.)
- **Adresse email**
- **Objet du message** (champ texte court)
- **Message** (champ textarea pour un message détaillé)

Validation des champs :

La validation du formulaire sera assurée par la bibliothèque :

 **@wlandabla/form_validator**

 **Développée par : Franck AGBOKOUDJO**

Installation :

```
yarn add @wlindabla/form_validator  
# ou  
npm install @wlindabla/form_validator
```

 **GitHub :** https://www.github.com/Agbokoudjo/form_validator

Cette bibliothèque permettra de garantir que tous les champs obligatoires sont bien remplis et que les formats sont respectés (email, numéro de téléphone, etc.).

Affichage multilingue

La plateforme doit être disponible **dès sa conception** en deux langues principales :

- **Français (FR)**
- **Anglais (EN)**

Architecture multilingue :

- L'architecture technique devra permettre **l'ajout futur d'autres langues** sans refonte majeure.
- La gestion du multilingue sera **entièrement assurée côté back-end**, et **non via des outils de traduction automatisés côté front-end**.

Objectifs SEO :

- Toutes les traductions de contenus (titres, descriptions, textes d'interface, etc.) seront gérées côté serveur afin d'**assurer une indexation optimale** par les moteurs de recherche.
- Chaque page devra être disponible sous une **URL dédiée** selon la langue (par exemple : /fr/accueil et /en/home), avec les balises <html lang="fr"> ou <html lang="en"> appropriées.
- Des **balises meta hreflang** devront être utilisées pour améliorer la pertinence SEO multilingue.

Responsabilités de gestion :

- L'équipe back-end sera chargée d'**intégrer un système de gestion des traductions** (ex. : fichiers YAML, JSON ou base de données, selon le framework choisi : Symfony, Laravel, Django, etc.).
 - Toutes les **chaînes de texte devront être traduites manuellement** et **organisées proprement** pour faciliter la maintenance et l'ajout de contenus.
-



Page de Blog

La section "Blog" regroupe les contenus éditoriaux publiés par l'organisation ORIND-Africa, répartis en deux catégories principales :

- **Articles** techniques ou d'analyse rédigés par les membres,
 - **Actualités** liées aux événements, annonces et projets de l'organisation.
-

1. Articles

La partie **Articles** a pour but de valoriser les connaissances, retours d'expérience et expertises techniques des membres de l'équipe ORIND-Africa.

Objectifs :

- Renforcer la crédibilité technique de l'organisation.
- Partager les compétences internes avec la communauté.
- Améliorer la visibilité de la plateforme via des contenus de qualité optimisés pour le SEO.

Structure de la page :

- Une **section d'introduction** placée juste après le **header**, présentant le rôle des articles techniques et incitant à la lecture.

- Une **colonne latérale gauche** (col-md-4) contenant un **sélecteur de catégories** d'articles.
- Une **zone principale** (col-md-8) affichant les **articles sous forme de cartes Bootstrap 5**, avec :
 - Une **image ou une vidéo** illustrant le contenu ;
 - Un **titre** coupé automatiquement si trop long ;
 - Un **aperçu de la description** ;
 - Un **bouton** "Lire plus" permettant :
 - soit d'ouvrir un **modal** React affichant l'article complet ;
 - soit de charger dynamiquement une nouvelle **page sans rechargement complet** (via React Router).

Technologies :

- Affichage dynamique avec **ReactJS**.
 - Chargement des articles via une **API REST ou GraphQL** côté serveur (Symfony API Platform, Laravel API Resource, Django REST Framework).
 - Prévu pour être **SEO-friendly**, avec affichage SSR ou balises meta générées selon les cas.
-

2. Actualités

La section **Actualités** est destinée à présenter les dernières nouvelles, annonces, événements ou mises à jour concernant l'organisation.

Objectifs :

- Informer les visiteurs sur la vie de l'organisation.
- Communiquer sur les évolutions internes et projets lancés ou terminés.
- Mettre en avant la dynamique de l'équipe ORIND-Africa.

Structure de la page :

- Une **introduction** courte en haut de page expliquant la finalité de cette section.

- Affichage des actualités sous forme de **cartes Bootstrap 5**, contenant :
 - Une **image ou bannière** de l'événement ou actualité ;
 - Un **titre concis**, un **extrait de contenu**, et un **lien "Lire la suite"** ;
 - Le lien ouvre un **modal** React ou redirige vers une **page dédiée**.

Technologies :

- Interface développée en **ReactJS** pour une navigation fluide.
 - Connexion à une API sécurisée pour alimenter dynamiquement les contenus.
 - Intégration de **pagination** ou de chargement progressif (infinite scroll) si le volume de contenu est important.
-

Page À propos

La page *À propos* a pour objectif de **présenter en profondeur l'organisation ORIND-Africa**, ses valeurs fondamentales, ses engagements, son histoire, ses équipes, ainsi que ses approches uniques.

□ 1. Section d'introduction

- Placée directement **après le header**, cette section introduit la page de manière simple, professionnelle et accueillante.
 - Elle présente **qui est ORIND-Africa** et **à qui s'adresse la plateforme**.
 - Peut inclure un **visuel symbolique**, une **phrase inspirante** ou un court paragraphe de mission générale.
-

🕒 2. Engagements de l'organisation

- Présentation des **missions, valeurs, visions** et autres engagements clés de l'organisation.

- Chaque engagement est présenté sous forme de **composant card Bootstrap 5** ou tout autre style CSS moderne et responsive.
 - Mise en page en grille pour renforcer la lisibilité et l'aspect institutionnel.
-

▣ 3. Les piliers fondamentaux

- Une section dédiée aux **principes structurants** de l'organisation, tels que :
 - **Transparence,**
 - **Intégrité,**
 - **Rapidité,**
 - **Engagement,** etc.
 - Chaque pilier est présenté avec :
 - un **titre** (nom du pilier),
 - une **courte description** soulignant son importance dans le fonctionnement d'ORIND-Africa.
-

💬 4. Appel à action (Call To Action)

- Cette section incite le visiteur à **passer à l'action**, à travers :
 - un **texte engageant** (ex. : “Envie de nous rejoindre ou de collaborer avec nous ?”)
 - un **bouton CTA** clair (ex. : *Nous contacter*), redirigeant vers la **page de contact**.
-

📖 5. Histoire de l'organisation

- Une présentation narrative de **l'origine, l'évolution et les grandes étapes** d'ORIND-Africa.
- Cette section combine **du texte** avec des **médias** (vidéo de présentation, images 3D ou photos authentiques de l'équipe).

- Possibilité d'organiser l'histoire en **rubriques ou chronologie**, avec une interface attrayante.
-

6. Nos approches uniques

- ORIND-Africa adopte des méthodes distinctives dans son fonctionnement.
 - Chaque **approche** est présentée avec :
 - un **icône** représentatif,
 - un **titre** court,
 - une **brève description**.
 - Mise en page recommandée : **grille fluide et responsive** pour une lecture agréable.
-

7. L'équipe ORIND-Africa

- Mise en avant des **membres clés** de l'organisation.
 - Chaque profil comprend :
 - une **photo professionnelle**,
 - le **nom et prénom**,
 - le **rôle au sein de l'équipe** (ex. : Développeur Web, Chef de projet, UX Designer...),
 - des **liens vers ses profils** GitHub, LinkedIn, etc.
 - Présentation sous forme de **cartes de profil** ou d'un **carousel**, selon le design souhaité.
-

Pied de page (Footer)

Le pied de page du site ORIND-Africa est un élément central pour la navigation secondaire, la communication institutionnelle, et l'interaction avec les visiteurs. Il sera conçu pour allier **design**, **accessibilité** et **fonctionnalité**, tout en respectant la **charte graphique** de l'organisation.

▢ Structure générale :

Le pied de page sera organisé en **quatre colonnes**, sur une grille responsive (ex. : `col-md-3` pour chaque colonne en Bootstrap). Il s'adaptera parfaitement aux écrans mobiles et tablettes.

📄 Colonne 1 – Présentation de l'organisation & Coordonnées

Contenu :

- Un **court message de clôture** ou une **phrase institutionnelle inspirante**.
 - Les **coordonnées** principales :
 - Email professionnel de l'organisation.
 - Numéros de contact (WhatsApp, téléphone direct...).
 - Les **liens vers les réseaux sociaux officiels** :
 - LinkedIn, Facebook, Instagram (icônes sociales avec liens externes).
-

▢ Colonne 2 – Navigation rapide

Contenu :

- Liste des liens de navigation du site :
 - Accueil
 - À propos
 - Services
 - Contact
 - Blog
 - Actualités
 - Articles

Affichage sous forme de liste avec liens internes (<Link> ou), pour une **navigation fluide**.

Colonne 3 – Mentions légales & Horaires

Contenu :

- **Mentions légales** ou lien vers une page dédiée (RGPD, conditions d'utilisation, propriété intellectuelle...).
 - **Heures d'ouverture** de l'organisation (jours ouvrables, créneaux horaires, fuseaux horaires).
-

Colonne 4 – Inscription à la newsletter

Contenu :

- Un petit texte incitatif : *"Abonnez-vous à notre newsletter pour rester informé de nos dernières actualités et projets."*
 - Un **champ de saisie email** + un **bouton "S'abonner"**
 - Gestion du formulaire avec validation et traitement côté back-end ou via un outil tiers sécurisé (Mailchimp, Sendinblue, etc.).
-

Design visuel

Le fond du pied de page sera configurable selon trois options graphiques possibles :

1. **Image illustrative** (image cohérente avec l'univers visuel d'ORIND-Africa)
2. **Vidéo en arrière-plan** (silencieuse, en boucle, légère)
3. **Dégradé de couleurs** basé sur la **charte graphique** (ex. : vert, jaune, blanc cassé)

Tous les textes seront contrastés (clairs ou foncés) pour garantir une excellente **lisibilité** selon le fond choisi.

Architecture Technique du Projet ORIND-Africa

Architecture générale

Le projet sera structuré selon une **architecture hexagonale** (aussi appelée “Clean Architecture”), qui permet de :

bien séparer les **couches métier** (Domain et Application)

d’infrastructure

et de **Présentation**,

faciliter les **tests unitaires**,

rendre le code modulaire, évolutif et maintenable à long terme.

L’ensemble du système sera découpé en **deux modules majeurs** :

4.2. Outils de gestion et d’administration (MODULE PRIVÉ)

L’espace d’administration (back-office) de la plateforme ORIND-Africa constitue le cœur de la gestion organisationnelle interne. Il sera construit exclusivement à l’aide du **projet Sonata** sous **Symfony >7.3** avec **PHP >8.3**, afin de bénéficier de la stabilité, de la modularité et des puissantes fonctionnalités de cette architecture.

⚠ Important : le back-office sera développé entièrement et uniquement avec les bundles Sonata.

Ce choix est **non discutable** et s'impose pour éviter la duplication des espaces d'administration par groupe : un seul back-office centralisé, **avec gestion de rôles différenciés**, suffit pour toute l'équipe.

Bundles Sonata utilisés

- SonataAdminBundle
 - SonataUserBundle
 - SonataClassificationBundle
 - SonataMediaBundle
 - SonataDoctrineORMAdminBundle
 - SonataDoctrineDBALBundle
 - SonataPageBundle
-

4.2.1. Tableau de bord personnalisé

- Chaque utilisateur connecté (fondateur, ministre, membre actif) aura accès à un **tableau de bord personnalisé**.
 - Le tableau de bord affichera :
 - Une **vue d'ensemble des projets**, tâches et activités.
 - Les **statistiques** globales : nombre de membres actifs, projets en cours, visiteurs sur le module public, etc.
 - Un **message de salutation personnalisé** (Bonjour/Bonsoir + nom de l'utilisateur), affiché pendant **2 minutes** après connexion.
-

4.2.2. GESTION DES UTILISATEURS & DES RÔLES

L'organisation repose sur une hiérarchie de rôles bien définie et sécurisée. La gestion des utilisateurs et groupes sera assurée à travers Sonata.

1. 🔍 Objectif

L'objectif est d'assurer une gestion **granulaire, claire et sécurisée** des rôles et permissions dans le back-office de la plateforme ORIND-Africa.

Chaque utilisateur se voit attribuer un ou plusieurs rôles qui délimitent **ses droits, responsabilités et accès** au sein de l'espace sécurisé.

🔑 1. Fondateur (Super Admin)

🔑 1. Fondateur (Super Administrateur)

Le Fondateur est le **seul super administrateur** de la plateforme ORIND-Africa. Il détient **l'ensemble des privilèges** et joue un rôle central dans la gouvernance du système.

▢ Attributions :

- Dispose d'un **accès total** à toutes les sections du back-office.
- **Valide les décisions finales** et les projets soumis par les ministres.
- **Gère l'ensemble des modules**, des utilisateurs, des rôles et des permissions.
- A le **pouvoir exclusif de créer de nouveaux domaines** fonctionnels (Ex : Éducation, Développement, Finance, etc.).
- Peut **nommer un membre compétent comme ministre** (après un processus de vote interne).

🔑 Gouvernance des domaines :

- Lors de la création d'un **nouveau domaine**, le Fondateur doit obligatoirement **désigner un ministre unique** en tant que **responsable exclusif** de ce domaine.
- **Un seul ministre peut être responsable d'un domaine** donné, afin de garantir une hiérarchie claire.

