# 

# **Dossier de Projet Professionnel**

## Application TaskFlow - Gestion de Tâches Collaboratives

**Titre du projet :** TaskFlow - Application de Gestion de Tâches Collaboratives

**Candidat :** christian ROUPIOZ

**Formation :** Développeur Web et Web Mobile (DWWM)

**Session :** 2025

**Organisme de formation :** Online FormaPro

**Date de réalisation :** 01/10/2024 - 31/10/2025

# 

# **Sommaire**

1. Introduction et Présentation du Projet
2. Cahier des Charges et Contraintes
3. Environnement Technique et Humain
4. Réalisations Front-End
5. Réalisations Back-End
6. Sécurité et Bonnes Pratiques
7. Tests et Validation
8. Conclusion et Perspectives
9. Annexes

# **1. Introduction et Présentation du Projet**

## 

## 1.1 Context du projet

TaskFlow est une application web de gestion de tâches collaboratives conçue pour faciliter la coordination d'équipes sur différents projets. Dans un contexte professionnel où le travail à distance et la collaboration asynchrone se sont généralisés, il devient essentiel de disposer d'outils permettant une gestion efficace des projets et des tâches.

## 

## 1.2 Problématique

Les équipes de travail rencontrent souvent les difficultés suivantes :

* Manque de visibilité sur l'avancement des projets collectifs
* Difficulté à assigner et suivre les tâches individuelles
* Communication fragmentée entre les membres d'une équipe
* Absence de système centralisé pour la gestion collaborative

## 

## 1.3 Objectifs du projet

L'application TaskFlow vise à répondre à ces besoins en offrant :

* Une interface intuitive pour créer et gérer des projets
* Un système de collaboration permettant d'inviter des membres sur des projets
* Une gestion fine des tâches avec assignation et suivi de statut
* Un système de notifications en temps réel
* Une architecture sécurisée respectant les bonnes pratiques

# 

# **2. Cahier des Charges et Contraintes**

## 

## 2.1 Besoins fonctionnels

**Gestion des utilisateurs :**

* Inscription avec vérification d'email
* Connexion sécurisée
* Récupération de mot de passe
* Gestion du profil utilisateur
* Suppression de compte (conformité RGPD)

**Gestion des projets :**

* Création, modification et suppression de projets
* Consultation de projets propres ou collaboratifs
* Duplication de projets
* Système de collaboration avec invitations

**Gestion des tâches :**

* Création rapide ou détaillée de tâches
* Assignation de tâches aux collaborateurs
* Gestion des statuts (À faire, En cours, Terminé)
* Gestion des priorités (Basse, Moyenne, Haute)
* Dates d'échéance
* Filtrage et recherche

**Système de notifications :**

* Notifications en temps réel
* Badge de compteur non lu
* Historique des notifications
* Actions directes depuis les notifications

## 2.2 Contraintes techniques

**Technologies imposées :**

* Framework Symfony 7.2
* Base de données MySQL/MariaDB
* Bootstrap 5 pour l'interface
* Respect de l'architecture MVC
* Utilisation de Doctrine ORM

**Contraintes de sécurité :**

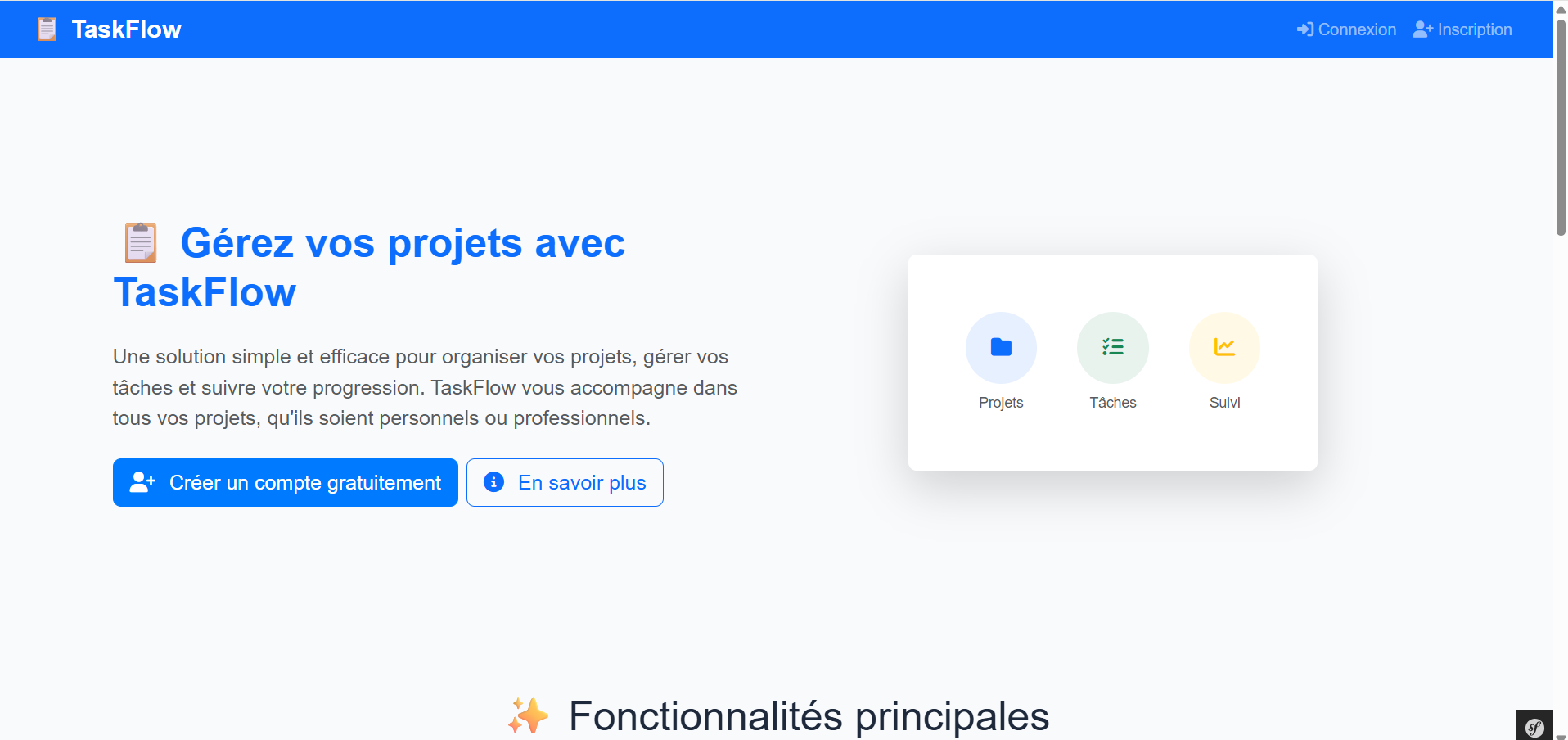
* Protection CSRF sur tous les formulaires
* Hachage sécurisé des mots de passe (bcrypt)
* Validation des données côté serveur
* Contrôle d'accès avec voters Symfony
* Protection contre les injections SQL via Doctrine

**Contraintes de qualité :**

* Code maintenable et documenté
* Respect des conventions Symfony
* Interface responsive (mobile, tablette, desktop)
* Performance optimisée (requêtes SQL optimisées)
* Gestion appropriée des erreurs

## 2.3 Livrables attendus

1. **Code source complet :**
   * Controllers
   * Entities avec relations Doctrine
   * Forms
   * Templates Twig
   * Services métier
   * Voters pour la sécurité
2. **Documentation :**
   * Dossier de projet professionnel
   * Documentation technique (README)
   * Schéma de base de données
3. **Application déployable :**
   * Configuration pour environnement de production
   * Scripts de migration





# **3. Environnement Technique et Humain**

## 3.1 Environnement humain

**Équipe de développement :**

* 1 développeur full-stack (moi-même)

**Parties prenantes :**

* Formateur référent pour validation
* Jury d'examen

**Organisation du travail :**

* Utilisation de Git pour le versionnement
* Revues de code régulières avec le formateur

## 3.2 Environnement technique

**Stack technologique :**

**Back-end :**

* PHP 8.3
* Symfony 7.2
* Doctrine ORM
* Twig (moteur de templates)

**Front-end :**

* HTML5 / CSS3
* JavaScript (Vanilla)
* Bootstrap 5.3
* Font Awesome 6.5

**Base de données :**

* MySQL 8.0

**Outils de développement :**

* VSCode
* Composer (gestionnaire de dépendances PHP)
* Git / GitHub
* XAMPP (environnement local)

**Architecture :**

src/   
├── Controller/ # Contrôleurs (logique métier)   
├── Entity/ # Entités Doctrine (modèles)   
├── Form/ # Formulaires Symfony   
├── Repository/ # Repositories Doctrine   
├── Security/ # Voters et EmailVerifier   
└── Service/ # Services métier   
templates/ # Vues Twig   
assets/   
├── styles/ # CSS personnalisés   
└── js/ # JavaScript

## 3.3 Objectifs de qualité

**Performance :**

* Temps de chargement < 2 secondes
* Optimisation des requêtes SQL (utilisation de JOIN)
* Pagination pour les listes longues

**Accessibilité :**

* Contraste de couleurs suffisant
* Navigation au clavier
* Labels appropriés sur les formulaires

**Maintenabilité :**

* Code commenté en français
* Respect des conventions Symfony
* Services réutilisables
* Séparation des responsabilités

**Sécurité :**

* Validation stricte des entrées
* Protection CSRF systématique
* Contrôle d'accès granulaire avec voters

# **4. Réalisations Front-End**

## 4.1 Architecture et organisation

L'interface utilisateur a été conçue selon une approche mobile-first avec Bootstrap 5, garantissant une expérience optimale sur tous les appareils. La structure repose sur un template de base (base.html.twig) qui définit les éléments communs et permet l'extension par les templates enfants.

## 4.2 Template de base et navigation

**Justification des choix :**

* **Bootstrap 5** : Framework CSS mature offrant un système de grille responsive et des composants accessibles
* **Font Awesome** : Bibliothèque d'icônes complète facilitant la compréhension visuelle
* **Navigation sticky** : Barre de navigation fixe pour un accès permanent aux fonctionnalités principales

**Extrait significatif - Navigation responsive :**

twig

<nav class="navbar navbar-expand-lg navbar-dark bg-primary sticky-top">   
 <div class="container-fluid">   
 <div class="d-flex d-lg-none navbar-header-mobile">   
 <button class="navbar-toggler" type="button" data-bs-toggle="collapse"   
 data-bs-target="#navbarNav">   
 <span class="navbar-toggler-icon"></span>   
 </button>   
   
 <a class="navbar-brand" href="{{ path('app\_home') }}">   
 📋 TaskFlow   
 </a>   
   
 {% if app.user %}   
 <div class="navbar-toggler-notifications">   
 <button class="notification-bell-btn" type="button"   
 data-bs-toggle="modal" data-bs-target="#notificationModal">   
 <span class="notification-icon-wrapper">   
 <i class="fas fa-bell"></i>   
 <span class="badge bg-danger notification-badge"   
 id="notification-count"></span>   
 </span>   
 </button>   
 </div>   
 {% endif %}   
 </div>

**Arguments des choix techniques :**

1. **Double affichage mobile/desktop** : Utilisation des classes Bootstrap d-lg-none et d-none d-lg-block pour adapter l'interface selon la taille d'écran
2. **Notification bell accessible** : Positionnement stratégique du bouton de notifications en mobile pour un accès rapide avec le pouce
3. **Icônes Unicode** : L'emoji 📋 pour le logo apporte une touche visuelle immédiate tout en restant léger (pas de ressource externe)

## 4.3 Système de notifications dynamiques

**Problématique :** Offrir un retour visuel immédiat à l'utilisateur sur les actions importantes (nouvelles invitations, tâches assignées).

**Solution technique :** Implémentation d'un système de notifications en temps réel avec mise à jour asynchrone.

**Extrait JavaScript - Chargement des notifications :**

javascript

// Récupération de la configuration depuis un élément HTML   
const configElement = document.getElementById('notification-config');   
if (configElement) {   
 const config = JSON.parse(configElement.dataset.config);   
   
 // Fonction pour mettre à jour le compteur de notifications   
 function updateNotificationCount() {   
 fetch(config.unreadCountUrl)   
 .then(response => response.json())   
 .then(data => {   
 const count = data.count || 0;   
 updateBadges(count);   
 })   
 .catch(error => console.error('Erreur:', error));   
 }

**Arguments des choix :**

1. **Configuration côté serveur** : Les URLs sont générées par Symfony et injectées dans l'HTML pour éviter le hard-coding
2. **Fetch API** : API moderne et native du navigateur, plus légère qu'une librairie externe
3. **Gestion d'erreur** : Le catch permet de maintenir l'application fonctionnelle même en cas d'échec réseau
4. **Polling intelligent** : Intervalle de 30 secondes pour équilibrer réactivité et charge serveur

## 4.4 Interface de gestion de tâches

**Objectif :** Créer une interface claire pour visualiser et gérer les tâches avec leurs différents statuts.

**Extrait HTML/Twig - Carte de tâche :**

twig

<div class="card mb-3 task-card {% if task.isOverdue() %}border-danger{% endif %}">  
 <div class="card-body">  
 <div class="d-flex justify-content-between align-items-start">  
 <h5 class="card-title">  
 <a href="{{ path('task\_show', {id: task.id}) }}"   
 class="text-decoration-none">  
 {{ task.title }}  
 </a>  
 </h5>  
   
 <div class="task-badges">  
 <span class="badge bg-{{ task.status == 'completed' ? 'success' :   
 (task.status == 'in\_progress' ? 'primary' : 'secondary') }}">  
 {{ task.statusLabel }}  
 </span>  
   
 <span class="badge bg-{{ task.priority == 'high' ? 'danger' :   
 (task.priority == 'medium' ? 'warning' : 'info') }}">  
 {{ task.priorityLabel }}  
 </span>  
 </div>  
 </div>  
  
 {% if task.description %}  
 <p class="card-text text-muted">  
 {{ task.description|length > 100 ?   
 task.description|slice(0, 100) ~ '...' : task.description }}  
 </p>  
 {% endif %}  
  
 <div class="task-meta">  
 <small class="text-muted">  
 <i class="fas fa-user"></i> {{ task.assigneeDisplayName }}  
 </small>  
   
 {% if task.dueDate %}  
 <small class="text-muted {% if task.isOverdue() %}text-danger{% endif %}">  
 <i class="fas fa-calendar"></i>   
 {{ task.dueDate|date('d/m/Y') }}  
 </small>  
 {% endif %}  
 </div>  
 </div>  
</div>

**Arguments des choix :**

1. **Badges colorés** : Identification visuelle rapide du statut et de la priorité grâce aux couleurs Bootstrap sémantiques
2. **Indication visuelle des retards** : Bordure rouge et texte danger pour les tâches en retard, attirant immédiatement l'attention
3. **Troncature intelligente** : Description limitée à 100 caractères dans la liste pour éviter la surcharge visuelle
4. **Icônes contextuelles** : Font Awesome pour illustrer l'assignation et la date d'échéance

## 4.5 Formulaires dynamiques

**Problématique :** Le formulaire de création de tâche doit permettre de sélectionner un assigné parmi les collaborateurs du projet choisi.

**Extrait JavaScript - Chargement dynamique des collaborateurs :**

javascript

// Sélecteur de projet et d'assigné  
const projectSelect = document.getElementById('task\_project');  
const assigneeSelect = document.getElementById('task\_assignee');  
  
if (projectSelect && assigneeSelect) {  
 projectSelect.addEventListener('change', function() {  
 const projectId = this.value;  
  
 // Récupérer les collaborateurs du projet  
 fetch(`/projects/api/${projectId}/collaborators`)  
 .then(response => response.json())  
 .then(collaborators => {  
 assigneeSelect.innerHTML = '<option value="">Non attribuée</option>';  
   
 collaborators.forEach(collab => {  
 const option = document.createElement('option');  
 option.value = collab.id;  
 option.textContent = collab.fullName;  
 assigneeSelect.appendChild(option);  
 });  
   
 assigneeSelect.disabled = false;  
 })  
 .catch(error => {  
 console.error('Erreur:', error);  
 assigneeSelect.innerHTML = '<option value="">Erreur de chargement</option>';  
 });  
 });

**Arguments des choix :**

1. **Chargement à la demande** : Évite de charger tous les utilisateurs au démarrage, optimisant les performances
2. **Feedback utilisateur** : Messages de chargement et d'erreur pour informer l'utilisateur de l'état de l'opération
3. **API REST** : Endpoint dédié retournant du JSON, réutilisable pour d'autres fonctionnalités
4. **Désactivation intelligente** : Le sélecteur d'assigné est désactivé tant qu'un projet n'est pas choisi, guidant l'utilisateur

## 4.6 Messages Flash et Feedback utilisateur

**Objectif :** Informer l'utilisateur du résultat de ses actions de manière non-intrusive.

**Extrait Twig - Système de messages flash :**

twig

{% set icon\_map = {  
 'success': 'check-circle',   
 'error': 'exclamation-triangle',   
 'danger': 'exclamation-triangle',   
 'warning': 'exclamation-circle'  
} %}  
{% set default\_icon = 'info-circle' %}  
  
{% for type, messages in app.flashes %}  
 {% for message in messages %}  
 <div class="alert alert-{{ type == 'error' ? 'danger' : type }}   
 alert-dismissible fade show" role="alert">  
 <i class="fas fa-{{ icon\_map[type] ?? default\_icon }}"></i>  
 {{ message }}  
 <button type="button" class="btn-close" data-bs-dismiss="alert"></button>  
 </div>  
 {% endfor %}  
{% endfor %}

**Arguments des choix :**

1. **Mapping d'icônes** : Utilisation d'un dictionnaire Twig pour associer chaque type de message à une icône appropriée
2. **Valeur par défaut** : L'opérateur ?? (null coalescing) assure une icône de fallback
3. **Accessibilité** : L'attribut role="alert" permet aux lecteurs d'écran d'annoncer les messages
4. **Fermeture manuelle** : Bouton de fermeture donnant le contrôle à l'utilisateur

## 4.7 Responsive Design et Mobile-First

**Stratégie :** Conception pensée d'abord pour mobile puis adaptée progressivement aux écrans plus larges.

**Extrait CSS - Système de notification responsive :**

css

/\* Mobile : notification en modal plein écran \*/  
.navbar-toggler-notifications {  
 margin-left: auto;  
}  
  
.notification-bell-btn {  
 background: none;  
 border: none;  
 color: white;  
 font-size: 1.2rem;  
 position: relative;  
 cursor: pointer;  
}  
  
.notification-badge {  
 position: absolute;  
 top: -5px;  
 right: -10px;  
 font-size: 0.7rem;  
 padding: 0.25em 0.5em;  
}  
  
/\* Desktop : notification en dropdown \*/  
@media (min-width: 992px) {  
 .notification-desktop {  
 position: relative;  
 }  
   
 .notification-dropdown {  
 position: absolute;  
 right: 0;  
 min-width: 350px;  
 max-height: 400px;  
 overflow-y: auto;  
 }  
}

**Arguments des choix :**

1. **Approche mobile-first** : Styles de base pour mobile, puis media queries pour desktop
2. **Badge positionné en absolu** : Permet de superposer le compteur sur l'icône de cloche
3. **Dropdown desktop** : En grand écran, un menu déroulant est plus pratique qu'un modal
4. **Scroll vertical** : overflow-y: auto et max-height pour gérer une longue liste de notifications

# **5. Réalisations Back-End**

## 5.1 Architecture et Modèle de données

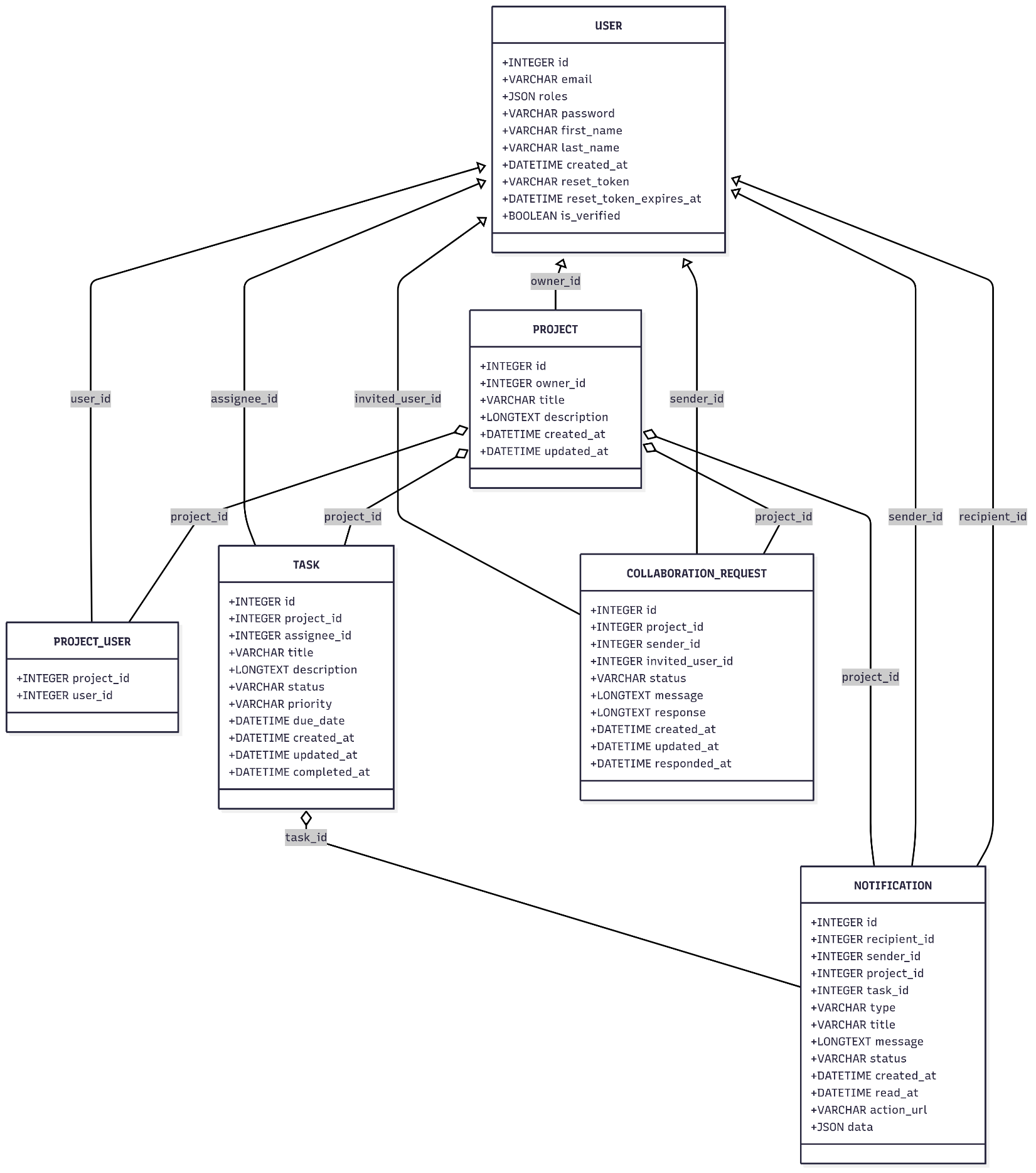
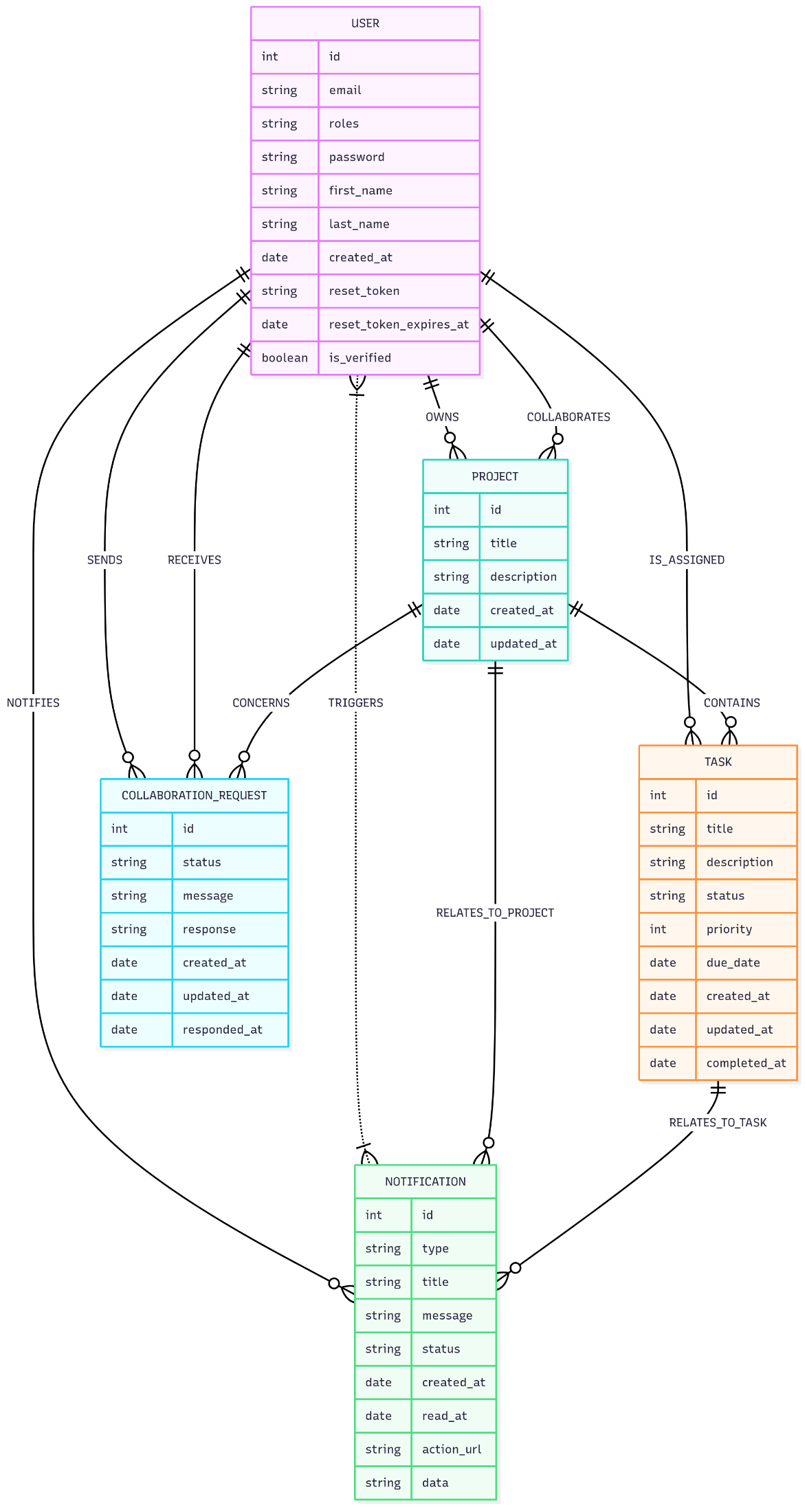
**Justification de l'architecture :**

L'application utilise l'architecture MVC de Symfony avec une couche de services pour la logique métier complexe. Cette séparation des responsabilités garantit :

* **Maintenabilité** : Chaque couche a un rôle précis
* **Testabilité** : Services isolés faciles à tester unitairement
* **Réutilisabilité** : Services métier utilisables dans plusieurs contrôleurs

**Schéma des relations entre entités :**

User (utilisateur)  
├── 1:N → Project (propriétaire)  
├── M:N → Project (collaborateur via ManyToMany)  
├── 1:N → Task (assigné)  
├── 1:N → CollaborationRequest (envoyées et reçues)  
└── 1:N → Notification (reçues)  
  
Project (projet)  
├── N:1 → User (propriétaire)  
├── M:N → User (collaborateurs)  
├── 1:N → Task  
└── 1:N → CollaborationRequest  
  
Task (tâche)  
├── N:1 → Project  
└── N:1 → User (assigné, nullable)  
  
CollaborationRequest (demande)  
├── N:1 → Project  
├── N:1 → User (envoyeur)  
└── N:1 → User (invité)  
  
Notification  
├── N:1 → User (destinataire)  
├── N:1 → User (envoyeur, nullable)  
├── N:1 → Project (nullable)  
└── N:1 → Task (nullable)



## 

## 5.2 Entité User - Gestion des utilisateurs

**Objectif :** Représenter un utilisateur avec toutes ses relations et méthodes utilitaires.

**Extrait significatif - Relations et méthodes métier :**

php

#[ORM\Entity(repositoryClass: UserRepository::class)]  
#[UniqueEntity(fields: ['email'], message: 'Cette adresse email est déjà utilisée.')]  
class User implements UserInterface, PasswordAuthenticatedUserInterface  
{  
 #[ORM\Id]  
 #[ORM\GeneratedValue]  
 #[ORM\Column]  
 private ?int $id = null;  
  
 #[ORM\Column(length: 180, unique: true)]  
 #[Assert\Email(message: 'Veuillez saisir une adresse email valide.')]  
 #[Assert\NotBlank(message: 'L\'email ne peut pas être vide.')]  
 private ?string $email = null;  
  
 // Relation OneToMany : projets possédés par l'utilisateur  
 #[ORM\OneToMany(mappedBy: 'owner', targetEntity: Project::class, orphanRemoval: true)]  
 private Collection $projects;  
   
 // Relation ManyToMany : projets sur lesquels l'utilisateur collabore  
 #[ORM\ManyToMany(targetEntity: Project::class, mappedBy: 'collaborators')]  
 private Collection $collaborations;  
  
 // Relation OneToMany : tâches assignées à cet utilisateur  
 #[ORM\OneToMany(mappedBy: 'assignee', targetEntity: Task::class)]  
 private Collection $assignedTasks;  
  