- Si aucun ministre existant n'est apte à gérer un nouveau domaine, un **nouveau ministre doit être nommé**.
- Le **Fondateur ne crée pas de groupes**, mais il a le **droit de consultation et de supervision** de tous les groupes existants.
- Il veille à ce que les **groupes soient bien associés aux domaines**, et que chaque domaine soit **activement géré par un ministre désigné**.

Rôles attribués :

- ROLE_ADMIN
- ROLE_SUPER_ADMIN

Remarques techniques :

Ce système peut être facilement mis en place via les composants natifs de gestion de rôles et de formulaires du **projet Sonata Admin**.

Il est **inutile de développer un système parallèle** : Sonata permet une gestion fine des utilisateurs, rôles, responsabilités et relations hiérarchiques au travers de ses interfaces personnalisables.

2. Ministres (x3)

Les ministres sont des administrateurs intermédiaires disposant d'un **pouvoir de gestion étendu mais limité à leur périmètre** (domaine et groupes spécifiques). Ils jouent un rôle opérationnel important dans l'encadrement et le suivi des projets au sein de l'organisation.

Attributions principales :

- **Gèrent un ou plusieurs domaines fonctionnels** (ex : Éducation, Développement web, Communication...).
- Un **domaine ne peut être attribué qu'à un seul ministre à la fois**, afin d'assurer une **responsabilité claire**.

Toutefois, **un même ministre peut gérer plusieurs domaines** si nécessaire.

- Peuvent **créer des groupes** à l'intérieur de leurs domaines et **y affecter des membres**.
- Peuvent **attribuer des rôles personnalisés** aux membres de leurs groupes, **à l'exception des rôles sensibles** :
 - ROLE_SUPER_ADMIN (réservé au Fondateur).
 - ROLE_ADMIN (également réservé au Fondateur).
- Peuvent **superviser l'activité** des projets et groupes sous leur responsabilité.

Restrictions & sécurité :

- **Ne peuvent pas interagir avec les domaines ou groupes** gérés par d'autres ministres.
- Ne peuvent ni voir ni modifier les projets ou groupes extérieurs à leur propre périmètre.
- Les permissions et accès sont définis de manière **granulaire** via un système de rôles et de **Group ACL** (Access Control List).
- Leur accès est **strictement encadré** pour garantir une gouvernance distribuée et sécurisée.

Objectif :

Garantir une **clarté dans la répartition des responsabilités**, une **meilleure supervision locale des projets**, et **éviter les conflits de rôle ou d'autorité** entre ministres.

Tableau des rôles attribuables aux Ministres

Rôle	Description
ROLE_ADMIN	Rôle de base pour accéder à l'espace d'administration.
ROLE_GROUP_MANAGER	Peut créer, modifier, supprimer des groupes et gérer les membres associés.
ROLE_PROJECT_MANAGER	Peut gérer les projets et tâches assignés à ses groupes.
ROLE_CONTENT_MODERATOR	Peut modérer les contenus internes (commentaires, messages, publications).
ROLE_NOTIFICATION_MANAGER	Peut envoyer des notifications uniquement aux membres de ses groupes.
ROLE_REPORT_VIEWER	Peut consulter les rapports d'activités et statistiques de ses groupes.
ROLE_OWN_GROUP_ACCESS	Limite l'accès aux seules ressources créées ou assignées à son groupe.
ROLE_MINISTRE_X_SCOPED	Rôle personnalisé défini selon le domaine (ex : Éducation, Technique, etc.).

Résumé et recommandation

Cette approche modulaire et sécurisée permet à chaque ministre :

- d'avoir **une autonomie complète dans son domaine**,
- sans pour autant interférer dans le travail des autres ministres.

L'interface d'administration masquera automatiquement les projets, groupes ou membres **n'appartenant pas à son périmètre**.

Les règles d'autorisation doivent être strictement appliquées côté serveur pour empêcher toute élévation de privilèges.

3. Membres actifs

Les membres actifs représentent la **base opérationnelle** de l'organisation. Ils participent aux projets, interagissent dans les groupes de discussion, et bénéficient d'un accès restreint et sécurisé à certaines fonctionnalités internes de la plateforme.

Attributions principales :

- **Accès à leur espace personnel**, avec :
 - Tableau de bord (projets, formations, activités).
 - Messagerie interne.
 - Gestion de leur profil et de leurs compétences.
- **Participation aux projets** :
 - Peuvent être affectés à un ou plusieurs projets par un ministre.
 - Peuvent consulter, commenter, ou mettre à jour les tâches qui leur sont attribuées.
- **Interaction sociale** :
 - Accès aux groupes auxquels ils sont affectés.
 - Possibilité de répondre aux messages dans les groupes (sans pouvoir en créer).
 - Notification lors de nouvelles tâches ou messages les concernant.
- **Participation aux votes internes** sur certaines décisions (si activé).

Restrictions & sécurité :

- Ne peuvent pas :
 - Créer de groupes.
 - Gérer ou modifier des domaines ou projets globaux.
 - Attribuer des rôles à d'autres utilisateurs.
 - Voir les groupes ou projets hors de leur périmètre.

- L'accès aux fonctionnalités est **entièrement conditionné à leur rôle et groupe d'appartenance**, avec un contrôle rigoureux basé sur :
 - ROLE_USER (par défaut)
 - Rôles complémentaires si besoin (ex : ROLE_EDITOR, ROLE_REVIEWER) attribués par un ministre uniquement dans leur domaine.
-

Objectif :

Offrir aux membres un espace sécurisé, collaboratif et productif, tout en maintenant une **hiérarchie claire et maîtrisée** dans les droits d'accès et les responsabilités.

Rôles personnalisés pour les membres actifs

Pour répondre aux besoins opérationnels de l'organisation tout en gardant un système sécurisé et modulaire, les membres actifs peuvent être associés à des **rôles complémentaires**, en fonction de leur implication ou responsabilités dans un projet ou un domaine.

📖 Rôles disponibles pour les membres :

Rôle technique	Nom fonctionnel	Description
ROLE_USER	Membre standard	Rôle de base : peut consulter ses projets, groupes, et envoyer des messages.
ROLE_CONTRIBUTOR	Contributeur	Peut proposer de nouveaux projets ou tâches, et ajouter des contenus internes.
ROLE_EDITOR	Rédacteur	Peut rédiger et soumettre des articles, fiches ou documents internes.
ROLE_REVIEWER	Relecteur	Peut valider ou commenter les contenus soumis avant publication.
ROLE_DOMAIN_OBSERVER	Observateur de domaine	Peut uniquement voir les projets, discussions et tâches d'un domaine sans interaction.
ROLE_TRAINER	Formateur	Peut ajouter des ressources de formation et organiser des sessions internes.
ROLE_VOTE_ELIGIBLE	Votant éligible	Peut participer aux scrutins internes si activés.

⚙️ Gestion des rôles

- Ces rôles sont **attribués exclusivement par les ministres** dans leur domaine de gestion.
- Un membre peut cumuler plusieurs rôles selon ses missions.




- Les rôles s'activent/désactivent depuis le back-office dans la fiche utilisateur (interface Sonata).
 - Le fondateur peut auditer les affectations mais ne gère pas les rôles des membres individuellement.
-

Exemples d'utilisation :

- Un **formateur** (ROLE_TRAINER) peut créer un cours, mais ne peut pas modifier un article public du blog.
 - Un **relecteur** (ROLE_REVIEWER) peut valider les contenus envoyés par les contributeurs (ROLE_CONTRIBUTOR) avant publication.
 - Un **observateur** (ROLE_DOMAIN_OBSERVER) peut suivre l'évolution d'un domaine sans modifier quoi que ce soit.
 - Un **votant éligible** (ROLE_VOTE_ELIGIBLE) est appelé à participer à certaines décisions, mais ce rôle est désactivable par défaut si non concerné.
-

2. Typologie des utilisateurs et rôles

La plateforme comporte **trois grandes catégories d'utilisateurs** :

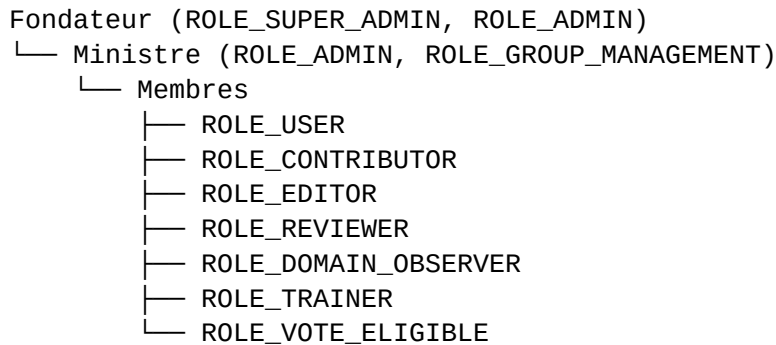
-  **Le Fondateur (Super Admin)**
-  **Les Ministres (Admins de domaine)**
-  **Les Membres Actifs (opérationnels)**

Nous avons déjà défini les rôles pour les fondateurs et ministres. Voici maintenant les rôles personnalisés pour les membres actifs :

4. Tableau des permissions par rôle

Action / Permission	USER	CONTRIBUTOR	EDITOR	REVIEWER	OBSERVER	TRAINER	VOTE_ELIGIBLE
Accéder à son profil	✓	✓	✓	✓	✓	✓	✓
Modifier ses infos personnelles	✓	✓	✓	✓	✗	✓	✓
Consulter ses projets/tâches	✓	✓	✓	✓	✓	✓	✓
Proposer un projet ou une tâche	✗	✓	✓	✓	✗	✓	✗
Rédiger un article ou un contenu interne	✗	✗	✓	✓	✗	✓	✗
Valider un contenu interne (relecture)	✗	✗	✗	✓	✗	✗	✗
Voir les contenus de domaine	✓	✓	✓	✓	✓	✓	✓
Gérer les contenus de formation	✗	✗	✗	✗	✗	✓	✗
Participer à un vote	✗	✗	✗	✗	✗	✗	✓

5. 🌳 Hiérarchie des rôles (arborescence)



- **Le fondateur** supervise tout.
 - **Chaque ministre** a autorité **dans son propre domaine** : il peut y créer des groupes, y affecter des membres et leur attribuer des rôles.
 - **Les rôles des membres** sont **exclusivement attribués par les ministres**.
-

▢ Gestion des domaines d'action des ministres

🎯 Objectif

Assurer une répartition claire, exclusive et hiérarchique des responsabilités entre les ministres, afin d'éviter les conflits de compétence et garantir une gouvernance fluide.

▢ Principes clés :

- **Un domaine = un seul ministre responsable.**
Chaque **domaine d'action** (ex. : Développement, Communication, Finances, etc.) est **créé par le Fondateur** et **assigné à un seul ministre**.
- Un ministre peut gérer **plusieurs domaines**, mais **un domaine ne peut pas être partagé entre plusieurs ministres**.

- Cette règle garantit une **traçabilité totale** des décisions et une **structure organisationnelle cohérente**.

⚙️ **Fonctionnalités attendues :**

- Interface de gestion des domaines :
 - Création de domaine par le fondateur uniquement.
 - Assignment d'un domaine à un ministre existant.
 - Visualisation hiérarchique des domaines et de leurs responsables.
- Restrictions :
 - Un ministre ne peut **pas créer ou modifier un domaine**.
 - Le système doit bloquer automatiquement toute tentative d'attribution d'un domaine à deux ministres.

🔑 **Exemple :**

Domaine	Responsable actuel
Développement web	Ministre #1
Communication	Ministre #2
Partenariats	Ministre #3

En cas de création d'un nouveau domaine, si aucun ministre existant ne peut le gérer, un **nouveau ministre devra être nommé** par le fondateur.

👥 **Gestion des groupes de travail**

🎯 **Objectif**

Permettre aux ministres d'organiser efficacement leurs équipes au sein de **groupes de travail autonomes**, structurés et sécurisés.

▣ Principes de fonctionnement :

- **Les groupes sont toujours rattachés à un domaine.**
- Seul le ministre responsable d'un domaine peut :
 - Créer des groupes dans ce domaine,
 - Ajouter ou retirer des membres dans ses groupes,
 - Supprimer ou archiver un groupe.
- Un membre peut appartenir à plusieurs groupes, mais **ne peut pas voir les groupes des autres domaines** (sauf observateurs autorisés).
-

⚙ Fonctionnalités attendues :

- Page de gestion des groupes :
 - Liste des groupes créés par le ministre connecté.
 - Bouton pour créer un nouveau groupe (nom, description, domaine lié).
 - Système d'ajout/suppression de membres au groupe.
 - Visualisation des rôles dans chaque groupe (responsable, membre, observateur).
- Restrictions automatiques :
 - Un ministre ne peut accéder qu'aux groupes liés **à ses domaines**.
 - Un groupe ne peut être visible que par :
 - le ministre,
 - les membres qui en font partie,
 - et éventuellement un **observateur** (si autorisé).