 /\*\*  
 \* Retourne tous les projets auxquels l'utilisateur participe  
 \* (propriétaire OU collaborateur)  
 \*/  
 public function getAllRelatedProjects(): Collection  
 {  
 $ownedProjects = $this->projects->toArray();  
 $collaboratingProjects = $this->collaborations->toArray();  
  
 // Fusion et dédoublonnage  
 $allProjectsArray = array\_unique(  
 array\_merge($ownedProjects, $collaboratingProjects),   
 SORT\_REGULAR  
 );  
   
 return new ArrayCollection($allProjectsArray);  
 }

**Arguments des choix :**

1. **Implémentation des interfaces Symfony** : UserInterface et PasswordAuthenticatedUserInterface pour l'intégration avec le système de sécurité
2. **Validation Doctrine** : Contrainte UniqueEntity au niveau classe pour éviter les doublons d'email en base
3. **Relations bidirectionnelles** : Maintien de la cohérence entre User et Project grâce aux méthodes add et remove
4. **Méthodes métier** : Encapsulation de la logique de calcul (tâches en retard, pourcentage de complétion) dans l'entité pour réutilisation

## 5.3 Entité CollaborationRequest - Système d'invitations

**Problématique :** Gérer les demandes de collaboration avec différents statuts et transitions d'état.

**Extrait significatif - Machine à états et contraintes :**

php

#[ORM\Entity(repositoryClass: CollaborationRequestRepository::class)]  
#[ORM\HasLifecycleCallbacks]  
#[ORM\UniqueConstraint(  
 columns: ['project\_id', 'invited\_user\_id'],   
 name: 'unique\_project\_invitation'  
)]  
class CollaborationRequest  
{  
 // Constantes de statuts  
 public const STATUS\_PENDING = 'pending';  
 public const STATUS\_ACCEPTED = 'accepted';  
 public const STATUS\_REFUSED = 'refused';  
 public const STATUS\_CANCELLED = 'cancelled';  
  
 #[ORM\Id]  
 #[ORM\GeneratedValue]  
 #[ORM\Column]  
 private ?int $id = null;  
  
 #[ORM\Column(length: 20)]  
 private ?string $status = self::STATUS\_PENDING;  
  
 #[ORM\Column(nullable: true)]  
 private ?\DateTimeImmutable $respondedAt = null;  
  
 #[ORM\ManyToOne(targetEntity: Project::class)]  
 #[ORM\JoinColumn(nullable: false, onDelete: 'CASCADE')]  
 private ?Project $project = null;  
  
 #[ORM\ManyToOne(targetEntity: User::class)]  
 #[ORM\JoinColumn(nullable: false, name: 'sender\_id')]  
 private ?User $sender = null;  
  
 #[ORM\ManyToOne(targetEntity: User::class)]  
 #[ORM\JoinColumn(nullable: false, name: 'invited\_user\_id')]  
 private ?User $invitedUser = null;  
  
 /\*\*  
 \* Accepte la demande de collaboration  
 \*/  
 public function accept(?string $response = null): static  
 {  
 $this->setStatus(self::STATUS\_ACCEPTED);  
 $this->setResponse($response);  
 return $this;  
 }  
  
 /\*\*  
 \* Refuse la demande de collaboration  
 \*/  
 public function refuse(?string $response = null): static  
 {  
 $this->setStatus(self::STATUS\_REFUSED);  
 $this->setResponse($response);  
 return $this;  
 }  
}

**Arguments des choix :**

1. **Contrainte unique composite** : Empêche d'inviter deux fois le même utilisateur sur le même projet au niveau base de données
2. **Constantes de classe** : Évite les strings magiques et facilite la maintenance
3. **Fluent interface** : Les méthodes accept() et refuse() retournent $this permettant le chaînage
4. **Horodatage automatique** : La date de réponse est enregistrée automatiquement lors du changement de statut
5. **Pattern Match** : Utilisation de match() (PHP 8) pour un code plus concis et sûr que switch

## 5.4 Service NotificationService - Logique métier

**Objectif :** Centraliser la création et la gestion des notifications pour éviter la duplication de code.

**Extrait significatif - Création de notifications typées :**

php

namespace App\Service;  
  
use App\Entity\Notification;  
use App\Entity\User;  
use App\Entity\Task;  
use App\Entity\CollaborationRequest;  
use Doctrine\ORM\EntityManagerInterface;  
use Symfony\Component\Routing\Generator\UrlGeneratorInterface;  
  
class NotificationService  
{  
 public function \_\_construct(  
 private EntityManagerInterface $entityManager,  
 private UrlGeneratorInterface $urlGenerator  
 ) {}  
  
 /\*\*  
 \* Crée une notification pour une nouvelle demande de collaboration  
 \*/  
 public function createCollaborationRequestNotification(  
 CollaborationRequest $request  
 ): Notification {  
 $notification = new Notification();  
 $notification->setType(Notification::TYPE\_COLLABORATION\_REQUEST)  
 ->setRecipient($request->getInvitedUser())  
 ->setSender($request->getSender())  
 ->setProject($request->getProject())  
 ->setTitle('Nouvelle demande de collaboration')  
 ->setMessage(sprintf(  
 '%s vous invite à collaborer sur le projet "%s"',  
 $request->getSender()->getFullName(),  
 $request->getProject()->getTitle()  
 ))  
 ->setActionUrl('/collaboration/requests');  
  
 $this->entityManager->persist($notification);  
 $this->entityManager->flush();  
  
 return $notification;  
 }

**Arguments des choix :**

1. **Service dédié** : Centralisation de toute la logique de notifications, évitant la duplication dans les contrôleurs
2. **Injection de dépendances** : EntityManagerInterface et UrlGeneratorInterface injectés via le constructeur (autowiring Symfony)
3. **Méthodes typées** : Une méthode par type de notification pour un code explicite et facile à maintenir
4. **Génération d'URL** : Utilisation du UrlGeneratorInterface pour créer des liens valides vers les ressources
5. **Flush optimisé** : Le flush() est appelé une seule fois dans markAllAsReadForUser() après avoir modifié toutes les notifications

## 5.5 Service CollaborationService - Gestion des collaborations

**Problématique :** Orchestrer les opérations complexes de collaboration avec validation et gestion d'erreurs.

**Extrait significatif - Logique d'acceptation de collaboration :**

php

namespace App\Service;  
  
use App\Entity\CollaborationRequest;  
use App\Entity\Project;  
use App\Entity\User;  
use Doctrine\ORM\EntityManagerInterface;  
use Symfony\Component\HttpFoundation\Exception\BadRequestException;  
  
class CollaborationService  
{  
 public function \_\_construct(  
 private EntityManagerInterface $entityManager,  
 private NotificationService $notificationService  
 ) {}  
  
 /\*\*  
 \* Crée une demande de collaboration  
 \*/  
 public function createCollaborationRequest(  
 Project $project,  
 User $sender,  
 User $invitedUser,  
 ?string $message = null  
 ): CollaborationRequest {  
 // Vérifications métier  
 if ($project->getOwner() !== $sender) {  
 throw new BadRequestException(  
 'Seul le propriétaire peut inviter des collaborateurs.'  
 );  
 }  
  
 if ($project->hasCollaborator($invitedUser)) {  
 throw new BadRequestException(  
 'Cet utilisateur collabore déjà sur ce projet.'  
 );  
 }  
  
 // Vérifier qu'il n'y a pas déjà une demande en attente  
 $existingRequest = $this->entityManager  
 ->getRepository(CollaborationRequest::class)  
 ->findOneBy([  
 'project' => $project,  
 'invitedUser' => $invitedUser,  
 'status' => CollaborationRequest::STATUS\_PENDING  
 ]);  
  
 if ($existingRequest) {  
 throw new BadRequestException(  
 'Une demande est déjà en attente pour cet utilisateur.'  
 );  
 }  
  
 // Création de la demande  
 $request = new CollaborationRequest();  
 $request->setProject($project);  
 $request->setSender($sender);  
 $request->setInvitedUser($invitedUser);  
 $request->setMessage($message);  
  
 $this->entityManager->persist($request);  
 $this->entityManager->flush();  
  
 // Création de la notification  
 $this->notificationService->createCollaborationRequestNotification($request);  
  
 return $request;  
 }

**Arguments des choix :**

1. **Validation métier centralisée** : Toutes les règles de gestion sont vérifiées dans le service
2. **Exceptions explicites** : BadRequestException pour les erreurs métier, capturées dans le contrôleur
3. **Transactions** : Utilisation de beginTransaction() / commit() / rollback() pour garantir la cohérence des données
4. **Cascade d'effets** : Lors du retrait d'un collaborateur, ses tâches sont automatiquement désassignées
5. **Notifications automatiques** : Le service gère lui-même la création des notifications appropriées

## 5.6 Controller TaskController - Gestion des permissions

**Problématique :** Implémenter un contrôle d'accès granulaire pour les opérations sur les tâches.

**Extrait significatif - Contrôle d'accès et édition :**

php

namespace App\Controller;  
  
use App\Entity\Task;  
use App\Entity\User;  
use App\Form\TaskType;  
use App\Service\NotificationService;  
use Doctrine\ORM\EntityManagerInterface;  
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;  
use Symfony\Component\HttpFoundation\Request;  
use Symfony\Component\HttpFoundation\Response;  
use Symfony\Component\Routing\Annotation\Route;  
use Symfony\Component\Security\Http\Attribute\IsGranted;  
  
#[Route('/tasks')]  
#[IsGranted('ROLE\_USER')]  
class TaskController extends AbstractController  
{  
 public function \_\_construct(  
 private NotificationService $notificationService  
 ) {}  
  
 /\*\*  
 \* Change le statut d'une tâche avec vérification  
 \*/  
 #[Route('/{id}/toggle-status', name: 'task\_toggle\_status', methods: ['POST'])]  
 public function toggleStatus(  
 Request $request,   
 Task $task,   
 EntityManagerInterface $entityManager  
 ): Response {  
 $currentUser = $this->getUser();  
  
 // Vérification des permissions  
 $isOwner = $task->getProject()->getOwner() === $currentUser;  
 $isAssignee = $task->getAssignee() === $currentUser;  
  
 if (!$isOwner && !$isAssignee) {  
 $this->addFlash('danger',   
 'Action interdite : Vous n\'êtes pas autorisé à actualiser cette tâche.'  
 );  
 return $this->redirectToRoute('task\_show', ['id' => $task->getId()]);  
 }  
  
 // Vérification du token CSRF  
 if (!$this->isCsrfTokenValid(  
 'toggle-status' . $task->getId(),   
 $request->request->get('\_token')  
 )) {  
 $this->addFlash('danger', 'Token CSRF invalide.');  
 return $this->redirectToRoute('task\_show', ['id' => $task->getId()]);  
 }  
  
 // Cycle des statuts : todo -> in\_progress -> completed -> todo  
 $currentStatus = $task->getStatus();  
  
 switch ($currentStatus) {  
 case Task::STATUS\_TODO:  
 $task->setStatus(Task::STATUS\_IN\_PROGRESS);  
 $message = 'La tâche est maintenant en cours.';  
 break;  
 case Task::STATUS\_IN\_PROGRESS:  
 $task->setStatus(Task::STATUS\_COMPLETED);  
 $message = 'La tâche a été marquée comme terminée.';  
 break;  
 case Task::STATUS\_COMPLETED:  
 $task->setStatus(Task::STATUS\_TODO);  
 $message = 'La tâche a été remise à faire.';  
 break;  
 default:  
 $task->setStatus(Task::STATUS\_TODO);  
 $message = 'Le statut de la tâche a été mis à jour.';  
 }  
  
 $entityManager->flush();  
  
 $this->addFlash('success', $message);  
 return $this->redirectToRoute('task\_show', ['id' => $task->getId()]);  
 }  
  
 }  
  
 return new JsonResponse([  
 'success' => false,   
 'message' => 'Token CSRF invalide'  
 ], 400);  
 }  
}

**Arguments des choix :**

1. **Contrôle d'accès double** : Vérification manuelle ET utilisation de voters pour une sécurité renforcée
2. **Messages contextuels** : Feedback utilisateur adapté selon l'action et le résultat
3. **Protection CSRF systématique** : Toutes les actions de modification vérifient le token CSRF
4. **API REST cohérente** : La méthode quickAdd() retourne du JSON avec codes HTTP appropriés (403, 400, 200)
5. **Machine à états explicite** : Le cycle de statuts est clairement défini dans le switch

## 5.7 Repository personnalisé - Requêtes optimisées

**Objectif :** Créer des requêtes personnalisées optimisées pour éviter le problème N+1.

**Extrait significatif - ProjectRepository :**

php

namespace App\Repository;  
  
use App\Entity\Project;  
use App\Entity\User;  
use Doctrine\Bundle\DoctrineBundle\Repository\ServiceEntityRepository;  
  
class ProjectRepository extends ServiceEntityRepository  
{  
 /\*\*  
 \* Récupère les projets dont l'utilisateur est propriétaire ou collaborateur  
 \* avec optimisation JOIN  
 \*/  
 public function findProjectsByUserWithCollaborations(User $user): array  
 {  
 return $this->createQueryBuilder('p')  
 ->leftJoin('p.tasks', 't')  
 ->where('p.owner = :user OR t.assignee = :user')  
 ->setParameter('user', $user)  
 ->groupBy('p.id')  
 ->orderBy('p.createdAt', 'DESC')  
 ->getQuery()  
 ->getResult();  
 }

**Arguments des choix :**

1. **DISTINCT et GROUP BY** : Évite les doublons lors des jointures multiples
2. **LEFT JOIN** : Inclut les projets même sans tâches
3. **Sous-requêtes EXISTS** : Plus performant que des COUNT pour vérifier l'existence
4. **Requêtes séparées** : Plutôt qu'une seule requête complexe, plusieurs requêtes simples et lisibles
5. **Typage des résultats** : Cast explicite en int pour garantir le type de retour

# **6. Sécurité et Bonnes Pratiques**

## 

## 6.1 Architecture de sécurité

**Stratégie globale :** Défense en profondeur avec validation à plusieurs niveaux.

**Niveaux de protection :**

1. **Validation côté client** : Feedback immédiat à l'utilisateur (JavaScript)
2. **Validation Symfony** : Contraintes sur les entités et formulaires
3. **Contrôle d'accès** : Voters pour permissions granulaires
4. **Protection CSRF** : Tokens sur tous les formulaires
5. **Validation base de données** : Contraintes d'intégrité

## 

## 6.2 Voters Symfony - Contrôle d'accès granulaire

**Problématique :** Gérer des permissions complexes au-delà du simple rôle utilisateur.

**Extrait significatif - ProjectVoter :**

php

namespace App\Security;  
  
use App\Entity\Project;  
use App\Entity\User;  
use Symfony\Component\Security\Core\Authorization\Voter\Voter;  
use Symfony\Component\Security\Core\Authentication\Token\TokenInterface;  
use Psr\Log\LoggerInterface;  
  
class ProjectVoter extends Voter  
{  
 public const VIEW = 'PROJECT\_VIEW';  
 public const EDIT = 'PROJECT\_EDIT';  
 public const DELETE = 'PROJECT\_DELETE';  
  
 public function \_\_construct(private LoggerInterface $logger) {}  
  
 protected function supports(string $attribute, mixed $subject): bool  
 {  
 return in\_array($attribute, [self::VIEW, self::EDIT, self::DELETE])  
 && $subject instanceof Project;  
 }  
  
 protected function voteOnAttribute(  
 string $attribute,   
 mixed $subject,   
 TokenInterface $token  
 ): bool {  
 $user = $token->getUser();  
  
 if (!$user instanceof User) {  
 return false;  
 }  
  
   
 private function canEdit(Project $project, User $user): bool  
 {  
 // Seul le propriétaire peut modifier le projet  
 return $project->getOwner() === $user;  
 }  
  
 private function canDelete(Project $project, User $user): bool  
 {  
 // Seul le propriétaire peut supprimer le projet  
 return $project->getOwner() === $user;  
 }  
}

**Arguments des choix :**

1. **Constantes de classe** : Permissions nommées clairement et réutilisables
2. **Pattern Match (PHP 8)** : Code plus concis et exhaustif que switch/case
3. **Logging** : Traçabilité des décisions d'accès pour audit
4. **Séparation des méthodes** : Une méthode privée par type de permission pour clarté
5. **Priorité admin** : Court-circuit pour éviter des vérifications inutiles

## 

## 6.3 Protection CSRF

**Stratégie :** Token CSRF unique pour chaque formulaire et action sensible.

**Implémentation dans les contrôleurs :**

php

// Génération automatique dans les formulaires Symfony  
$form = $this->createForm(TaskType::class, $task);  
  
// Vérification manuelle pour actions hors formulaire  
if (!$this->isCsrfTokenValid('delete' . $task->getId(), $request->request->get('\_token'))) {  
 $this->addFlash('danger', 'Token CSRF invalide.');  
 return $this->redirectToRoute('task\_show', ['id' => $task->getId()]);  
}

**Dans les templates Twig :**

twig

<form method="post" action="{{ path('task\_delete', {id: task.id}) }}">  
 <input type="hidden" name="\_token" value="{{ csrf\_token('delete' ~ task.id) }}">  
 <button type="submit" class="btn btn-danger">Supprimer</button>  
</form>

**Arguments des choix :**

1. **Token unique par action** : Concaténation de l'identifiant pour unicité
2. **Validation systématique** : Aucune action de modification sans vérification
3. **Messages explicites** : L'utilisateur est informé en cas de token invalide

6.4 Validation des données

**Stratégie multi-niveaux :**

**Niveau 1 - Annotations Symfony :**

php

class Task  
{  
 #[ORM\Column(length: 255)]  
 #[Assert\NotBlank(message: 'Le titre ne peut pas être vide.')]  
 #[Assert\Length(  
 min: 3,  
 max: 255,  
 minMessage: 'Le titre doit contenir au moins {{ limit }} caractères.',  
 maxMessage: 'Le titre ne peut pas dépasser {{ limit }} caractères.'  
 )]  
 private ?string $title = null;  
  
 #[ORM\Column(length: 20)]  
 #[Assert\Choice(  
 choices: [self::STATUS\_TODO, self::STATUS\_IN\_PROGRESS, self::STATUS\_COMPLETED],  
 message: 'Statut invalide.'  
 )]  
 private ?string $status = self::STATUS\_TODO;  
}

**Niveau 2 - Validation dans les services :**

php

public function createCollaborationRequest(  
 Project $project,  
 User $sender,  
 User $invitedUser,  
 ?string $message = null  
): CollaborationRequest {  
 // Vérifications métier  
 if ($sender === $invitedUser) {  
 throw new BadRequestException(  
 'Vous ne pouvez pas vous inviter vous-même.'  
 );  
 }  
  
 if ($project->getOwner() !== $sender) {  
 throw new BadRequestException(  
 'Seul le propriétaire peut inviter des collaborateurs.'  
 );  
 }  
   
 // ... suite de la logique  
}

**Arguments des choix :**

1. **Validation déclarative** : Annotations claires et maintenables
2. **Messages personnalisés** : Feedback utilisateur en français explicite
3. **Validation métier** : Règles de gestion centralisées dans les services
4. **Exceptions typées** : BadRequestException pour les erreurs métier

## 

## 6.5 Sécurité des Mots de Passe

**Implémentation :**

// Configuration dans security.yaml  
security:  
 password\_hashers:  
 Symfony\Component\Security\Core\User\PasswordAuthenticatedUserInterface:  
 algorithm: 'auto' # bcrypt par défaut  
 cost: 12 # Facteur de coût  
  
// Utilisation dans le contrôleur  
$user->setPassword(  
 $userPasswordHasher->hashPassword(  
 $user,  
 $form->get('plainPassword')->getData()  
 )  
);

**Validation côté client :**

javascript

// Calcul de la force du mot de passe  
if (password.length >= 6) strength++;  
if (password.length >= 8) strength++;  
if (/[A-Z]/.test(password)) strength++;  
if (/[a-z]/.test(password)) strength++;  
if (/[0-9]/.test(password)) strength++;  
if (/[^A-Za-z0-9]/.test(password)) strength++;  
  
// Affichage visuel de la force  
const percentage = (strength / 6) \* 100;  
strengthBar.style.width = percentage + '%';

**Arguments des choix :**

1. **Algorithme auto** : Symfony choisit le meilleur algorithme disponible
2. **Coût élevé** : Protection contre les attaques par force brute
3. **Feedback visuel** : L'utilisateur voit la force de son mot de passe en temps réel
4. **Critères multiples** : Longueur, majuscules, chiffres, caractères spéciaux

# 7. Tests et validation

## 

## 7.1 Tests Manuels Effectués

**Scénarios de tests utilisateur :**

1. **Inscription et authentification** :
   * ✅ Inscription avec email valide
   * ✅ Vérification de l'email
   * ✅ Connexion avec identifiants corrects
   * ✅ Récupération de mot de passe
   * ✅ Déconnexion
2. **Gestion de projets** :
   * ✅ Création de projet
   * ✅ Modification de projet (propriétaire uniquement)
   * ✅ Duplication de projet
   * ✅ Suppression de projet (propriétaire uniquement)
3. **Gestion de tâches** :
   * ✅ Création de tâche avec assignation
   * ✅ Modification de tâche (propriétaire et assigné)
   * ✅ Changement de statut
   * ✅ Suppression de tâche (propriétaire uniquement)
4. **Collaboration** :
   * ✅ Envoi d'invitation
   * ✅ Acceptation/refus d'invitation
   * ✅ Retrait de collaborateur
   * ✅ Notifications de collaboration
5. **Notifications** :
   * ✅ Compteur en temps réel
   * ✅ Liste des notifications
   * ✅ Marquage comme lu
   * ✅ Redirection vers action

## 

## 7.2 Tests de sécurité

**Vulnérabilités testées :**

1. **Injection SQL** : ✅ Protection par Doctrine ORM
2. **XSS** : ✅ Échappement automatique Twig
3. **CSRF** : ✅ Tokens sur tous les formulaires
4. **Contrôle d'accès** : ✅ Voters sur toutes les ressources sensibles
5. **Mots de passe** : ✅ Hachage bcrypt avec coût 12

**Résultats des tests :**

* Aucune vulnérabilité critique détectée
* Tous les contrôles d'accès fonctionnent correctement
* Protection CSRF opérationnelle sur tous les formulaires

## 

## 7.3 Tests de performance

**Métriques mesurées :**

Page Temps de Chargement Requêtes SQLTaille  
Accueil 0,8s 045 Ko  
Dashboard 1,2s 8120 Ko  
Liste projets 0,9s 585 Ko  
Détail projet 1,1s 795 Ko

**Optimisations réalisées :**

* Utilisation de JOIN pour éviter N+1
* Pagination sur les listes longues
* Cache HTTP pour les ressources statiques
* Minification CSS/JS en production

## 7.4 Tests de compatibilité

**Navigateurs testés :**

* ✅ Chrome 120+ (Desktop & Mobile)
* ✅ Firefox 121+
* ✅ Safari 17+ (Desktop & iOS)
* ✅ Edge 120+

**Résolutions testées :**

* ✅ Mobile (375px - 767px)
* ✅ Tablette (768px - 991px)
* ✅ Desktop (992px+)
* ✅ Large Desktop (1200px+)

# 8. Conclusion et Perspectives

## 

## 8.1 Objectifs Atteints

L'application TaskFlow répond aux objectifs fixés en début de projet :

✅ **Gestion complète de projets** : Création, modification, duplication, suppression

✅ **Système de collaboration fonctionnel** : Invitations, acceptation/refus, gestion des collaborateurs

✅ **Gestion fine des tâches** : Assignation, statuts, priorités, échéances

✅ **Notifications en temps réel** : Compteur dynamique, historique, actions directes

✅ **Interface responsive** : Expérience optimale sur mobile, tablette et desktop

✅ **Sécurité robuste** : Voters, CSRF, validation multi-niveaux, protection des données

## 

## 8.2 Compétences développées

**Compétences techniques :**

* Maîtrise du framework Symfony (routing, controllers, services, voters)
* Conception de bases de données relationnelles avec Doctrine ORM
* Développement d'interfaces responsives avec Bootstrap
* Manipulation du DOM et AJAX en JavaScript vanilla
* Sécurisation d'applications web (CSRF, contrôle d'accès, validation)

**Compétences méthodologiques :**

* Architecture MVC et séparation des responsabilités
* Conception orientée objet (entités, services, repositories)
* Gestion de projet agile
* Documentation technique
* Tests et débogage

## 

## 8.3 Difficultés Rencontrées et Solutions

**Difficulté 1 : Gestion des relations bidirectionnelles Doctrine**

*Problème :* Incohérences dans la synchronisation des collections ManyToMany entre Project et User.

*Solution :* Ajout de méthodes de synchronisation dans les setters pour maintenir la cohérence des deux côtés de la relation.

php

public function addCollaborator(User $collaborator): static  
{  
 if (!$this->collaborators->contains($collaborator)) {  
 $this->collaborators->add($collaborator);  
 $collaborator->addCollaboration($this); // Synchronisation  
 }  
 return $this;  
}

**Difficulté 2 : Requêtes N+1**

*Problème :* Multiplication des requêtes SQL lors de l'affichage de listes avec relations.

*Solution :* Utilisation de LEFT JOIN dans les QueryBuilder et création de méthodes de repository optimisées.

php

public function findProjectsByUserWithCollaborations(User $user): array  
{  
 return $this->createQueryBuilder('p')  
 ->leftJoin('p.tasks', 't')  
 ->where('p.owner = :user OR t.assignee = :user')  
 ->setParameter('user', $user)  
 ->groupBy('p.id')  
 ->getQuery()  
 ->getResult();  
}

**Difficulté 3 : Gestion des permissions complexes**

*Problème :* Logique de permissions différente selon le contexte (propriétaire, collaborateur, assigné).

*Solution :* Implémentation de Voters Symfony pour centraliser la logique d'autorisation.

php

private function canEdit(Project $project, User $user): bool  
{  
 // Seul le propriétaire peut modifier le projet  
 return $project->getOwner() === $user;  
}

**Difficulté 4 : Notifications en temps réel sans WebSockets**

*Problème :* Besoin d'actualiser les notifications sans surcharger le serveur.

*Solution :* Système de polling intelligent avec intervalle de 30 secondes et pause automatique quand l'onglet n'est pas visible.

javascript

function handleVisibilityChange() {  
 isDocumentVisible = !document.hidden;  
 if (isDocumentVisible) {  
 updateInterval = setInterval(updateNotificationCount, 30000);  
 } else {  
 clearInterval(updateInterval);  
 }  
}

## 

## 8.4 Perspectives d'évolution

**Améliorations à court terme :**

* Ajout de commentaires sur les tâches
* Système de tags/labels pour tâches et projets
* Export PDF/Excel des projets
* Filtres avancés avec recherche textuelle
* Pièces jointes sur les tâches

**Évolutions à moyen terme :**

* Tableau Kanban avec drag & drop
* Diagramme de Gantt pour visualisation temporelle
* Notifications par email configurables
* API REST pour applications tierces
* Application mobile native (React Native/Flutter)
* Intégration calendrier (Google Calendar, Outlook)

**Améliorations techniques :**

* Tests automatisés (PHPUnit, Symfony Panther)
* CI/CD avec GitHub Actions
* Monitoring et alerting (Sentry)
* Cache Redis pour performances
* WebSockets pour notifications temps réel (Mercure)
* Elasticsearch pour recherche avancée
* Docker pour conteneurisation

**Fonctionnalités avancées :**

* Rapports et statistiques avancés
* Templates de projets réutilisables
* Système de rappels automatiques
* Intégration Slack/Teams
* Mode hors ligne (PWA)
* Tableau de bord personnalisable

## 

## 8.5 Retour d'expérience

Ce projet m'a permis de consolider mes compétences en développement web full-stack et de comprendre l'importance d'une architecture solide. La mise en place d'un système de collaboration avec gestion fine des permissions a été particulièrement enrichissante.

**Points forts du projet :**

* Architecture MVC bien structurée
* Code maintenable et documenté
* Interface utilisateur intuitive
* Sécurité renforcée à tous les niveaux
* Performance optimisée

**Apprentissages clés :**

* L'importance de la planification et de la conception avant le développement
* La valeur d'une bonne séparation des responsabilités
* L'utilité des services pour centraliser la logique métier
* L'efficacité des Voters pour la gestion des permissions
* L'importance des tests pour garantir la fiabilité

L'utilisation de Symfony m'a fait apprécier les avantages d'un framework mature avec ses conventions et son écosystème. La courbe d'apprentissage a été compensée par la productivité et la maintenabilité du code produit.

**Impact personnel :** Ce projet a renforcé ma confiance en mes capacités à concevoir et développer une application complète. J'ai appris à gérer un projet de A à Z, de la conception à la mise en production, en passant par le développement et les tests.

TaskFlow démontre ma capacité à concevoir et développer une application web complète, sécurisée et performante, en respectant les bonnes pratiques du développement moderne. Cette expérience m'a préparé à relever les défis du développement web professionnel.