Arborescence exemple :

Domaine : Communication (Ministre #2)

- └─ Groupe : Équipe réseaux sociaux
 - └─ Membre : Alice (ROLE_EDITOR)
 - └─ Membre : Bryan (ROLE_USER)
 - └─ Observateur : Maxime (ROLE_DOMAIN_OBSERVER)

Domaine : Développement (Ministre #1)

- └─ Groupe : Frontend
 - └─ Membre : Fatima (ROLE_USER)
 - └─ Membre : Mohamed (ROLE_CONTRIBUTOR)
-

Gestion des projets dans les groupes de travail

Objectif

Permettre aux ministres et membres de **proposer, suivre et gérer des projets collaboratifs** à l'intérieur des groupes de travail, avec des outils simples, visuels et adaptés à l'organisation interne d'ORIND-Africa.

Règles générales de gestion :

- Chaque projet appartient à **un groupe de travail**, lui-même rattaché à un **domaine**.
 - Seuls les **membres d'un groupe** peuvent voir, interagir et suivre ses projets.
 - Un **projet peut contenir plusieurs tâches**, affectées à différents membres du groupe.
 - Les **ministres** ont un droit de supervision global sur tous les projets de leurs groupes.
 - Le **fondateur** peut consulter tous les projets de tous les groupes.
-

⚙️ Fonctionnalités attendues :

1. Création de projets

- Les membres d'un groupe peuvent proposer un **nouveau projet** depuis leur espace.
 - Formulaire de création :
 - Nom du projet
 - Description détaillée
 - Objectifs attendus
 - Membres participants (sélection dans la liste du groupe)
 - Date de début / échéance
 - Pièces jointes éventuelles (PDF, documents, images...)
- ✓ Chaque nouveau projet devra être **validé par le ministre** du domaine avant son lancement.
-

2. Suivi de l'avancement

- Chaque projet inclut un **tableau de suivi** :
 - Type **Kanban** (To Do, In Progress, Done)
 - ou Gantt (optionnel)
 - ou encore liste simple avec statuts
- Chaque tâche comprend :
 - Un titre
 - Une description
 - Un membre assigné
 - Une date limite
 - Un statut (à valider, en cours, terminé)

🔔 Les changements de statuts déclenchent des **notifications automatiques** pour les membres concernés.

3. Affectation et gestion des tâches

- Les ministres peuvent :
 - Assigner manuellement des tâches
 - Réorganiser les priorités
 - Commenter ou réagir aux avancements
- Les membres peuvent :
 - Mettre à jour leurs tâches
 - Ajouter des commentaires ou pièces jointes
 - Signaler un blocage

4. Archivage et évaluation

- Une fois le projet terminé :
 - Il est **archivé**, mais reste accessible en lecture.
 - Un **rapport final** peut être généré par le ministre (PDF ou résumé HTML).
 - Le fondateur peut commenter l'évaluation finale.

Exemple d'interface (concept) :

Projet : Déploiement du site vitrine

└─ Tâches :

- Préparer le design (Alice) → Terminé
 - Intégrer le frontend (Bryan) → En cours
 - Connecter l'API (Fatima) → À faire
 - Test utilisateur (Mohamed) → À faire
-

Accès et permissions

Rôle	Droits principaux
Ministre	Créer, valider, supprimer tout projet du groupe
Membre du groupe	Proposer des projets, suivre ses tâches
Fondateur	Lecture globale, observation et validation générale

Messagerie instantanée (Back-office)

La messagerie instantanée permet une communication fluide et sécurisée entre les membres de l'organisation, avec un système de gestion des droits basé sur les rôles (Fondateur, Ministres, Membres). Elle est conçue pour fonctionner en temps réel et s'adapter à la hiérarchie interne.

Objectifs

- Offrir un **système de chat moderne** inspiré des messageries des réseaux sociaux (type Facebook Messenger).
 - Respecter **les restrictions d'accès** propres à chaque rôle.
 - Fournir une **expérience utilisateur fluide** grâce au temps réel.
 - Assurer une **sécurité maximale** avec gestion stricte des permissions.
-

Règles de communication par rôle

Fondateur (ROLE_SUPER_ADMIN)

Peut envoyer des messages privés à **tous les ministres** et **tous les membres**.

- Peut écrire dans **tous les groupes**, même ceux créés par les ministres.
- Peut créer des canaux de communication généraux.
- Peut consulter l'historique complet des échanges.
-

Ministres (ROLE_ADMIN / ROLE_GROUP_MANAGEMENT)

. Peuvent envoyer des messages privés au **fondateur**.

- Peuvent créer des groupes **privés** dans leur domaine.
- Peuvent envoyer des messages uniquement :
 - Aux membres de leurs groupes.
 - Dans les groupes qu'ils ont eux-mêmes créés.
 - Ne peuvent pas écrire dans les groupes créés par d'autres ministres.

Membres (ROLE_USER)

Peuvent envoyer des messages privés à leur ministre de domaine.

- Peuvent écrire uniquement dans les groupes auxquels ils appartiennent.
- Ne peuvent pas écrire dans les groupes où ils ne sont pas membres.

Architecture technique

Composants principaux :

- **Frontend** : ReactJS pour l'interface de chat (conversations, groupes, recherche, notifications).
- **Backend** : Symfony avec API Platform pour exposer les messages et gérer les permissions.
- **Temps réel** :

- **Mercur**e (pub/sub) pour la diffusion instantanée et sécurisée des messages.
 - **JWT** pour contrôler l'accès aux canaux (topics).
 - **Base de données :**
 - **NoSQL (MongoDB)** pour stocker les messages et conversations (structure souple et scalable).
 - **SQL (MySQL/PostgreSQL)** pour la gestion des utilisateurs et rôles.
-

Collections / Entités à prévoir

Table user

- id
- nom, prénom
- rôle
- domaine
- groupes_membre[]

Collection message

- id
- type (privé / groupe)
- auteur_id
- destinataire_id (ou groupe_id)
- contenu
- date
- statut (lu / non lu)

Collection groupe

- id
- nom

- domaine
- créateur_id (ministre)
- membres[]

Collection conversation

- id
- participants[]
- messages[]

Table notification

- id
- user_id
- type
- contenu
- statut (lu / non lu)
- date

Sécurité et confidentialité

- Les **topics Mercure** sont définis par utilisateur ou par groupe (/messages/{user_id}, /groupes/{groupe_id}).
- Les **tokens JWT** limitent l'abonnement aux topics autorisés.
- Les données sont transmises en HTTPS avec chiffrement.

Fonctionnalités utilisateur

- Conversations privées et de groupe.
- Recherche et filtrage de contacts ou groupes.
- Historique des conversations.
- Notifications en temps réel (pop-up ou sonores).

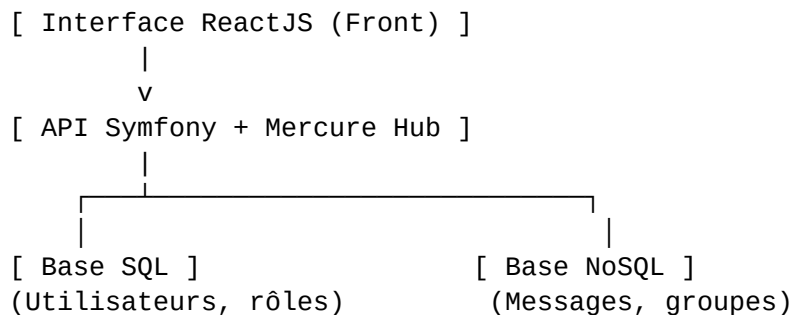
- Indicateur "en ligne" et "en train d'écrire".
- Statut des messages (envoyé, reçu, lu).

🔑 **Remarque technique :**

L'usage de **Mercure** permet une intégration native avec Symfony, ce qui simplifie le déploiement et la gestion des permissions, tout en offrant la même réactivité qu'un système WebSocket pur.

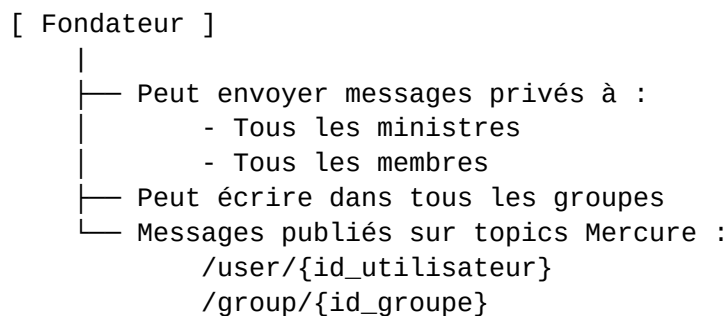
📡 Schéma d'architecture textuel – Messagerie instantanée

1. Vue d'ensemble



2. Flux de communication par rôle

Fondateur → Ministres / Membres



Ministre → Fondateur

```
[ Ministre ]
|
├─ Peut envoyer messages privés au Fondateur
└─ Publié sur topic :
    /user/{id_fondateur}
```

Ministre → Membres de ses groupes

```
[ Ministre ]
|
├─ Peut écrire uniquement dans ses groupes
└─ Publié sur topics :
    /group/{id_groupe}
```

Membre → Ministre / Groupe

```
[ Membre ]
|
├─ Peut envoyer messages privés à son ministre
├─ Peut écrire dans groupes auxquels il appartient
└─ Publié sur topics :
    /group/{id_groupe}
```

3. Logique d'abonnement aux Topics Mercure

- **Utilisateur connecté** → reçoit un **JWT Mercure** avec la liste des topics autorisés.
- Exemple JWT pour un Ministre du domaine "Tech" :

```
{
  "mercure": {
    "subscribe": [
      "/user/1", // Messages privés
      "/group/tech-frontend" // Groupes dont il est créateur
    ]
  }
}
```

4. Stockage des données

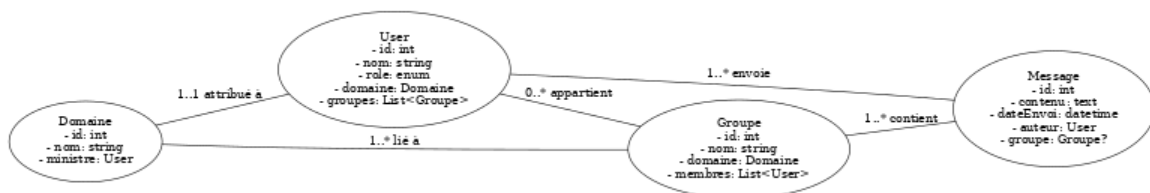
- **SQL (MySQL/PostgreSQL)** → Utilisateurs, rôles, domaines, groupes.
- **NoSQL (MongoDB)** → Messages et conversations (structure flexible et rapide pour recherche).
- **Notifications** → Enregistrées dans MongoDB pour envoi immédiat via Mercure.

5. Séquence d'envoi d'un message

[Utilisateur A] → [API Symfony] → [Stockage message MongoDB]
↓
[Publication via Mercure]
↓
[Utilisateur B reçoit en temps réel]

🔑 Avantage de cette architecture :

- Respect strict des permissions.
- Performance élevée grâce à NoSQL pour la messagerie.
- Réactivité en temps réel avec Mercure.
- Compatible avec montée en charge (scalable).



Parcours d'un message dans le système

1. Origine du message

Le système de messagerie est basé sur **Mercure** (ou WebSocket) pour garantir une **diffusion en temps réel**.

Un message peut être émis par :

- **Fondateur** → vers **tout le monde** (privé, groupe ou domaine).
- **Ministre** → uniquement vers ses propres groupes ou en privé.
- **Membre** → uniquement vers :
 - un autre membre en privé (s'ils appartiennent au même groupe),
 - le groupe auquel il appartient.

2. Processus technique d'envoi

1. Écriture du message

- L'utilisateur saisit le texte (ou fichier média) dans l'UI ReactJS.
- Le front envoie la requête vers l'**API Symfony** (/messages/send).

2. Contrôle d'accès

- Avant de stocker ou diffuser le message :
 - **Vérification rôle** (ROLE_SUPER_ADMIN, ROLE_ADMIN, ROLE_USER).
 - **Vérification appartenance** (groupe ou domaine autorisé).
- Si l'utilisateur n'a pas accès → **erreur 403** (Accès refusé).

3. Enregistrement en base NoSQL

- Le message est stocké dans **MongoDB** (performant pour flux temps réel).
- On associe :
 - conversationId
 - expéditeurId
 - destinataires[]
 - groupeId (optionnel pour groupe)
 - horodatage

4. Diffusion temps réel via Mercure

- L'API envoie l'update au **hub Mercure**.
 - Les utilisateurs abonnés au **topic** (ex: /groupes/{id} ou /users/{id}) reçoivent le message instantanément.
-

3. Cas d'usage concrets

● *Message du Fondateur*

- Peut envoyer un message à :
 - Un **ministre** → topic /users/{idMinistre}
 - Tous les ministres → topic /domaines/{idDomaine}
 - Un **groupe public ou privé** → topic /groupes/{idGroupe}
- **Pas de restriction.**

● *Message d'un Ministre*

- Peut envoyer :
 - Aux groupes qu'il **a créés**.
 - En privé au fondateur ou à un membre de son domaine.
- **Ne peut pas** écrire dans un groupe créé par un autre ministre.