# 9. Annexes

### 

### Annexe A : Schéma de Base de Données

**Diagramme Entité-Relations :**

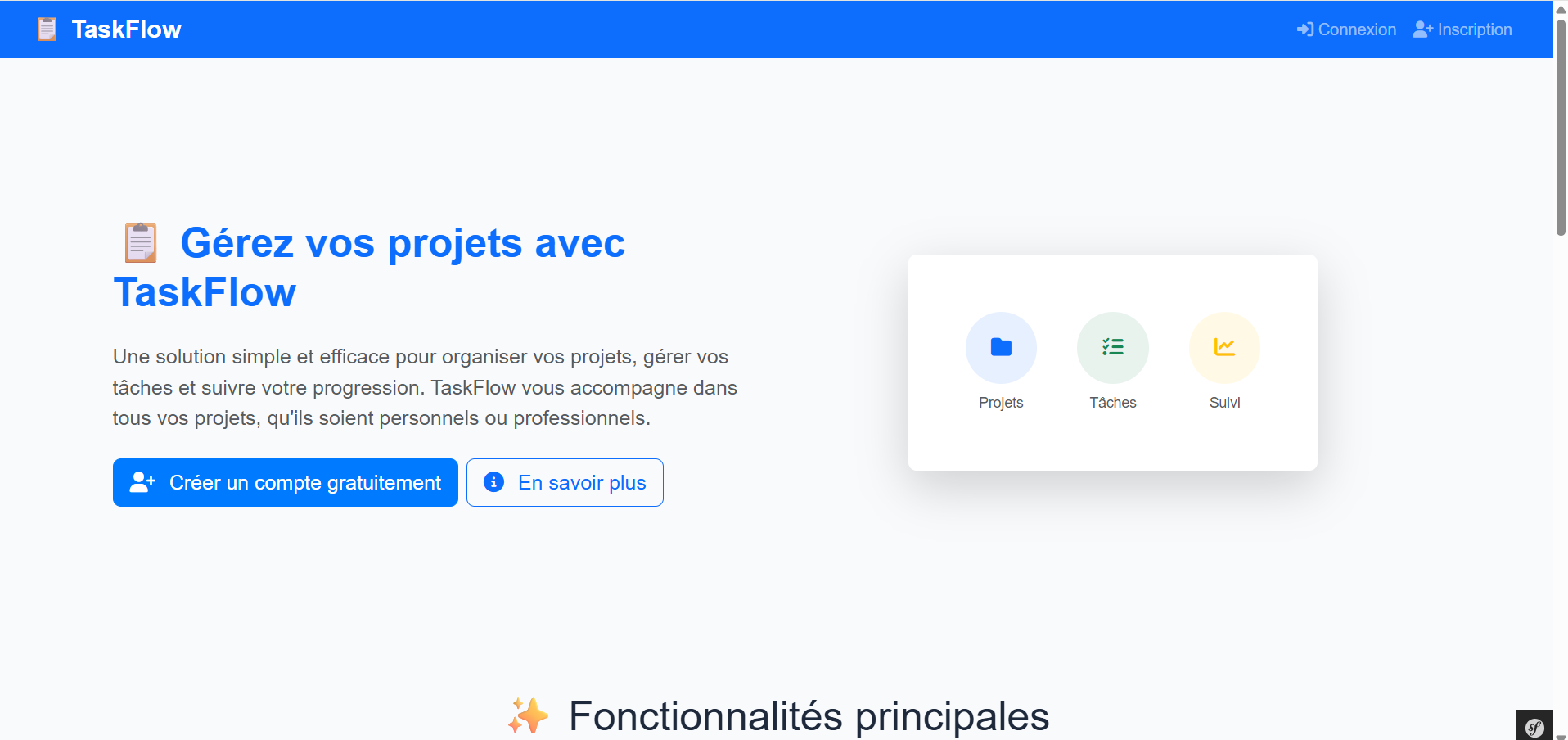
┌─────────────────────┐  
│ User │  
├─────────────────────┤  
│ id (PK) │  
│ email (UNIQUE) │  
│ password │  
│ firstName │  
│ lastName │  
│ roles (JSON) │  
│ isVerified │  
│ createdAt │  
│ resetToken │  
│ resetTokenExpiresAt │  
└─────────────────────┘  
 │  
 │ 1:N (owner)  
 ▼  
┌─────────────────────┐ ┌─────────────────────┐  
│ Project │◄───────►│ Collaborators │  
├─────────────────────┤ M:N │ (User-Project) │  
│ id (PK) │ └─────────────────────┘  
│ title │  
│ description │  
│ owner\_id (FK) │  
│ createdAt │  
│ updatedAt │  
└─────────────────────┘  
 │  
 │ 1:N  
 ▼  
┌─────────────────────┐  
│ Task │  
├─────────────────────┤  
│ id (PK) │  
│ title │  
│ description │  
│ status │  
│ priority │  
│ dueDate │  
│ project\_id (FK) │  
│ assignee\_id (FK) │  
│ createdAt │  
│ updatedAt │  
│ completedAt │  
└─────────────────────┘  
  
┌──────────────────────────┐  
│ CollaborationRequest │  
├──────────────────────────┤  
│ id (PK) │  
│ status │  
│ message │  
│ response │  
│ project\_id (FK) │  
│ sender\_id (FK) │  
│ invited\_user\_id (FK) │  
│ createdAt │  
│ updatedAt │  
│ respondedAt │  
│ UNIQUE(project, invited) │  
└──────────────────────────┘  
  
┌─────────────────────┐  
│ Notification │  
├─────────────────────┤  
│ id (PK) │  
│ type │  
│ title │  
│ message │  
│ status │  
│ recipient\_id (FK) │  
│ sender\_id (FK) │  
│ project\_id (FK) │  
│ task\_id (FK) │  
│ actionUrl │  
│ data (JSON) │  
│ createdAt │  
│ readAt │  
└─────────────────────┘

**Contraintes d'intégrité :**

* CASCADE sur Project → Task (suppression de projet supprime ses tâches)
* SET NULL sur Task → User (suppression d'utilisateur désassigne ses tâches)
* CASCADE sur CollaborationRequest → Project
* UNIQUE sur (project\_id, invited\_user\_id) dans CollaborationRequest

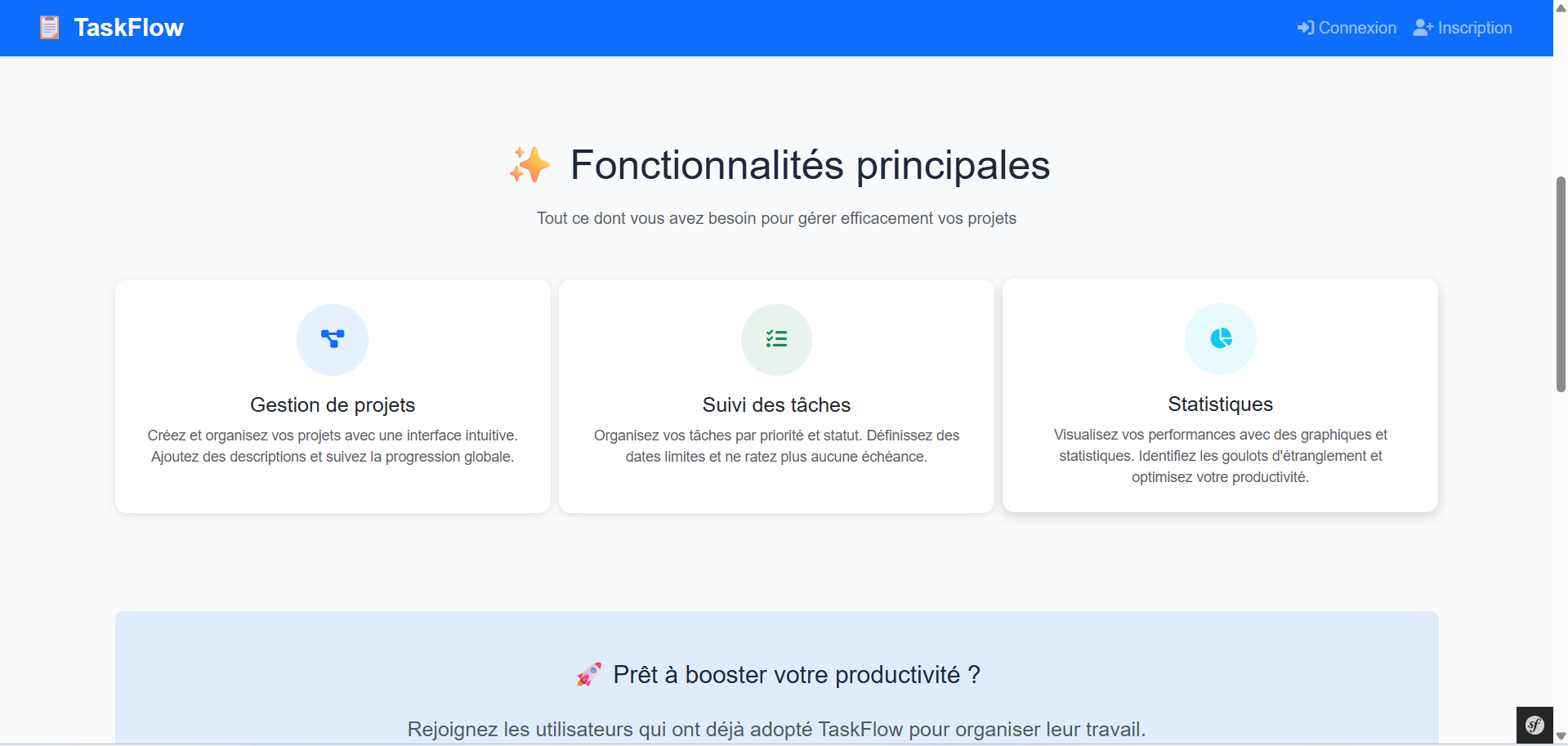
### Annexe B : Captures d'Écran

1. **Page d'accueil**
   * Vue desktop et mobile

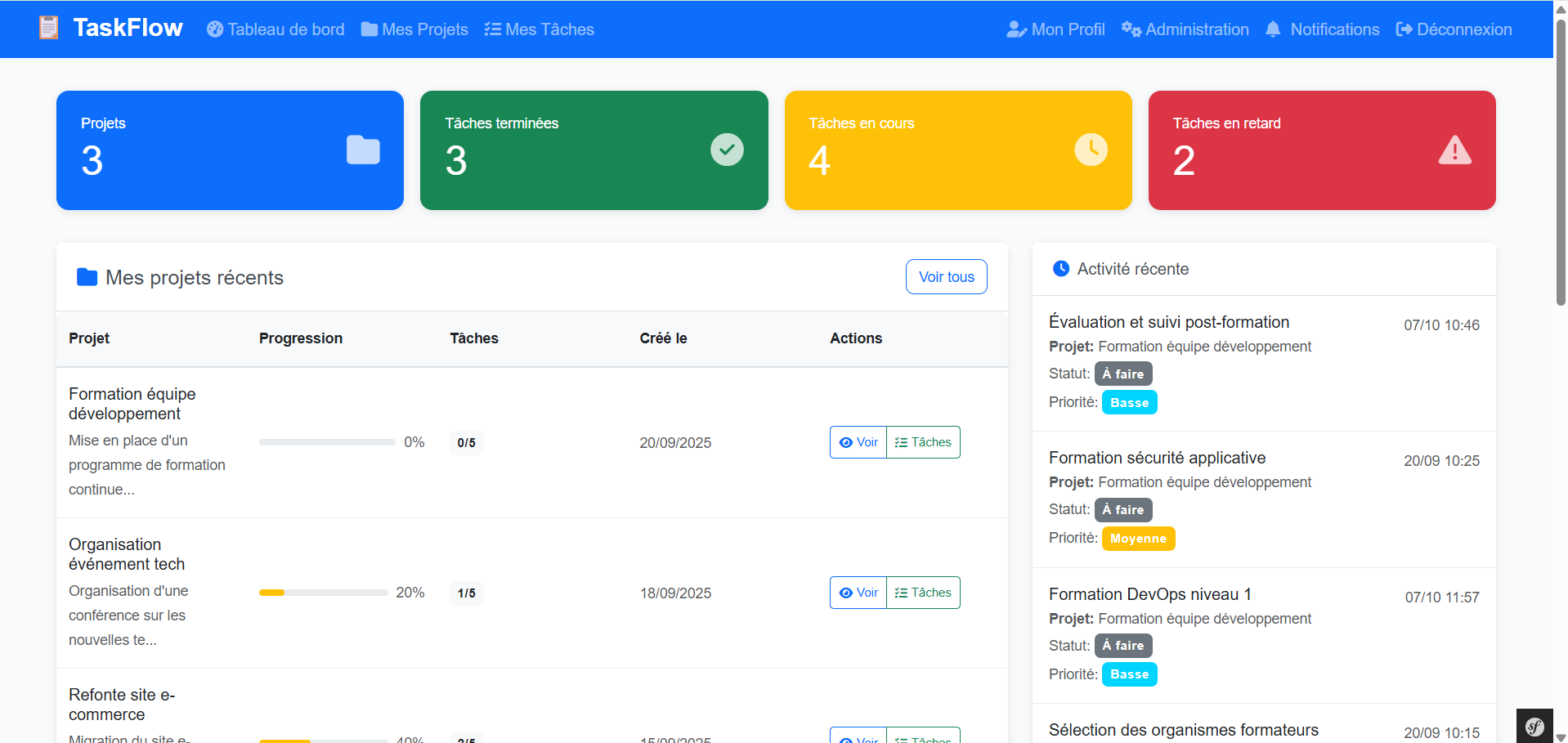




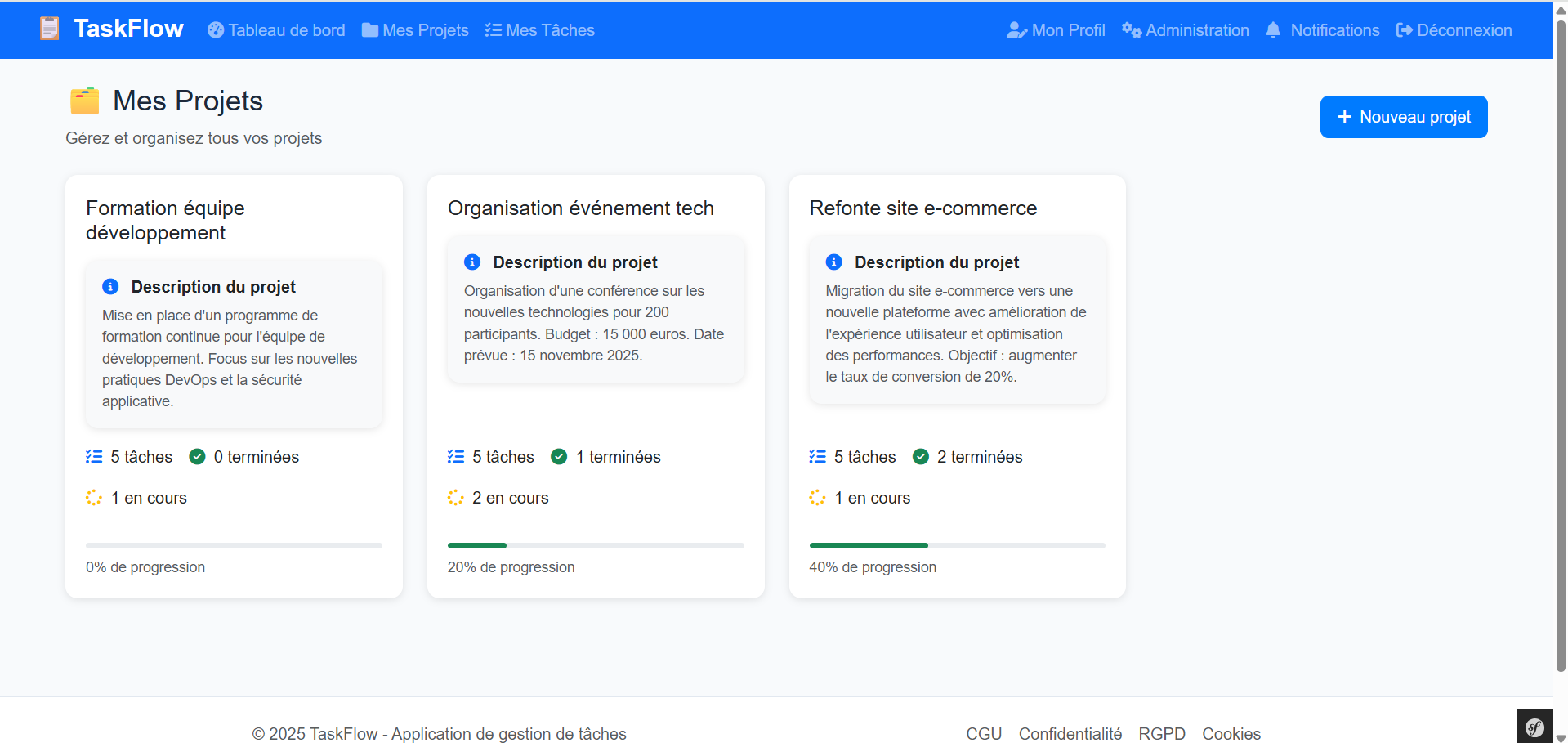
* + Sections de présentation des fonctionnalités