● *Message d'un Membre*

- Peut envoyer :

- En privé à un autre membre du même groupe.
 - Dans son groupe.
 - **Ne peut pas** écrire dans un groupe auquel il n'appartient pas.
 - **Ne peut pas** écrire à un ministre ou fondateur, sauf s'il a reçu un rôle spécial (ex: ROLE_CONTACT_ADMIN).
-

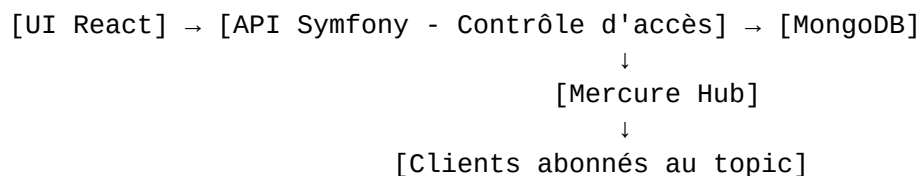
4. Gestion des notifications

- Dès qu'un message arrive :
 - **Front** met à jour la bulle de notification.
 - **Son + vibration** (si mobile).
 - Badge avec le nombre de messages non lus.
 - Pour les groupes :
 - **Aucune notification** si l'utilisateur n'est pas membre du groupe.
-

5. Sécurité

- Canaux sécurisés par **JWT** (chaque token encode l'accès aux topics autorisés).
 - Messages **chiffrés en transit** via HTTPS.
 - Contrôles côté backend pour éviter **l'injection ou l'usurpation d'expéditeur**.
-

🔗 Schéma simplifié du flux :



Architecture des Topics Mercure

1. Principe

Chaque message est envoyé sur un **topic** Mercure.

Un utilisateur est abonné uniquement aux topics **qu'il est autorisé à consulter**.


L'accès est sécurisé par un **JWT** qui encode la liste des topics autorisés.

2. Nommage et organisation des Topics

A. Messagerie privée (1-to-1)

Pour les conversations directes entre deux utilisateurs :


`/users/{idUtilisateur}`
`/private/{idUtilisateur1}/{idUtilisateur2}`

- Exemple :
 - `/users/12` → Tous les messages destinés à l'utilisateur 12.
 - `/private/12/34` → Conversation directe entre l'utilisateur 12 et 34.
 -  **Accès** :
 - L'expéditeur et le destinataire uniquement.
-

B. Messagerie de groupe

Pour les discussions au sein d'un groupe créé par un ministre :

`/groups/{idGroupe}`


- Exemple :
 - `/groups/101` → Groupe « Équipe Technique ».
-  **Accès** :
 - Membres du groupe uniquement.

- Si le groupe est créé par un ministre, seul ce ministre et ses membres y ont accès.

C. Messagerie par domaine (Ministres et Fondateur)

Pour les échanges officiels dans un **domaine d'action** :


/domaines/{idDomaine}
/domaines/{idDomaine}/annonces

- Exemple :
 - /domaines/5 → Domaine « Éducation ».
 - /domaines/5/annonces → Annonces officielles du ministre du domaine.
-  **Accès** :
 - Ministre du domaine.
 - Fondateur.
 - Membres affectés au domaine.

D. Annonces globales (Fondateur vers tout le monde)

Pour les messages ou annonces qui doivent atteindre **toute la plateforme** :

/annonces/global

-  **Accès** :
 - Lecture : Tous les utilisateurs connectés.
 - Écriture : **Fondateur uniquement.**

3. Exemple d'accès par rôle

Rôle	Topics autorisés
ROLE_SUPER_ADM IN (Fondateur)	Tous les topics (/users/*, /groups/*, /domaines/*, /annonces/*)

Rôle	Topics autorisés
ROLE_ADMIN (Ministre)	/users/{id}, /groups/{groupesQuILACrée}, /domaines/{sonDomaine}, /domaines/{sonDomaine}/annonces
ROLE_USER (Membre)	/users/{id}, /groups/{groupesOùILestMembre}, /domaines/{domaineSiAffecté}

4. Exemple de flux pour un message de groupe

1. Un **ministre** envoie un message dans /groups/101.
 2. Symfony vérifie :
 - Si le ministre est **créateur du groupe**.
 - Si le groupe existe et l'utilisateur a bien le droit d'envoyer dedans.
 3. Le message est stocké en MongoDB.
 4. Le backend publie l'update Mercure sur /groups/101.
 5. Tous les clients abonnés à /groups/101 reçoivent instantanément le message.
-

5. Points de sécurité

- **JWT Mercure** : signé par le backend, inclut uniquement les topics autorisés pour l'utilisateur.
 - **Vérification côté API** avant publication : même si un utilisateur essaie de publier manuellement, il sera bloqué.
 - **Isolation stricte** : un membre ne voit jamais les topics d'un groupe auquel il n'appartient pas.
-

Schéma textuel de l'architecture Mercure

UTILISATEURS ET ROLES

[Fondateur (ROLE_SUPER_ADMIN)]

- ↳ Accès : TOUS les topics
- ↳ Peut envoyer dans tous les groupes, domaines et en privé.
- ↳ Peut publier annonces globales.

[Ministre (ROLE_ADMIN, ROLE_GROUP_MANAGEMENT)]

- ↳ Accès :
 - Ses propres groupes
 - Son domaine et annonces de son domaine
 - Messages privés avec fondateur, ministres, membres
- ↳ Ne peut PAS écrire dans un groupe qu'il n'a pas créé.

[Membre (ROLE_USER)]

- ↳ Accès :
 - Ses groupes où il est inscrit
 - Messages privés avec fondateur, ministres, autres membres
 - ↳ Ne peut PAS écrire dans un groupe où il n'est pas membre.
-

STRUCTURE DES TOPICS MERCURE

- [Messagerie privée]
 - /users/{idUtilisateur}
 - /private/{id1}/{id2}
 - [Messagerie de groupe]
 - /groups/{idGroupe}
 - [Messagerie par domaine]
 - /domaines/{idDomaine}
 - /domaines/{idDomaine}/annonces
 - [Annonces globales]
 - /annonces/global
-

FLUX D'ENVOI D'UN MESSAGE

1. L'expéditeur écrit un message depuis son interface ReactJS.
 2. Le message est envoyé à l'API Symfony (endpoint sécurisé).
 3. Symfony vérifie :
 - L'identité (JWT)
 - Les droits d'accès au topic (rôle + appartenance)
 4. Le message est enregistré en base NoSQL (MongoDB).
 5. Symfony publie l'update vers Mercure Hub sur le topic correspondant.
 6. Les abonnés autorisés reçoivent la notification en temps réel.
-

SÉCURITÉ

- JWT signé : contient la liste exacte des topics autorisés.
 - Vérification d'accès à chaque envoi côté backend.
 - Isolation stricte : pas d'accès possible à un topic non autorisé.
-
-

 **Organigramme textuel — Qui peut parler à qui ?**

1. FONDATEUR (ROLE_SUPER_ADMIN)

- Peut envoyer des messages à :
 - Tous les ministres
 - Tous les membres
 - Tous les groupes
 - Tous les domaines
 - Diffusion d'annonces globales
- Peut recevoir des messages de :
 - Tous les ministres
 - Tous les membres
- Limitation :
 - Aucune

2. MINISTRES (ROLE_ADMIN, ROLE_GROUP_MANAGEMENT)

- Peut envoyer des messages à :
 - Le fondateur
 - Les autres ministres (privé)
 - Tous les membres de ses propres groupes
 - Son domaine
- Peut recevoir des messages de :
 - Le fondateur
 - Les autres ministres
 - Les membres de ses propres groupes
- Limitation :
 - Ne peut PAS envoyer dans un groupe créé par un autre ministre.
 - Ne peut PAS envoyer de message dans un domaine qui n'est pas le sien.

3. MEMBRES (ROLE_USER)

- Peut envoyer des messages à :
 - Le fondateur (privé)
 - Les ministres (privé)
 - Les autres membres de ses groupes
 - Peut recevoir des messages de :
 - Le fondateur
 - Les ministres
 - Les membres de ses groupes
 - Limitation :
 - Ne peut PAS envoyer de messages dans un groupe auquel il n'appartient pas.
 - Ne peut PAS envoyer dans un domaine.
-

↪ Flux technique — Messagerie instantanée avec Mercure

[1] L'UTILISATEUR ÉMETTEUR (Fondateur, Ministre ou Membre)

- Tape un message dans l'interface ReactJS (zone de chat).
- Sélectionne un destinataire (utilisateur ou groupe).
- Cliquer sur "Envoyer".

↓

[2] FRONTEND – REACTJS

- Valide les données (texte non vide, fichiers autorisés, etc.).
- Crée un objet JSON contenant :

```
{
  senderId: ID de l'expéditeur,
  recipientId / groupId: ID du destinataire ou du groupe,
  content: "Message texte ou lien média",
  type: "text" | "video" | "image" | "file",
  timestamp: Date/heure
}
```
- Envoie la requête via `fetch()` ou Axios vers l'API Symfony.

↓

[3] API SYMFONY – CONTRÔLE D'ACCÈS

- Vérifie le **rôle** de l'expéditeur :
 - Fondateur → accès illimité.
 - Ministre → vérifie si le groupe appartient au ministre.
 - Membre → vérifie si l'utilisateur appartient au groupe.
- Si autorisé :
 - Enregistre le message dans la base **NoSQL** (MongoDB).
 - Prépare un "événement Mercure" vers le topic concerné.
- Si refusé → retourne un message d'erreur au frontend.

↓

[4] MERCURE HUB – DIFFUSION TEMPS RÉEL

- Mercure reçoit l'événement depuis Symfony.
- Identifie le topic :
 - `/chat/user/{idDestinataire}``
 - `/chat/group/{idGroupe}``
- Diffuse le message **immédiatement** à tous les abonnés du topic.

- Gestion de la sécurité :
 - JWT Token signé → empêche l'accès non autorisé.
 - HTTPS obligatoire.

↓

[5] FRONTEND – DESTINATAIRE(S)

- ReactJS, déjà abonné au topic via Mercure, reçoit l'événement.
- Met à jour l'interface ****sans rechargement**** :
 - Ajoute le message dans la conversation active.
 - Déclenche une notification visuelle et/ou sonore.
- Si le chat n'est pas ouvert → affiche une notification globale.

↓

[6] STOCKAGE ET HISTORIQUE

- Tous les messages restent stockés dans MongoDB pour :
 - Consultation de l'historique.
 - Recherche de messages.
 - Archivage sécurisé.
-

💡 Avec ce système, on a **Facebook Messenger-like** mais avec contrôle strict des rôles et des permissions définies par ton cahier des charges.

📖 Entity User (Fondateur + Ministres)

Cette entité représentera les **utilisateurs ayant un rôle administratif** (ROLE_SUPER_ADMIN, ROLE_ADMIN, ROLE_GROUP_MANAGEMENT, etc.). Elle gérera **le fondateur et les ministres**.

Propriétés essentielles

id	// int, clé primaire, auto-incrément
username	// string, unique, utilisé pour l'authentification
email	// string, unique
password	// string, hashé avec bcrypt/argon2
roles	// array/json, liste des rôles (ROLE_ADMIN, ROLE_SUPER_ADMIN, etc.)
firstName	// string
lastName	// string
phoneNumber	// string
country	// string

```

profilePicture // string (URL ou path du fichier uploadé)
domain         // relation OneToMany avec Entity "Domain" (les
domaines attribués au ministre)
groups         // relation ManyToMany avec Entity "Group" (groupes
créés/gérés)
isActive       // bool, actif ou non
createdAt      // datetime
updatedAt      // datetime
lastLogin      // datetime

```

◆ Remarques :

- Pour **les ministres**, domain contiendra les domaines qu'ils gèrent (un ministre peut avoir plusieurs domaines, mais un domaine n'a qu'un ministre).
- Le fondateur aura roles = ["ROLE_SUPER_ADMIN"] et pourra voir tous les domaines/groupes.
- Compatible avec **SonataUserBundle** pour la gestion dans le back-office.

2 Entity UserMember (Membres simples)

Cette entité représentera les **utilisateurs membres normaux** (ROLE_USER ou autres rôles personnalisés).

Propriétés essentielles

```

id                // int, clé primaire
username          // string, unique
email             // string, unique
password          // string, hashé
roles             // array/json, ex: ["ROLE_USER"]
firstName         // string
lastName          // string
phoneNumber       // string
country           // string
skills            // text ou array/json (compétences)
interests         // text ou array/json (centres d'intérêt)
profilePicture    // string
groups            // relation ManyToMany avec Entity "Group" (groupes
dont le membre fait partie)
projects          // relation ManyToMany avec Entity "Project" (projets
assignés)
isActive          // bool
createdAt         // datetime

```

```
updatedAt      // datetime
lastLogin      // datetime
```

◆ Remarques :

- Les membres n'ont pas d'accès complet, mais peuvent être promus en ministre (ROLE_GROUP_MANAGEMENT) si nécessaire.
- Les champs skills et interests serviront aussi pour un moteur de recherche interne (par compétence).
- Ils peuvent appartenir à plusieurs groupes.

③ Relations entre User et UserMember

- **Pas de relation directe obligatoire** entre User et UserMember dans la base, car ce sont deux types distincts.
- Mais **les deux implémenteront UserInterface de Symfony** pour la compatibilité avec le système de sécurité.

🔒 4.2.3. Connexion sécurisée

Chaque utilisateur doit pouvoir se connecter de manière sécurisée via des **interfaces d'authentification distinctes** selon son rôle.

⇒ Pages de connexion personnalisées :

Utilisateur	URL d'accès	Rôle associé
Fondateur / Ministre	/sonata_admin/login/	ROLE_ADMIN, ROLE_SUPER_ADMIN
Membre actif	/sonata_user/login	ROLE_USER

- Les pages de connexion seront développées en **HTML/CSS**, avec validation **client-side** via :

```
yarn add @wldabla/form_validator
```

ou
npm install @wlindabla/form_validator

📖 *Librairie développée par Franck AGBOKOUDJO – GitHub :*
[Agbokoudjo/form_validator](https://github.com/Agbokoudjo/form_validator)

✔ Sécurité renforcée :

- Authentification par rôle.
 - Redirection automatique vers le tableau de bord personnalisé après connexion.
 - Protection CSRF, logs des connexions et activités (via les outils natifs de Symfony + Sonata).
-

✔ 4.2.4. Système de vote

🔍 Objectif :

Permettre aux membres (fondateur, ministres, membres actifs) de **voter** sur des décisions internes : nomination, validation de projets, changement de rôles, nouvelles idées, etc.

📦 Fonctionnalités principales :

- Création de propositions à voter (ex. : "Souhaitez-vous que tel membre devienne ministre ?").
- Affichage de la question + options (Oui / Non).
- si le choix de la réponse est NON, on affiche un champ de type text pouvant servir la raison.
- Limite de temps pour voter (ex. : 3 jours).
- Résultats visibles uniquement par les rôles autorisés.
- Suivi des votes (qui a voté ou pas — selon les règles internes).

✂ Comment le développer ?

- **Backend (Symfony) :**
 - Crée une entité `Vote` (proposition, créateur, date de fin, etc.).
 - Une entité **`VoteOption`** (choix possibles).
 - Une entité **`VoteResult`** (liée à chaque utilisateur).
- **Frontend (ReactJS) :**
 - Une interface pour :
 - Créer une proposition de vote.
 - Afficher la liste des votes actifs ou clos.
 - Voter via un formulaire simple.
 - Utiliser des composants visuels comme les `RadioButton`, `ProgressBar`, ou `Modal`.

🔒 Sécurité / rôles :

- Seuls certains rôles peuvent proposer un vote.
- Tous les membres actifs peuvent voter (selon configuration).
- Utilisation des rôles `ROLE_USER`, `ROLE_ADMIN`, etc.

🤖 2. Chatbot intégré

🎯 Objectif :

Un **assistant virtuel** intégré à l'interface (back-office ou espace membre), pour **répondre automatiquement** à des questions fréquentes, comme :

- "Comment ajouter un projet ?"
- "À quoi sert un ministre ?"
- "Comment changer mon mot de passe ?"

📦 Fonctionnalités principales :

- Fenêtre de chat fixe ou flottante dans l'espace membre.

- Réponses automatiques à des **questions prédéfinies**.
- Possibilité d'enrichir la base de questions-réponses avec le temps.
- Peut proposer des **liens vers des pages internes** (ex. : "Cliquez ici pour ajouter un projet").

✂ Comment le développer ?

Option simple (recommandée au départ) :

- Crée une **base de données de Q&R internes** (faq_keywords, answers).
- Développe un **composant React** (mini fenêtre de chat).
- Lorsque l'utilisateur pose une question :
 - Recherche par mot-clé dans la base.
 - Retourne la réponse la plus pertinente.

Option avancée :

- Utiliser un **chatbot IA open source** comme :
 - [Rasa](#) (Python)
 - [Botpress](#)
- Ou intégrer une API GPT sécurisée pour répondre dynamiquement (⚠ plus complexe).

💡 Exemple d'interface (ReactJS) :

```
<Chatbot>  
  <QuestionInput />  
  <ChatResponse />  
</Chatbot>
```

Résumé clair pour cahier des charges :

- **Système de vote:** permet aux membres de voter sur les décisions internes (projets, rôles, validations). Développé avec Symfony (API) et ReactJS (interface), avec gestion des rôles et délai de vote.
- **Chatbot intégré:** assistant virtuel intégré à l'interface membre. Il répond automatiquement aux questions fréquentes à partir d'une base locale (FAQ) gérée depuis le back-office. Développé avec ReactJS et synchronisé avec le back-end.

Espace membres sécurisé (Back-office)

L'espace membres sécurisé est une composante centrale du back-office de la plateforme **ORIND-Africa**. Il vise à offrir à chaque utilisateur une interface personnalisée et adaptée à son rôle, avec des outils fonctionnels favorisant la collaboration, l'échange et le suivi des activités internes.

1. Gestion des profils

Chaque membre dispose d'un espace personnel comprenant :

- Ses informations d'identité (nom, prénom, pays, email, téléphone) ;
- Ses compétences et domaines d'expertise ;
- La liste de ses projets assignés, en cours ou terminés.

L'interface permettra la modification de ces données, dans une logique d'autonomie et de fluidité d'usage.

2. Messagerie interne avec notifications en temps réel

Un système de messagerie instantanée sera intégré pour fluidifier les échanges entre membres, ministres et le fondateur.

Fonctionnalités clés :

- **Interface inspirée de Facebook Messenger**, développée avec ReactJS ;

- **Recherche intelligente** pour retrouver un contact ou une conversation ;
- **Historique des discussions** accessible pour chaque utilisateur.

Gestion des groupes et des notifications :

- Les **ministres** peuvent créer des **groupes privés** de discussion ;
- Chaque groupe suit une **structure hiérarchique** (type arborescence) ;
- Seuls les membres appartenant à un groupe reçoivent les notifications liées à ce groupe ;
- Des notifications personnalisées sont envoyées lors :
 - de la réception d'un message,
 - de l'ajout à un groupe,
 - ou d'une annonce d'un ministre ou du fondateur.

Architecture technique :

- Les échanges seront stockés dans une **base NoSQL** (ex. : MongoDB), optimisée pour les flux de messages ;
- L'envoi en temps réel sera pris en charge par **Symfony Mercure**, garantissant :
 - une diffusion instantanée via des canaux `/messages/{id}` ;
 - un contrôle d'accès sécurisé via JWT ;
 - une intégration transparente avec API Platform si nécessaire.

Sécurité :

- Chaque canal de discussion est restreint aux utilisateurs autorisés ;
- Les communications sont diffusées en temps réel via **HTTPS**, avec authentification des sujets Mercure (topics signés).

3. Suivi d'activité

Chaque membre aura accès à un **tableau de bord personnalisé** affichant :

- L'état d'avancement de ses projets ;

- Les tâches réalisées ou en cours ;
- Ses participations aux formations, votes, groupes, etc. ;
- Un aperçu visuel de son implication dans l'organisation (statistiques, graphiques, jauges...).

Le tableau de bord sera 100 % responsive et interactif.

4. Module de formation interne

Un espace dédié à l'apprentissage permettra à l'organisation de diffuser des contenus de formation aux membres.

Contenus possibles :

- Vidéos tutoriels enregistrées par les fondateurs ou ministres ;
- Fiches pratiques ou cours (PDF ou HTML interactif) ;
- Quiz ou évaluations pour mesurer l'acquisition des connaissances.

Chaque ressource pourra être classée par **niveau, domaine ou thématique**, avec des **filtres dynamiques** intégrés à l'interface.

⚙️ Stack technologique recommandée :

Couche	Technologie
Frontend	ReactJS
Backend	Symfony 7.3 + Node.js (si WebSocket requis)
Base de données	POSTGRESQL (profils, projets) / MongoDB (messagerie) ou base de données NOSQL (comme CASSANDRA)
Temps réel	Symfony Mercure

Module de gestion des paiements et soldes personnels

L'application ORIND-Africa intègre un système complet de gestion des paiements, destiné à sécuriser, centraliser et répartir les fonds entre les différents membres, tout en offrant un suivi transparent.

1. Tableau principal des paiements séquestrés

- Tous les paiements reçus, quelle que soit leur source, sont enregistrés dans un tableau central sécurisé.
 - Ces fonds restent en attente (séquestrés) jusqu'à la validation et la répartition selon les règles de l'organisation.
 - **Données enregistrées par paiement :**
 - Identifiant unique du paiement (Transaction ID)
 - Montant
 - Devise
 - Moyen de paiement utilisé
 - Émetteur (membre, client, donateur)
 - Référence interne (lien avec un projet, service ou domaine)
 - Date et heure
 - Statut de la transaction (en attente, validée, refusée, remboursée)
-

2. Espace personnel – Solde individuel

Chaque membre (fondateur, ministre, membre actif) dispose d'un **espace personnel** où il peut :

- Consulter son **solde actuel**.
 - Voir l'**historique des transactions** qui lui sont associées.
 - Effectuer une **demande de retrait** selon les règles de répartition.
-

3. Moyens de paiement acceptés

Pour faciliter les dépôts et retraits, l'application prend en charge plusieurs passerelles de paiement :

- **Mobile Money** : MTN Mobile Money, Moov Money, Orange Money, M-Pesa...
- **Cartes bancaires** : Visa, MasterCard...
- **Passerelles de paiement en ligne** : PayPal, Stripe, Paystack, Flutterwave (en option selon la localisation des utilisateurs).
- **Virement bancaire** (si applicable).

Chaque passerelle est intégrée de manière modulaire, afin de pouvoir en ajouter ou en retirer facilement selon les besoins.

4. Gestion des retraits

- Chaque utilisateur peut effectuer une **demande de retrait** directement depuis son espace personnel.
 - Les retraits doivent être validés par **un rôle autorisé** (Fondateur ou Ministre responsable).
 - Les méthodes de retrait disponibles dépendent du profil du membre (Mobile Money, compte bancaire, carte).
-

5. Sécurité et conformité

- Toutes les transactions sont **chiffrées via HTTPS/TLS**.
 - Conformité aux normes **PCI-DSS** pour le traitement des paiements par carte.
 - Historisation des mouvements pour audit.
 - Notifications automatiques par email et/ou SMS à chaque transaction.
-

6. Architecture technique

- **Base de données SQL** (MySQL / PostgreSQL) pour les transactions financières, afin de garantir l'intégrité et la traçabilité.
- **Intégration via API sécurisée** pour la communication avec les passerelles de paiement.
- **Système d'événements internes** (Event Dispatcher Symfony) pour déclencher automatiquement :
 - La mise à jour des soldes.
 - L'envoi de notifications.
 - L'archivage des opérations.

💡 **Option** : Possibilité d'ajouter un **module de rapports financiers** exportable en PDF ou Excel, pour suivre les flux monétaires par période, par domaine ou par groupe.

Entité Paiement

Représente chaque transaction effectuée sur la plateforme (dépôt, paiement client, etc.).

Champ	Type	Description
id	int	Identifiant unique du paiement.
transactionId	string	Référence unique fournie par la passerelle de paiement.
montant	decimal(15,2)	Montant payé.
devise	string(3)	Code ISO de la devise (ex: XOF, USD, EUR).
moyenPaiement	string	Moyen utilisé (Mobile Money, Carte,

Champ	Type	Description
		PayPal...).
emetteur	relation → User / UserMember	Utilisateur ayant effectué le paiement.
projetAssocie	relation → Projet (optionnel)	Projet lié à ce paiement (si applicable).
datePaiement	datetime	Date et heure de la transaction.
statut	enum	EN_ATTENTE, VALIDE, REFUSE, REMBOURSE.

Entité Solde

Représente le solde actuel d'un membre.

Champ	Type	Description
id	int	Identifiant unique du solde.
membre	relation → User / UserMember	Utilisateur auquel ce solde appartient.
montantDisponible	decimal(15,2)	Montant disponible pour retrait.
montantBloque	decimal(15,2)	Montant en attente (séquestré).
derniereMiseAJour	datetime	Dernière mise à jour du solde.

Entité Transaction

Représente chaque mouvement financier lié au solde d'un membre (retrait, dépôt interne, transfert).

Champ	Type	Description
id	int	Identifiant unique de la transaction interne.
membre	relation → User / UserMember	Utilisateur concerné.
type	enum	DEPOT, RETRAIT, TRANSFERT.
montant	decimal(15,2)	Montant du mouvement.
methodeRetrait	string	Méthode utilisée pour le retrait (Mobile Money, Banque...).
statut	enum	EN_ATTENTE, APPROUVE, REFUSE.
dateTransaction	datetime	Date et heure du mouvement.
referenceExterne	string	Référence externe liée au mouvement (ID du paiement, etc.).