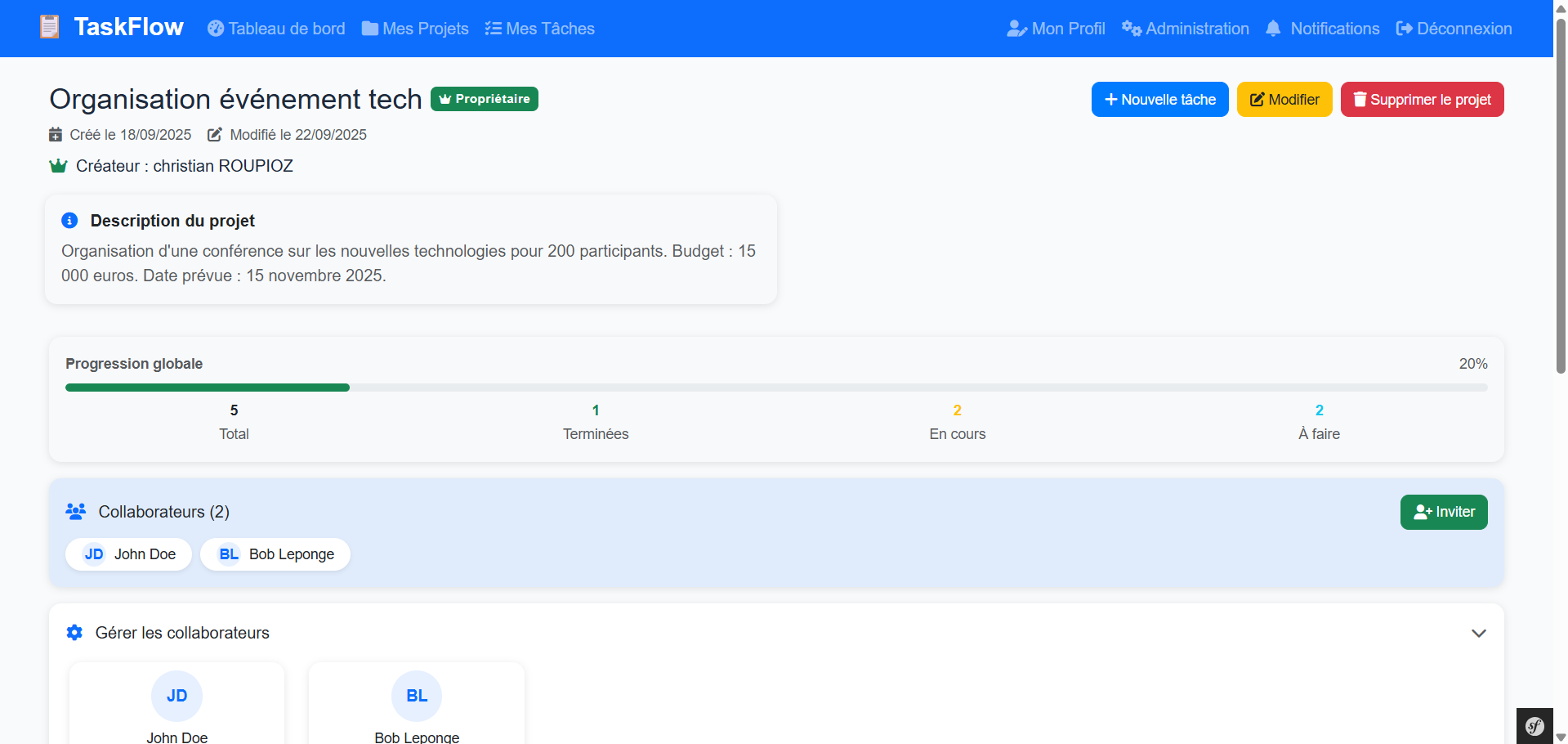
1. **Dashboard**
   * Vue d'ensemble des statistiques



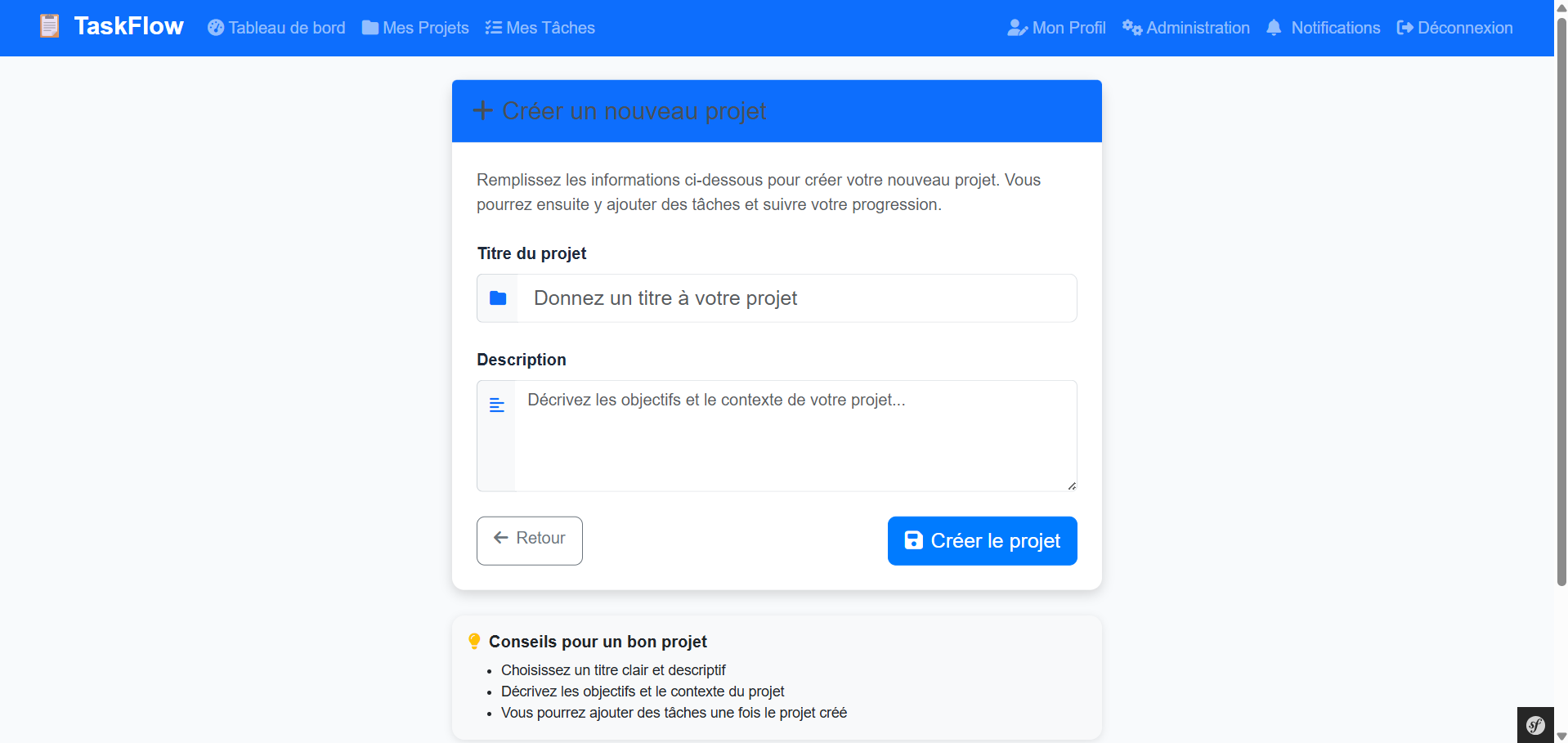
1. **Gestion de projets**
   * Liste des projets