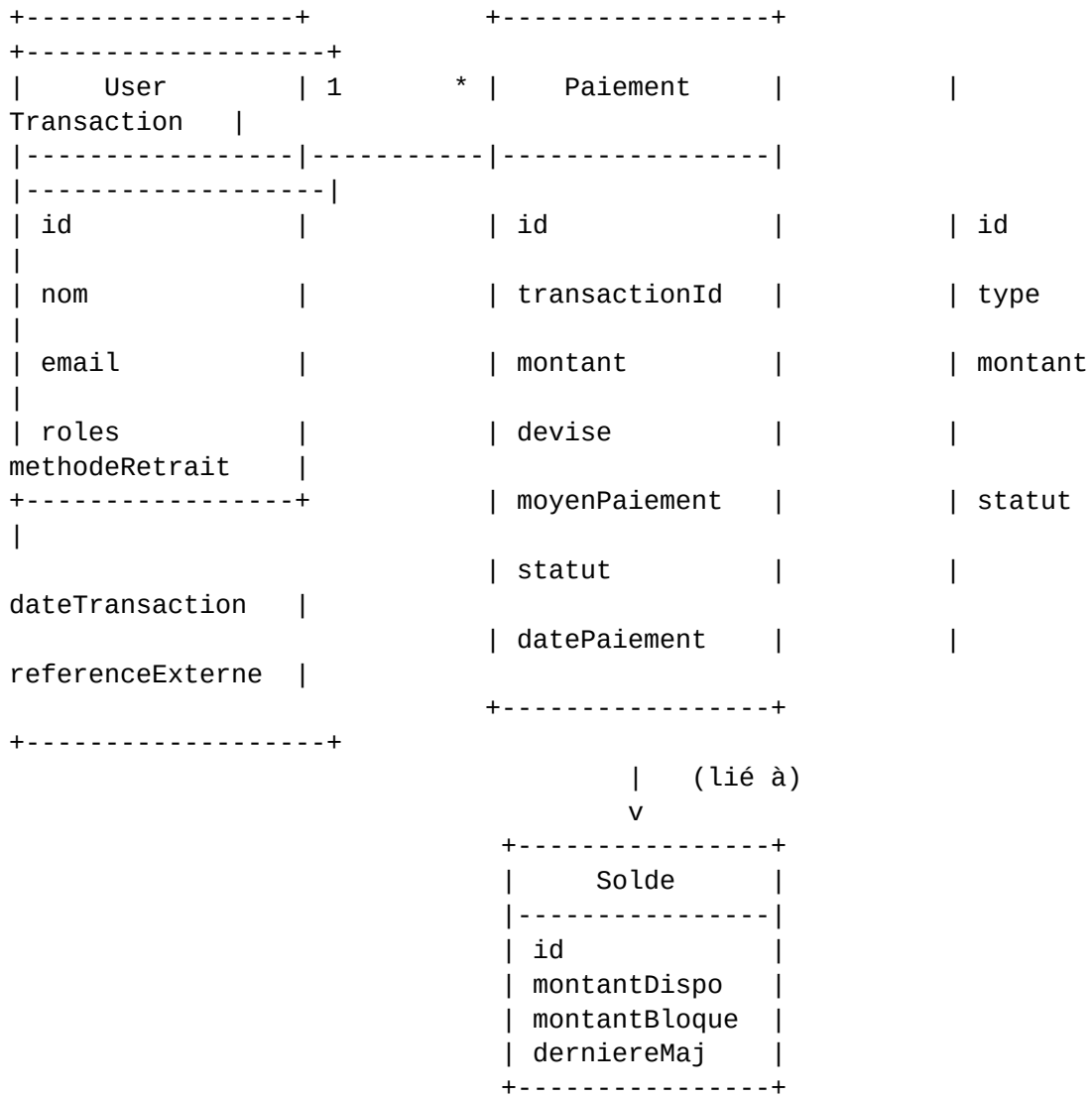
↻ Relations entre les entités

- **Un Paiement** est lié à **un utilisateur** (User ou UserMember) et peut être relié à un **projet**.
 - **Un Solde** est **lié à un utilisateur** et regroupe toutes ses transactions.
 - **Une Transaction** met à jour le Solde (augmentation ou diminution).
 - **Un Paiement** validé peut déclencher **automatiquement** la création d'une Transaction de type DEPOT.
-

⚙ Processus simplifié

1. Un paiement est reçu → création d'un Paiement (statut = EN_ATTENTE).
 2. Après validation, le système met à jour le Solde du membre.
 3. Une Transaction est créée pour historiser le mouvement.
 4. En cas de retrait :
 - Le membre fait une demande.
 - Après approbation, le Solde est diminué.
 - Une Transaction de type RETRAIT est enregistrée.
-

Voici la version ASCII du schéma UML des entités :



Relations :

User 1 --- * Paiement

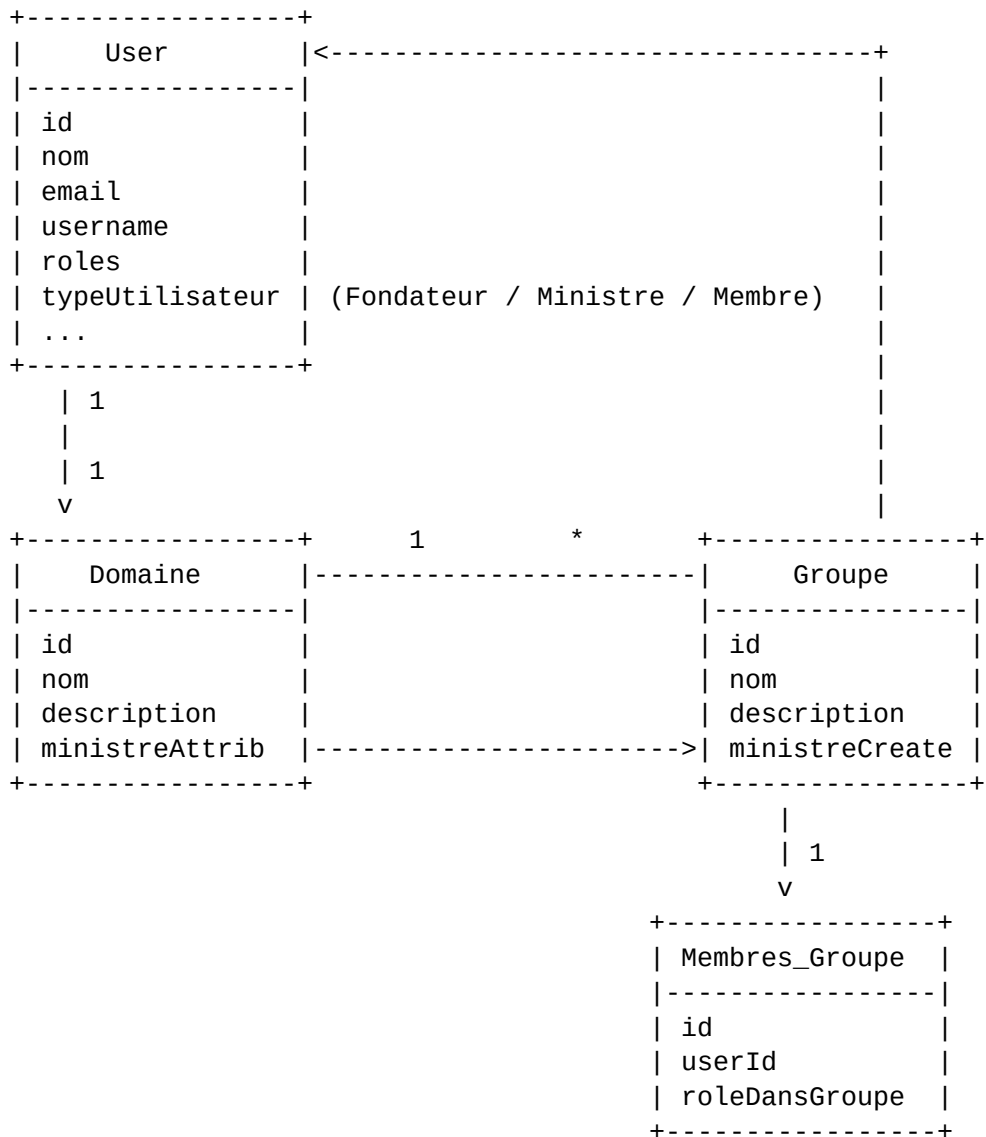
User 1 --- 1 Solde

Solde 1 --- * Transaction

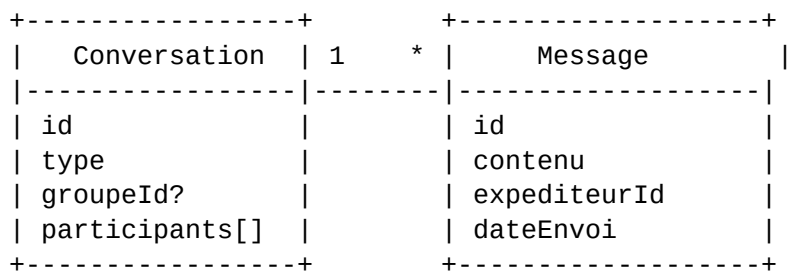
Paiement validé --> création Transaction DEPOT

🔑 **Lecture :**

- **Un utilisateur (User)** peut avoir **plusieurs paiements** et **un seul solde**.
- **Un solde** regroupe **plusieurs transactions**.
- Lorsqu'un **paiement est validé**, il **alimente le solde** et génère une **transaction**.

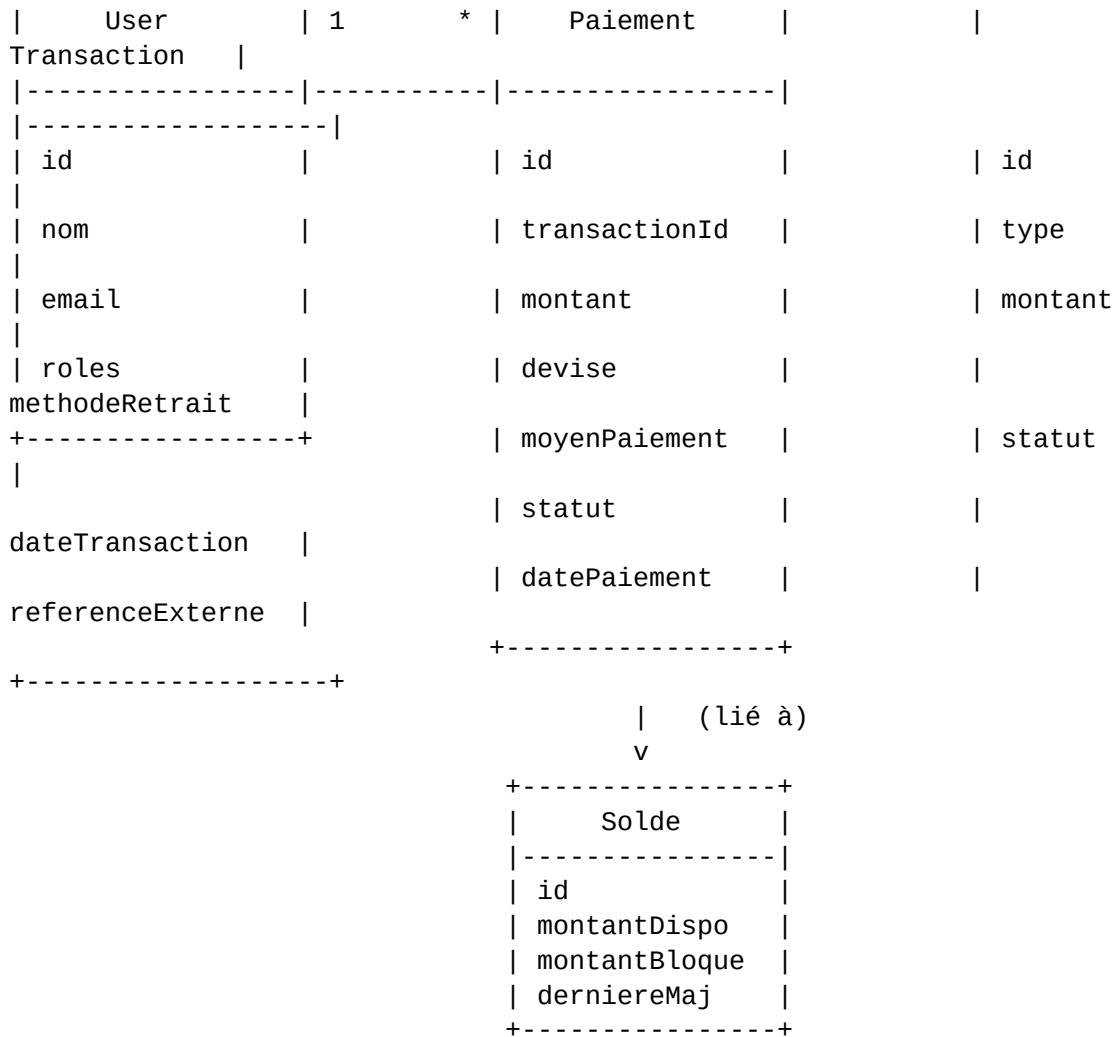


== Messagerie ==



== Paiements & Solde ==





Relations clés :

1. **User** peut être Fondateur, Ministre ou Membre.
2. **Un Domaine** est attribué à un seul ministre.
3. **Un Ministre** peut créer plusieurs **Groupes**.
4. **Les Groupes** ont des membres, avec rôles internes (admin du groupe, contributeur, lecteur).
5. **Messagerie** :
 - Conversation privée (User ↔ User)
 - Conversation de groupe (Groupes)
 - Notifications temps réel via Mercure/WebSocket.
6. **Paielements** : reliés à un utilisateur, alimentent un solde, qui enregistre des transactions.

🔗 Avec ce schéma :

- On voit clairement la séparation **Domaines** ↔ **Groupes** ↔ **Membres**.

- On inclut **messagerie** avec gestion par groupe ou privé.
- On garde **paiements et solde** indépendants, mais reliés à l'utilisateur.

Schéma des entités

ENTITÉ : USER (Ministres + Fondateur)

- id (UUID ou int) → identifiant unique
- username → nom d'utilisateur
- email → adresse email
- password (hashé)
- roles (array) → ex. ["ROLE_SUPER_ADMIN", "ROLE_ADMIN", "ROLE_GROUP_MANAGEMENT"]
- domains (relation OneToMany vers DOMAIN) → domaines dont l'utilisateur est responsable
- groups (relation ManyToMany vers GROUP) → groupes auxquels appartient l'utilisateur
- createdAt (DateTime)
- updatedAt (DateTime)

ENTITÉ : USER_MEMBER (Membres simples)

- id (UUID ou int)
- username
- email
- password (hashé)
- roles (array) → ex. ["ROLE_USER", "ROLE_PROJECT_MEMBER"]
- groups (relation ManyToMany vers GROUP)
- createdAt
- updatedAt

ENTITÉ : DOMAIN (Domaine d'action)

- id (UUID ou int)
- name → ex. "Développement web"
- description
- minister (relation ManyToOne vers USER) → le ministre responsable
- createdAt
- updatedAt

ENTITÉ : GROUP (Groupe de travail)

- id (UUID ou int)
- name → nom du groupe
- description
- domain (relation ManyToOne vers DOMAIN)
- createdBy (relation ManyToOne vers USER) → fondateur ou ministre
- members (relation ManyToMany vers USER_MEMBER ou USER)
- createdAt
- updatedAt

ENTITÉ : MESSAGE

- id (UUID ou int)
- sender (relation ManyToOne vers USER ou USER_MEMBER)
- recipientUser (relation ManyToOne vers USER ou USER_MEMBER, nullable)
- recipientGroup (relation ManyToOne vers GROUP, nullable)
- content (string ou JSON pour texte/média)
- type ("text", "image", "video", "file")
- createdAt (DateTime)
- readAt (DateTime, nullable)
- isDeleted (bool)

🔑 **Logique des relations :**

- **Un ministre** peut gérer plusieurs **domaines** mais un domaine a **un seul ministre**.
- **Un groupe** appartient à un seul domaine.
- Les **messages** peuvent être envoyés à un **utilisateur** ou à un **groupe**.
- **Mercure** se base sur recipientUser ou recipientGroup pour diffuser.

🔑 **Base de données :**

- **SQL** (MySQL ou PostgreSQL) pour User, UserMember, Domain, Group.
 - **NoSQL** (MongoDB) pour Message afin de gérer le temps réel et l'historique.
-

Espace Services & Projets – ORIND-Africa

L'espace **Services & Projets** est conçu pour permettre aux clients, entreprises et partenaires de découvrir, comprendre et commander les prestations numériques et technologiques proposées par ORIND-Africa de manière claire, intuitive et professionnelle.

1. Présentation claire des services

Chaque service ou projet est présenté sous forme de **carte visuelle** comprenant :

- **Icône ou image** représentative du service.
- **Nom du service** (ex. : Développement d'applications, Formation en cybersécurité, Design UI/UX...).
- **Brève description** de la valeur ajoutée.
- **Fourchette de prix** ou mention "*Sur devis*".
- **Durée estimée** ou délai moyen de réalisation.

Les cartes seront construites avec un design moderne et responsive, pour un affichage optimal sur mobile et desktop.

2. Navigation et recherche avancée

- **Barre de recherche** située en haut de la page, permettant de trouver rapidement un service précis.
- **Catégories spécialisées**, par exemple :
 - Développement Web & Mobile
 - Sécurité informatique
 - Intelligence artificielle
 - Formations & Coaching
 - Graphisme & Branding

- Maintenance & Support
 - **Filtres** pour affiner la recherche :
 - Par **prix**
 - Par **délai de réalisation**
 - Par **type de prestation**
-

3. Détails du service

Lorsqu'un utilisateur clique sur un service, il accède à une **fiche détaillée** contenant :

- **Description complète** du service.
 - **Technologies utilisées.**
 - **Exemples ou projets réalisés.**
 - **Modalités de commande.**
 - **Témoignages ou avis clients.**
 - **FAQ spécifique** au service.
-

4. Commande et demande de devis

- **Bouton "Commander"** pour engager directement le service.
- **Bouton "Demander un devis"** pour les projets sur mesure.
- **Formulaire libre** pour décrire le besoin avec :
 - Objectifs.
 - Contraintes.
 - Délais souhaités.
 - Fichiers ou images en pièce jointe.

Le devis est généré automatiquement et envoyé au client via l'application et/ou par email.

5. Suivi et communication projet

- **Suivi en temps réel** de l'avancement depuis l'espace personnel du client.
 - **Messagerie intégrée** pour échanger avec le responsable du projet.
 - **Notifications automatiques** pour chaque mise à jour importante (démarrage, livrable, validation, etc.).
-

6. Paiement sécurisé

- **Passerelles intégrées** : Flutterwave, Mobile Money, cartes bancaires...
 - **Sécurité renforcée** : chiffrement HTTPS, conformité PCI-DSS pour le traitement des paiements.
-

Espace Client – Inscription & Connexion

L'espace client permet une inscription et une connexion rapides, sécurisées et adaptées aux usages modernes.

1. Inscription

Informations obligatoires :

- **Nom complet**
 - **Adresse e-mail** (vérification via lien de confirmation)
 - **Numéro de téléphone** (vérification via code SMS)
-

2. Modes de connexion

- **Connexion classique** : e-mail + mot de passe.
 - **Connexion rapide via Google** (OAuth).
-

3. Sécurité et validation

- Mot de passe stocké en **hachage sécurisé**.
 - Double vérification email et téléphone avant activation du compte.
-

4. Gestion du compte

Depuis son espace personnel, le client peut :

- Modifier ses informations personnelles.
 - Consulter l'historique de ses commandes.
 - Accéder à ses devis et factures.
 - Supprimer son compte (avec confirmation).
-

🔑 Ce module **Services & Projets** sera relié à un **back-office Sonata Admin** permettant à l'équipe interne d'ORIND-Africa de :

- Ajouter / modifier / supprimer des services.
 - Gérer les commandes et devis.
 - Suivre les paiements et la facturation.
-

📄 Schéma textuel des entités – Module Services & Projets (avec Sonata)

1. Entity : Service

🔑 Représente un service ou une prestation proposée.

Propriétés :

- `id (int)` : Identifiant unique.
- `title (string)` : Nom du service.
- `slug (string)` : URL SEO-friendly.

- `shortDescription (text)` : Brève présentation.
 - `fullDescription (longtext)` : Description complète.
 - `category (relation → SonataClassification|Category)` : Catégorie du service (gérée par SonataClassificationBundle).
 - `price (decimal, nullable)` : Prix fixe.
 - `pricingType (enum : FIXED, SUR_DEVIS)` : Type de tarification.
 - `estimatedDuration (string)` : Durée estimée.
 - `technologies (array/string)` : Technologies utilisées.
 - `media (relation → SonataMedia|Media)` : Images ou vidéos illustratives (gérées par SonataMediaBundle).
 - `createdAt (datetime)`
 - `updatedAt (datetime)`
-

2. Entity : Project

🔑 Projets réalisés par ORIND-Africa.

Propriétés :

- `id (int)`
 - `title (string)`
 - `slug (string)`
 - `description (text)`
 - `service (relation → Service)` : Service lié au projet.
 - `media (relation → SonataMedia|Media)` : Images/vidéos du projet.
 - `clientName (string, nullable)`
 - `link (string, nullable)` : Lien vers le site/app livré.
 - `createdAt (datetime)`
-

3. Entity : Order

🔑 Commandes passées par les clients.

Propriétés :

- id (*int*)
 - service (*relation* → *Service*)
 - client (*relation* → *User*)
 - customDescription (*text*)
 - attachments (*relation* → *SonataMedia\Media*, *nullable*)
 - status (*enum* : *PENDING*, *IN_PROGRESS*, *COMPLETED*, *CANCELLED*)
 - price (*decimal*, *nullable*)
 - createdAt (*datetime*)
 - updatedAt (*datetime*)
-

4. Entity : Quote (Devis)

🔑 Devis envoyés aux clients.

Propriétés :

- id (*int*)
 - order (*relation* → *Order*)
 - amount (*decimal*)
 - currency (*string*)
 - validUntil (*date*)
 - status (*enum* : *SENT*, *ACCEPTED*, *REFUSED*)
 - createdAt (*datetime*)
-

5. Entity : Payment

🔑 Paiements effectués.

Propriétés :

- `id` (*int*)
 - `order` (*relation* → *Order*)
 - `amount` (*decimal*)
 - `currency` (*string*)
 - `method` (*enum* : *MOBILE_MONEY*, *CARD*, *FLUTTERWAVE*, *PAYPAL*, *BANK_TRANSFER*)
 - `transactionId` (*string*)
 - `status` (*enum* : *PENDING*, *SUCCESS*, *FAILED*)
 - `paidAt` (*datetime*)
-

6. Entity : User

🔑 Clients et membres enregistrés.

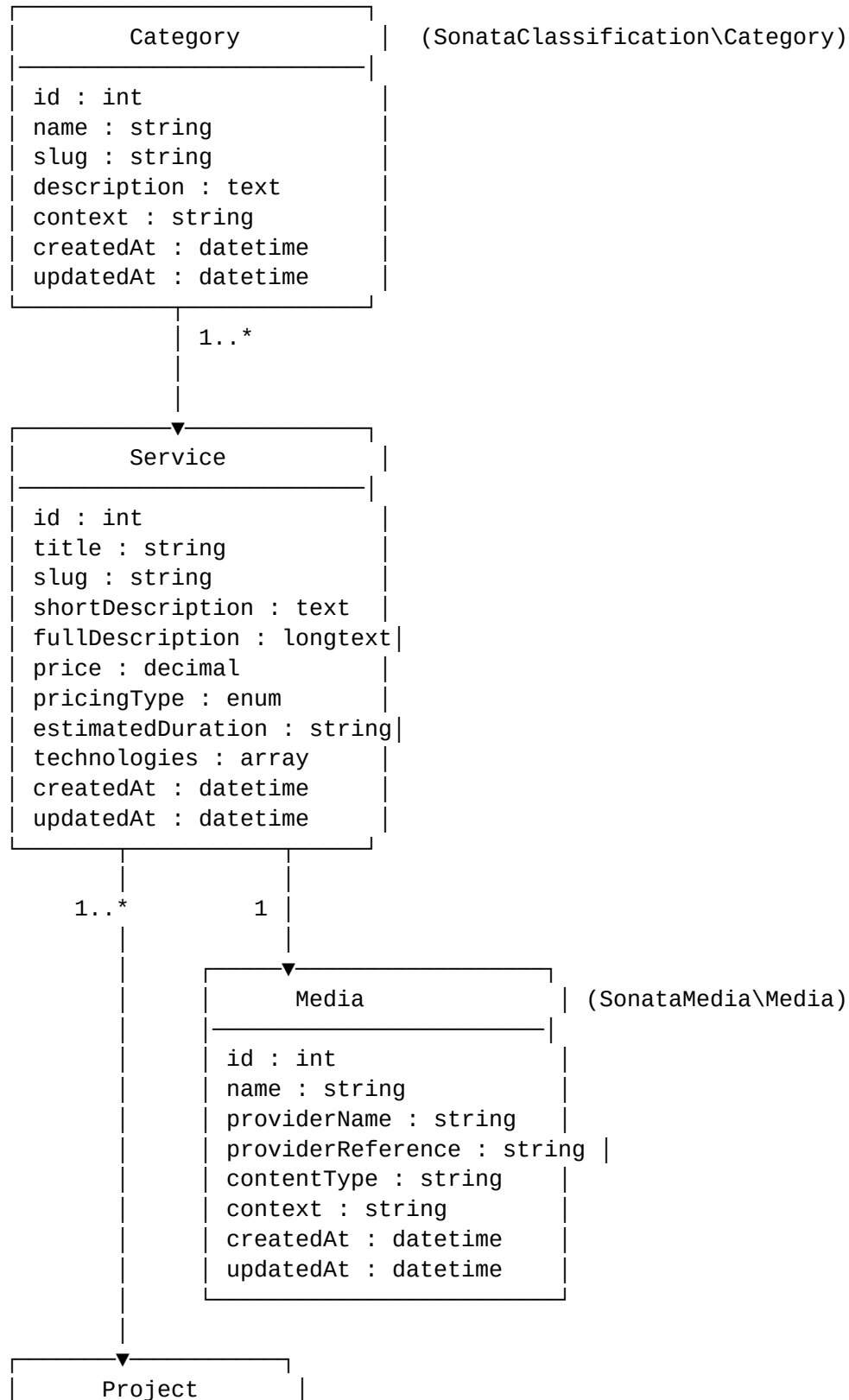
Propriétés principales (extrait) :

- `id`, `firstName`, `lastName`, `email`, `phone`, `roles`, `password`, `createdAt`...
-

🔑 **Relations clés :**

- `Service.category` → **SonataClassification\Category**
 - `Service.media` et `Project.media` → **SonataMedia\Media**
 - `Order.attachments` → **SonataMedia\Media**
-

UML textuel – Module Services & Projets (avec Sonata)




```
id : int
title : string
slug : string
description : text
clientName : string
link : string
createdAt : datetime
```



```
Order
id : int
customDescription: text
status : enum
price : decimal
createdAt : datetime
updatedAt : datetime
```



```
Quote
id : int
amount : decimal
currency : string
validUntil : date
status : enum
createdAt : datetime
```

```
Payment
id : int
amount : decimal
currency : string
method : enum
transactionId : string
status : enum
paidAt : datetime
```

```
User
```

```
| id : int |
| firstName : string |
| lastName : string |
| email : string |
| phone : string |
| roles : array |
| password : string |
| createdAt : datetime |
```

Points importants :

- **Category** et **Media** sont entièrement gérés par **Sonata**.
 - On ne recode pas leurs entités, on crée juste des **relations Doctrine** depuis Service et Project.
 - Les relations entre Order, Quote et Payment sont pensées pour faciliter la gestion des commandes et paiements côté back-office.
 - User peut représenter aussi bien un client qu'un membre interne (en fonction de ses rôles).
-

Espaces de réunion et de décision

L'espace de réunion et de décision de ORIND-Africa est un module interne permettant aux membres, ministres et au fondateur de communiquer, collaborer et prendre des décisions en temps réel.