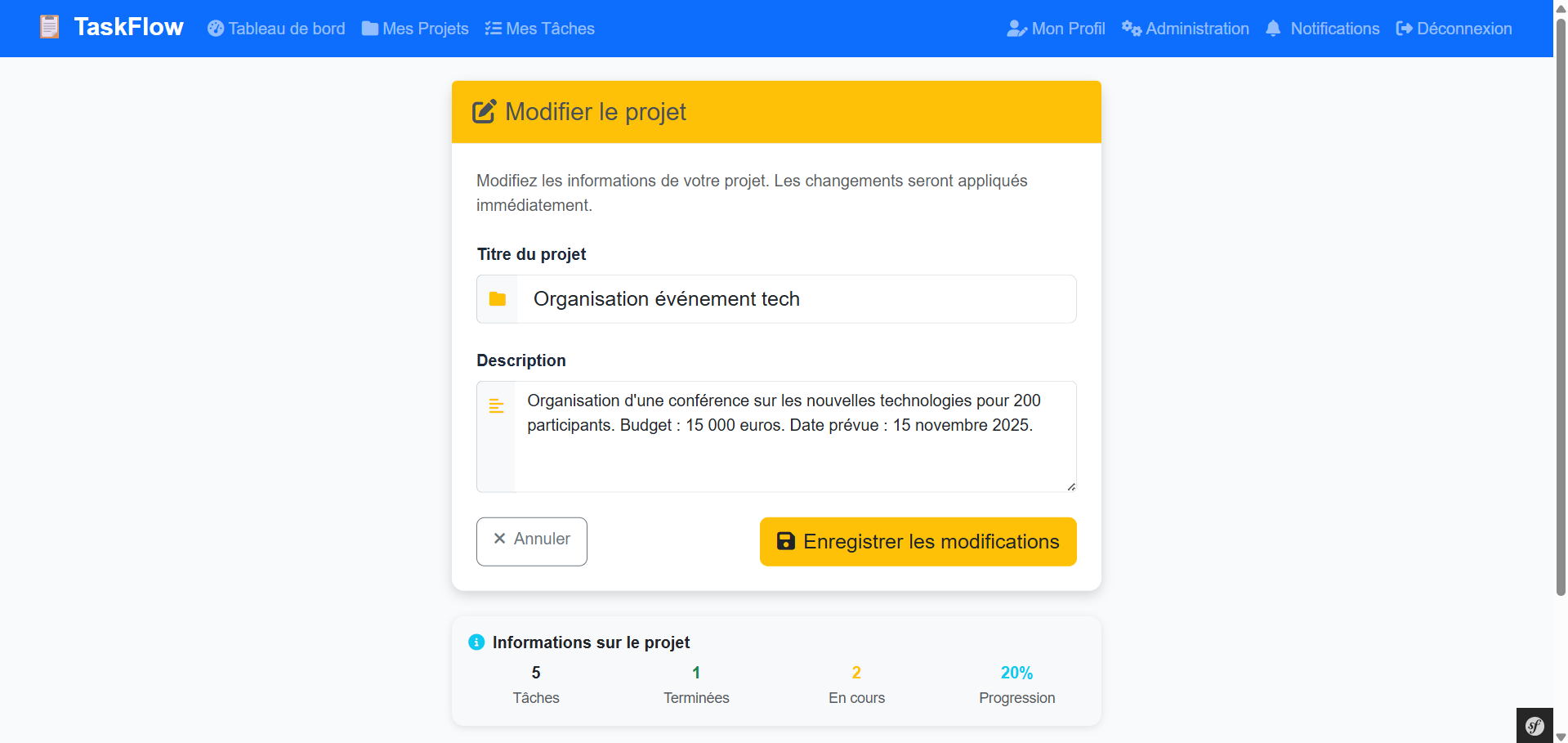


* + Détail d'un projet avec taches

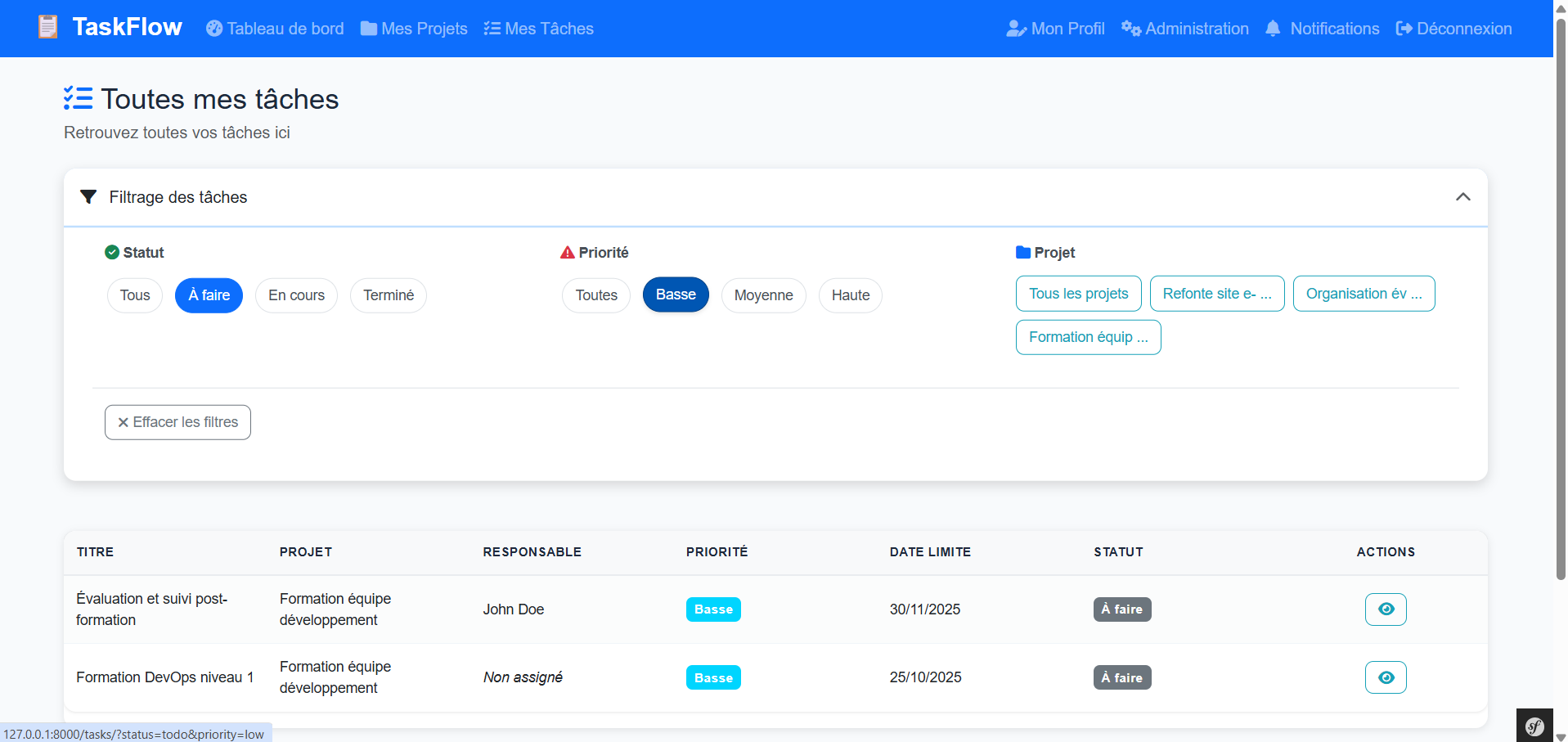


* + Formulaire de création/edition

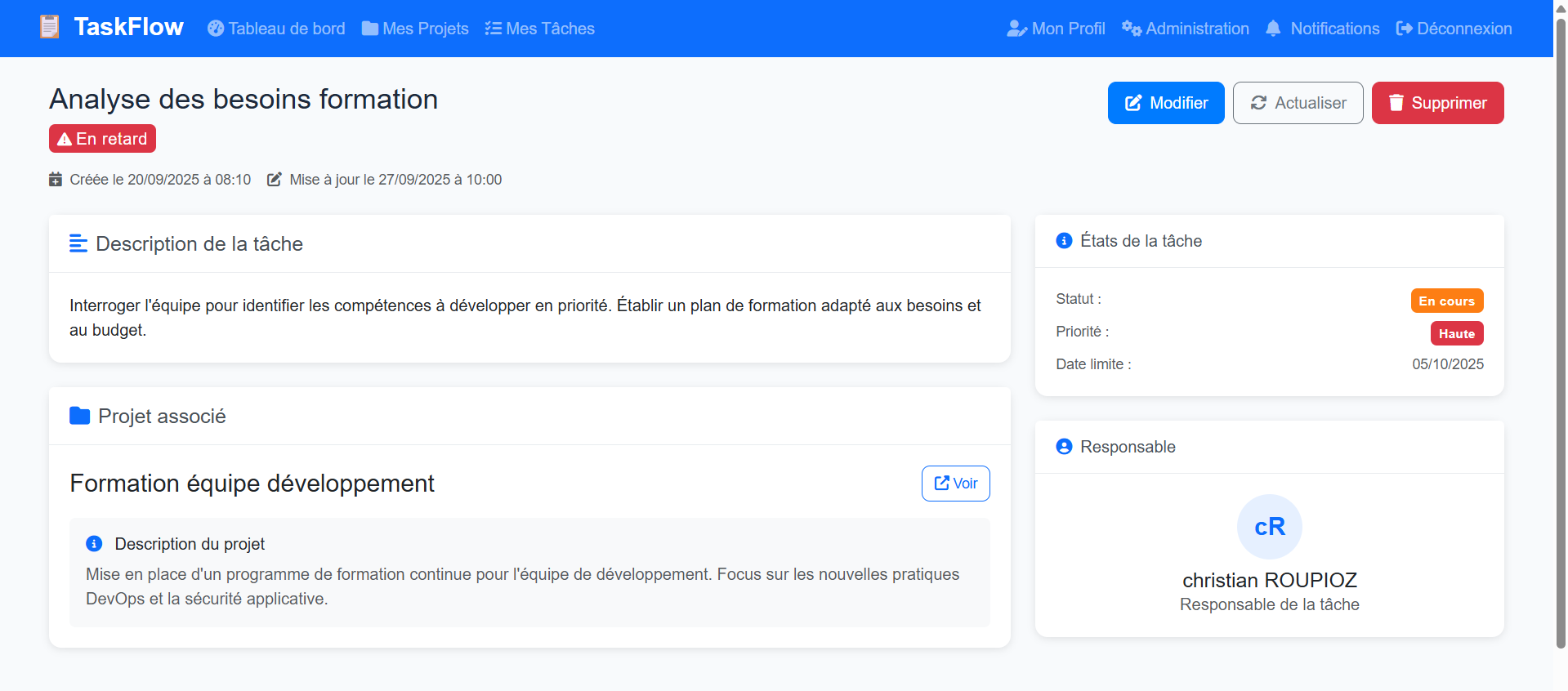




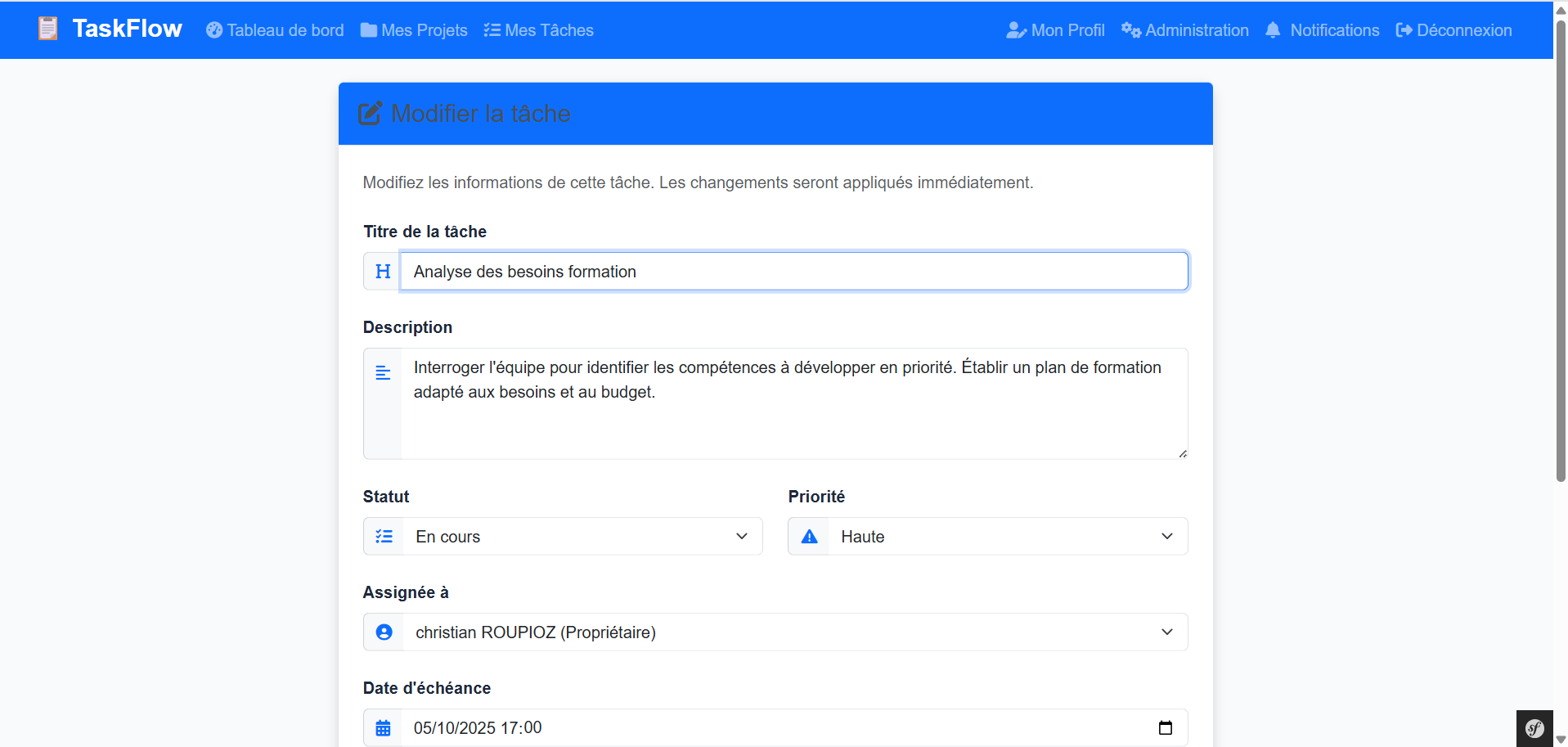
1. **Gestion de taches**
   * Liste des tâches avec filtres