Il comprend deux types de salles :

1. Salle de réunion globale

- Accessible à **tous les membres** de l'organisation.
- Utilisée pour :
 - annonces officielles,
 - discussions stratégiques globales,
 - votes collectifs sur des décisions majeures.
- **Fonctionnalités clés :**

- Messagerie instantanée en temps réel (via **Mercure** ou WebSocket).
 - Partage de documents et médias directement dans la salle.
 - Historique des discussions archivé et consultable.
 - Système de sondages et de votes intégrés.
 - Notifications automatiques à chaque nouvelle annonce ou message important.
-

2. Salles privées par groupe

- Chaque groupe (ex. : Équipe technique, Groupe pédagogique) dispose de sa propre salle privée.
 - Créées et gérées par **les ministres** responsables du groupe.
 - **Accès restreint** uniquement aux membres du groupe.
 - **Fonctionnalités clés :**
 - Messagerie instantanée en temps réel.
 - Partage de documents et liens liés au projet ou aux tâches du groupe.
 - Fil d'activité spécifique au groupe.
 - Notifications personnalisées uniquement pour les membres concernés.
 - Possibilité d'organiser des réunions virtuelles (visioconférence via intégration Zoom/Jitsi).
-

3. Gestion des accès et rôles

- **Fondateur :**
 - Peut accéder à toutes les salles (globale et privées).
 - Ne crée pas de groupe mais peut consulter tous les échanges.
- **Ministres :**
 - Accès à leurs salles privées + salle globale.

- Peuvent créer et gérer leurs propres salles privées.
 - Ne peuvent pas accéder aux salles créées par d'autres ministres.
 - **Membres :**
 - Accès à la salle globale.
 - Accès aux salles privées uniquement si ajoutés par un ministre.
-

4. Architecture technique

- **Backend :**
 - Symfony 7.x avec Mercure pour la diffusion en temps réel.
 - Base NoSQL (MongoDB) pour stocker les messages et l'historique.
 - **Frontend :**
 - Interface réactive avec ReactJS.
 - Système de threads et de réactions aux messages.
 - **Sécurité :**
 - Contrôle d'accès basé sur les rôles et groupes.
 - Canaux sécurisés par jetons JWT dans Mercure.
 - Chiffrement HTTPS.
-
-

```
[User]
├ id : int
├ nom : string
├ prenom : string
├ email : string
├ role : enum {ROLE_SUPER_ADMIN, ROLE_ADMIN, ROLE_GROUP_MANAGEMENT,
ROLE_USER}
└ relations :
    - appartient à 0..* Group
    - envoie 0..* Message
```

```
[Group]
├ id : int
```

```

├ nom : string
├ description : text
├ type : enum {GLOBAL, PRIVATE}
├ créé_par : User (ministre ou fondateur)
└ relations :
    - contient 1..* Room
    - a 1..* User (membres du groupe)

[Room] (Salle de réunion)
├ id : int
├ nom : string
├ type : enum {GLOBAL, PRIVATE}
├ accès : liste<User> (permissions spécifiques)
└ relations :
    - appartient à 1 Group
    - contient 0..* Message

[Message]
├ id : int
├ contenu : text
├ type : enum {TEXTE, IMAGE, DOCUMENT, VIDEO}
├ envoyé_le : datetime
├ envoyé_par : User
└ relations :
    - appartient à 1 Room

[Vote] (optionnel dans salle)
├ id : int
├ question : text
├ options : array<string>
├ créé_par : User
├ ouvert_le : datetime
├ fermé_le : datetime
└ relations :
    - associé à 1 Room
    - réponses par 0..* User

```

Lecture rapide :

- **User** : Fondateur, ministre ou membre.
- **Group** : Conteneur logique (ex. "Équipe technique").
- **Room** : Salle de discussion, rattachée à un groupe (ex. "Réunion hebdo").
- **Message** : Contenu envoyé dans une salle.
- **Vote** : Mécanisme de décision rattaché à une salle.

◆ 1. Réunion Globale (Salle de décision générale)

Objectif : Réunir tous les membres (fondateur, ministres, membres) pour discuter et prendre des décisions collectives.

🔑 Étapes :

1. Création

- **Uniquement le fondateur** peut créer ou modifier la salle globale.
- Cette salle est de type GLOBAL et liée au **Group "Tous les membres"**.

2. Accès

- Tous les utilisateurs (ROLE_SUPER_ADMIN, ROLE_ADMIN, ROLE_USER) y ont accès.
- Pas de restriction de domaine ou de groupe.

3. Envoi de messages

- Tous les membres peuvent envoyer un message.
- Les messages sont **instantanés** via WebSocket ou Mercure.

4. Votes

- Le fondateur ou un ministre peut initier un vote dans cette salle.
- Tous les membres peuvent voter, mais seuls les résultats validés par le fondateur sont considérés comme définitifs.

◆ 2. Réunion Privée (Par groupe)

Objectif : Réunir les membres d'un groupe spécifique pour une organisation interne.

🔑 Étapes :

1. Création

- **Uniquement un ministre** peut créer une salle privée.
- Chaque salle est liée à un **Group** (ex. "Équipe de développement frontend").

2. Accès

- Seuls les membres affectés à ce groupe peuvent voir et entrer dans la salle.
- Un ministre **ne peut pas** créer ou envoyer de messages dans un groupe qu'il n'a pas créé.

3. Envoi de messages

- Seuls les membres du groupe peuvent envoyer et lire les messages.
- Les messages restent privés et chiffrés.

4. Votes internes

- Le ministre du groupe peut lancer un vote.
- Les résultats sont visibles uniquement par les membres du groupe et le fondateur.

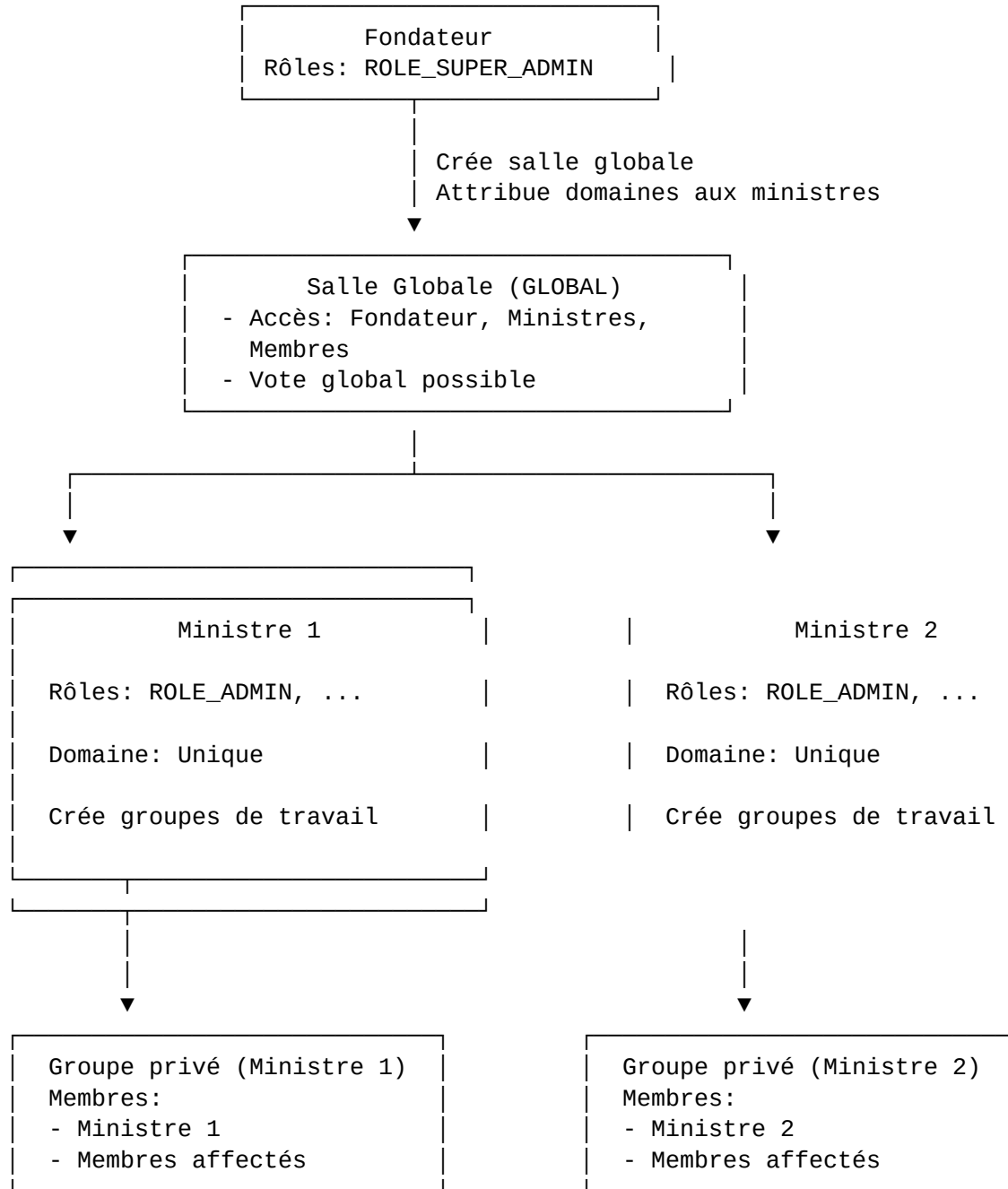
◆ 3. Communication Inter-rôles

- **Fondateur → Ministres** : Sans restriction.
- **Ministres → Fondateur** : Sans restriction.
- **Ministre → Ministre** : Possible seulement s'ils partagent un groupe commun.
- **Ministre → Membre** : Seulement si le membre est dans un de ses groupes.
- **Membre → Membre** : Seulement si les deux sont dans le même groupe.

◆ 4. Architecture technique

- **Backend** : Symfony + API Platform
- **Temps réel** : Symfony Mercure (ou WebSocket si besoin de chat très interactif)

- **Stockage :**
 - Messages → MongoDB (NoSQL pour performance)
 - Groupes, Salles, Votes → MySQL/PostgreSQL (relationnel)
 - **Sécurité :**
 - Accès contrôlé par **JWT + Rôles + Groupes**
 - Filtrage automatique des messages selon les permissions
-



Légende communication:

Fondateur	→	Ministres	:	Sans restriction
Ministres	→	Fondateur	:	Sans restriction
Ministres	↔	Ministres	:	Seulement si groupe commun
Ministres	→	Membres	:	Seulement si membre du groupe
Membres	↔	Membres	:	Seulement si groupe commun

Explication

- **Salle globale** = point de rencontre de toute l'organisation (décisions générales).
 - **Groupes privés** = zones restreintes à un ministre et ses membres.
 - **Flux de communication** strictement contrôlés par **rôles** et **appartenance au groupe**.
 - **Stockage** :
 - MySQL/PostgreSQL → gestion des groupes, salles et rôles.
 - MongoDB → messages en temps réel.
 - **Technos temps réel** : Symfony Mercure (pub/sub sécurisé).
-

Crédits et Contact

Document conçu par :

Architecte Web & Logiciel : Franck AGBOKOUDJO

Email : internationaleswebservices@gmail.com

Téléphone / WhatsApp : +229 01 67 25 18 86

GitHub : <https://www.github.com/Agbokoudjo/>

Page Facebook : *Web Apps & Services*

LinkedIn : <https://www.linkedin.com/in/internationales-web-apps-services-120520193/>