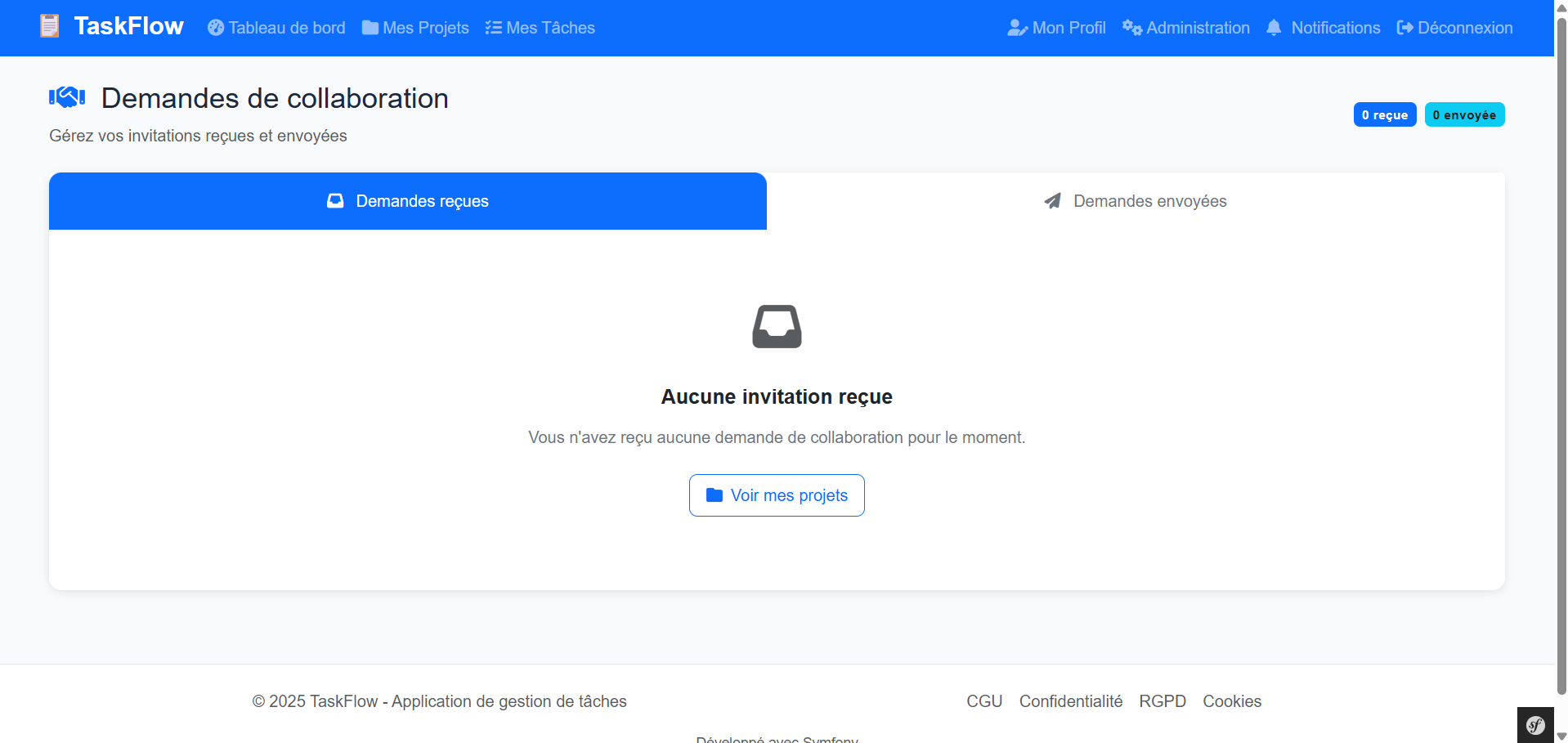
* + Détail d'une tâche



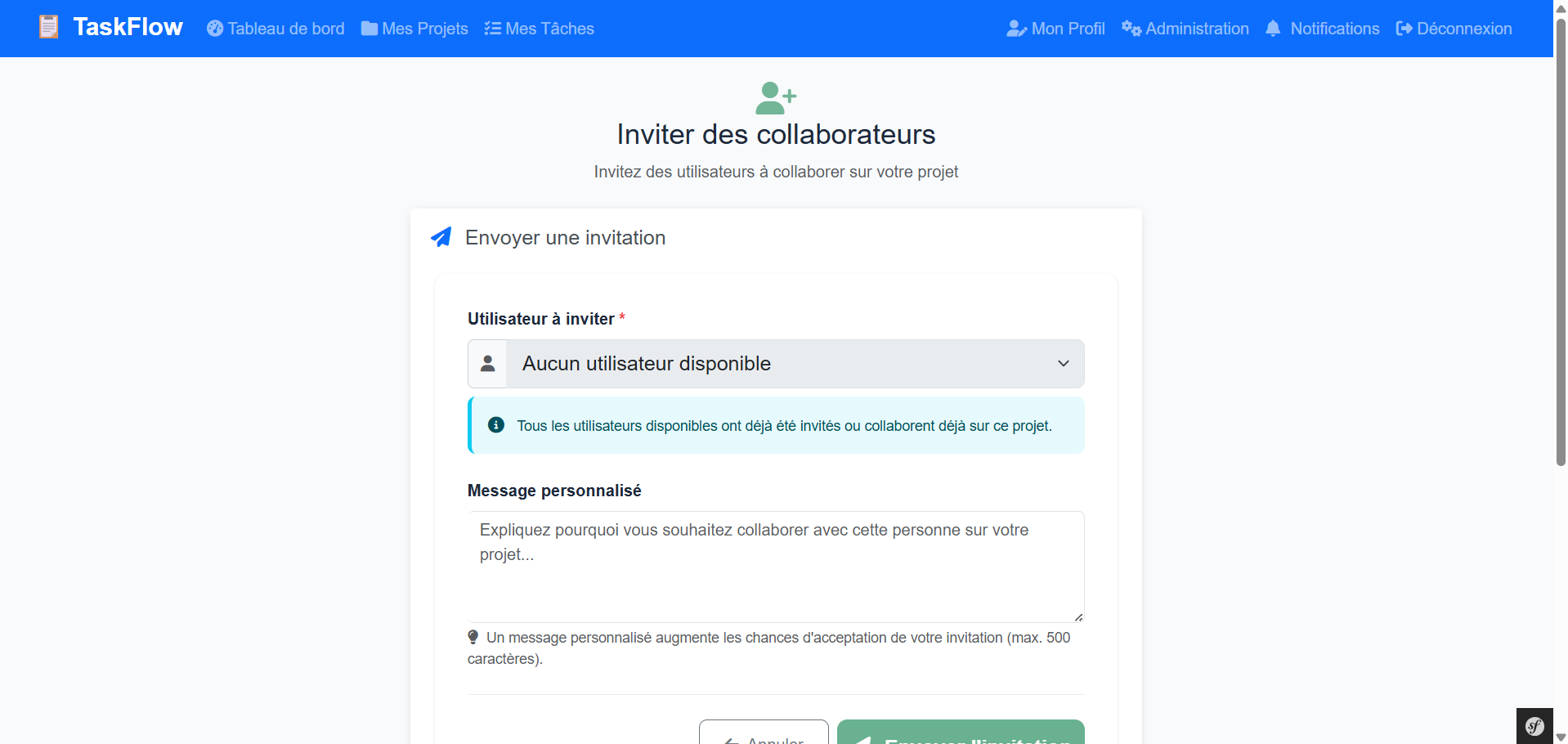
* + Formulaire d'édition



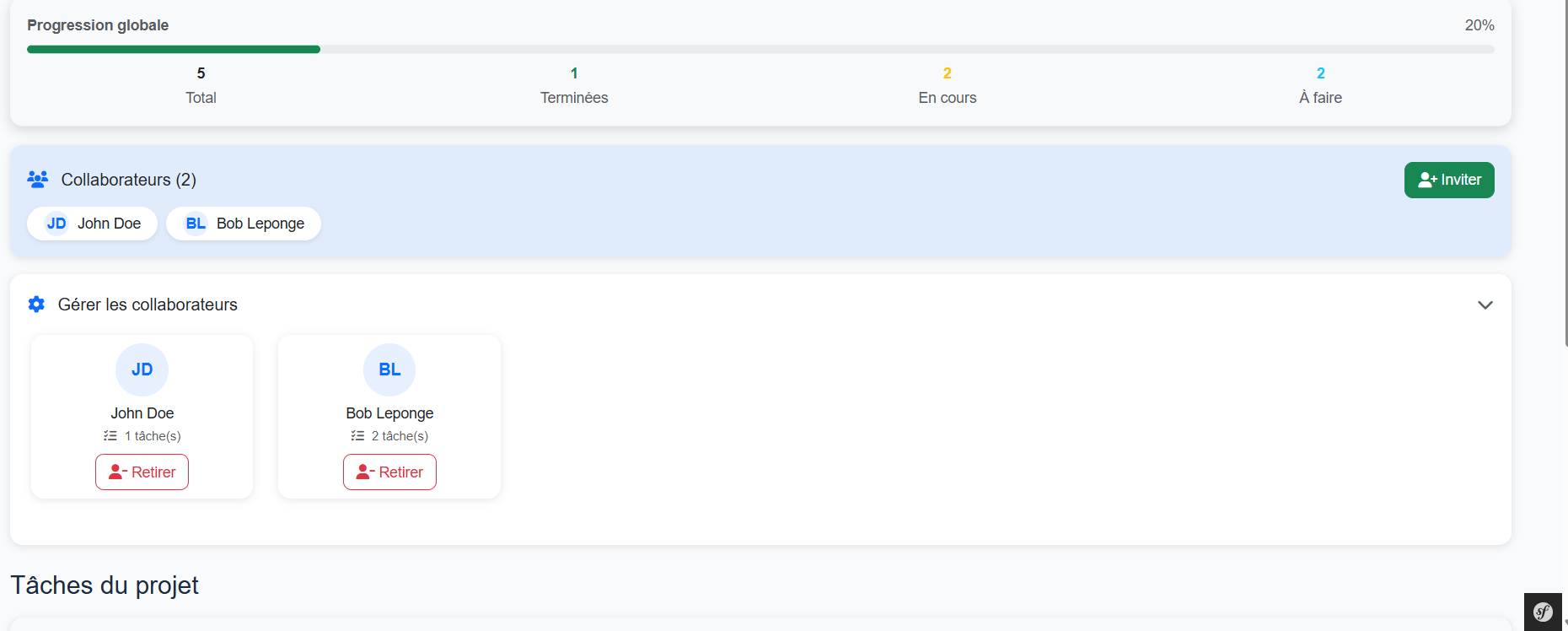
1. **Collaboration**
   * Liste des demandes de collaboration



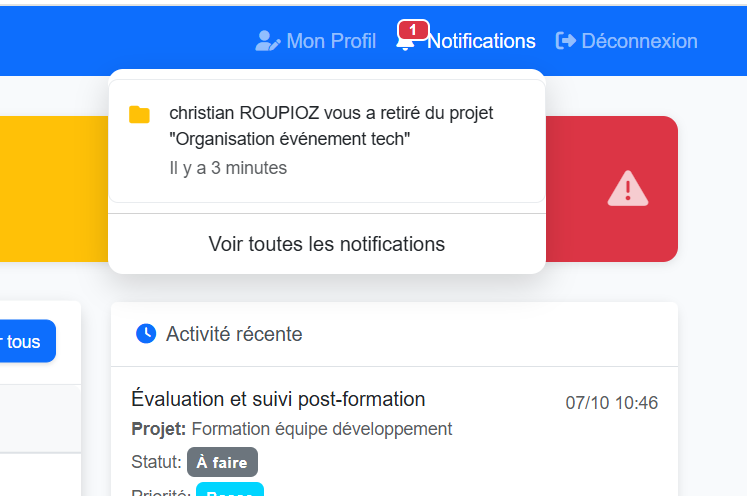
* + Formulaire d'invitation



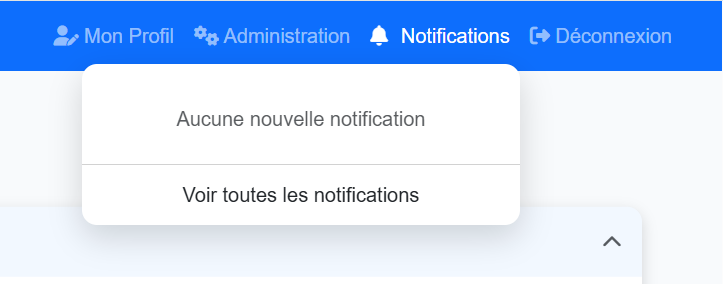
* + Interface de gestion des collaborateurs



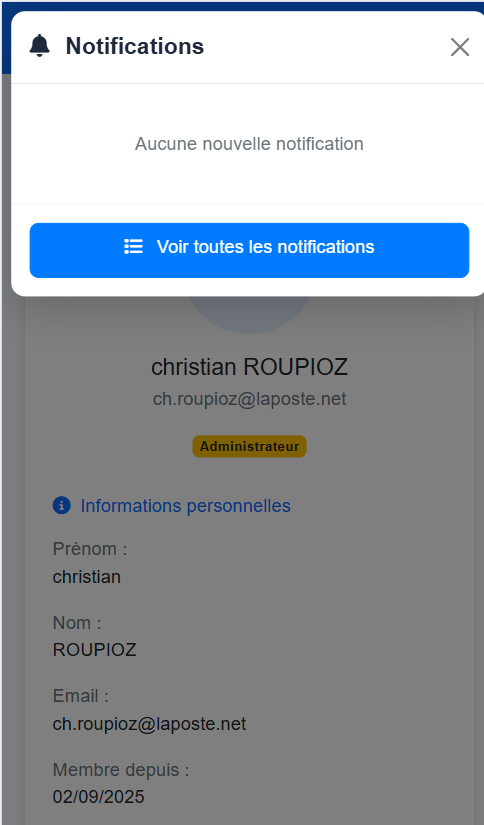
1. **Notifications**
   * Badge de compteur



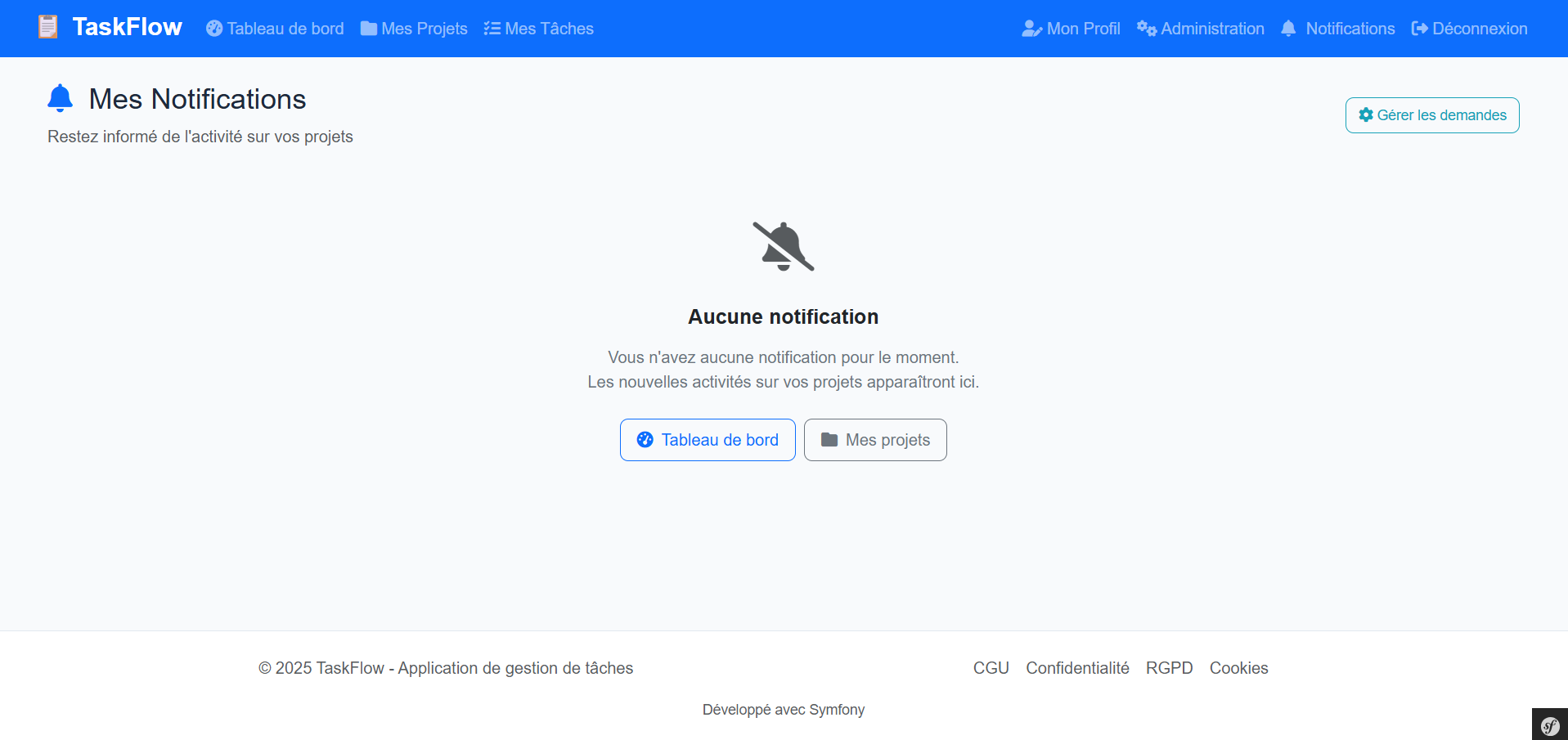
* + Liste déroulante (desktop)



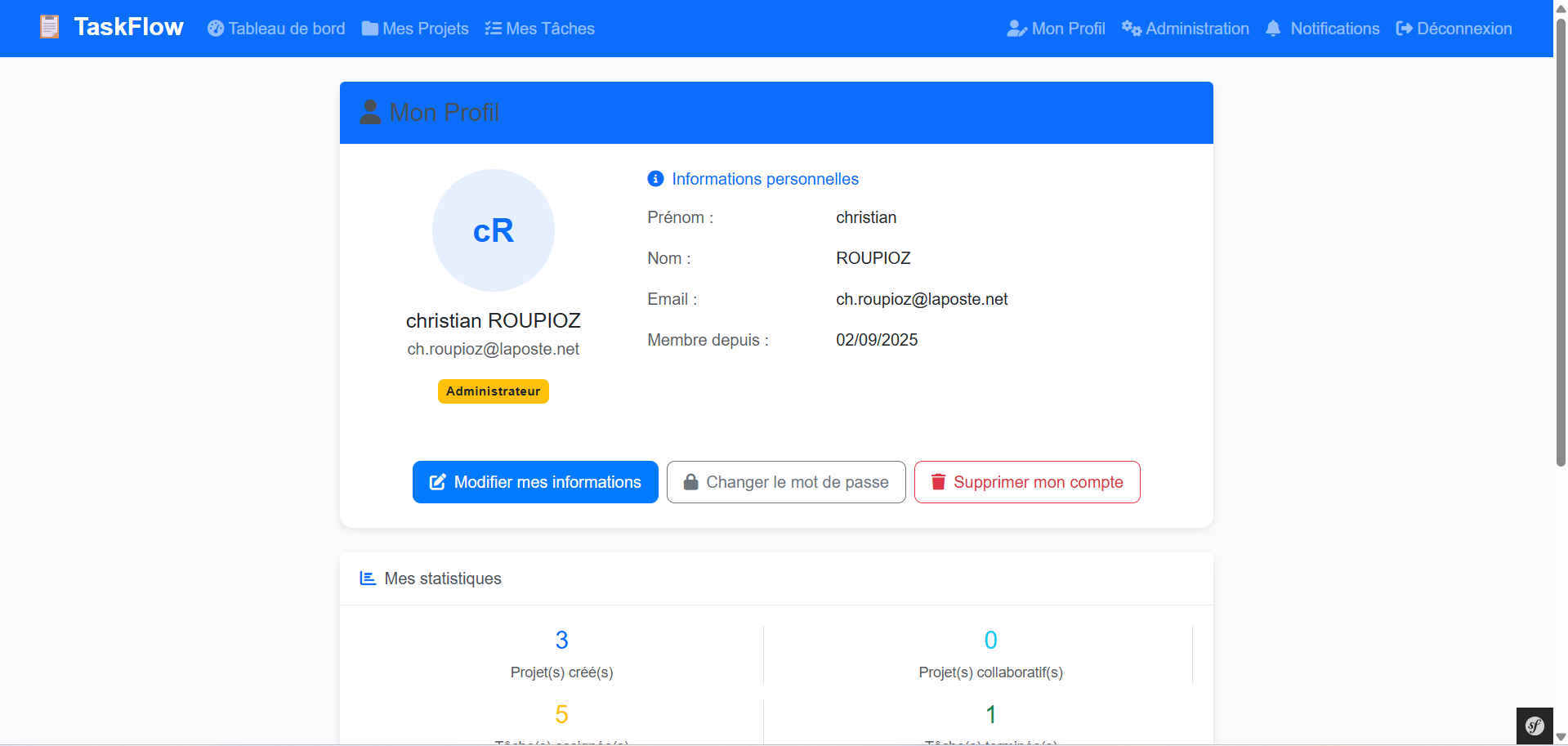
* + Modal (mobile)



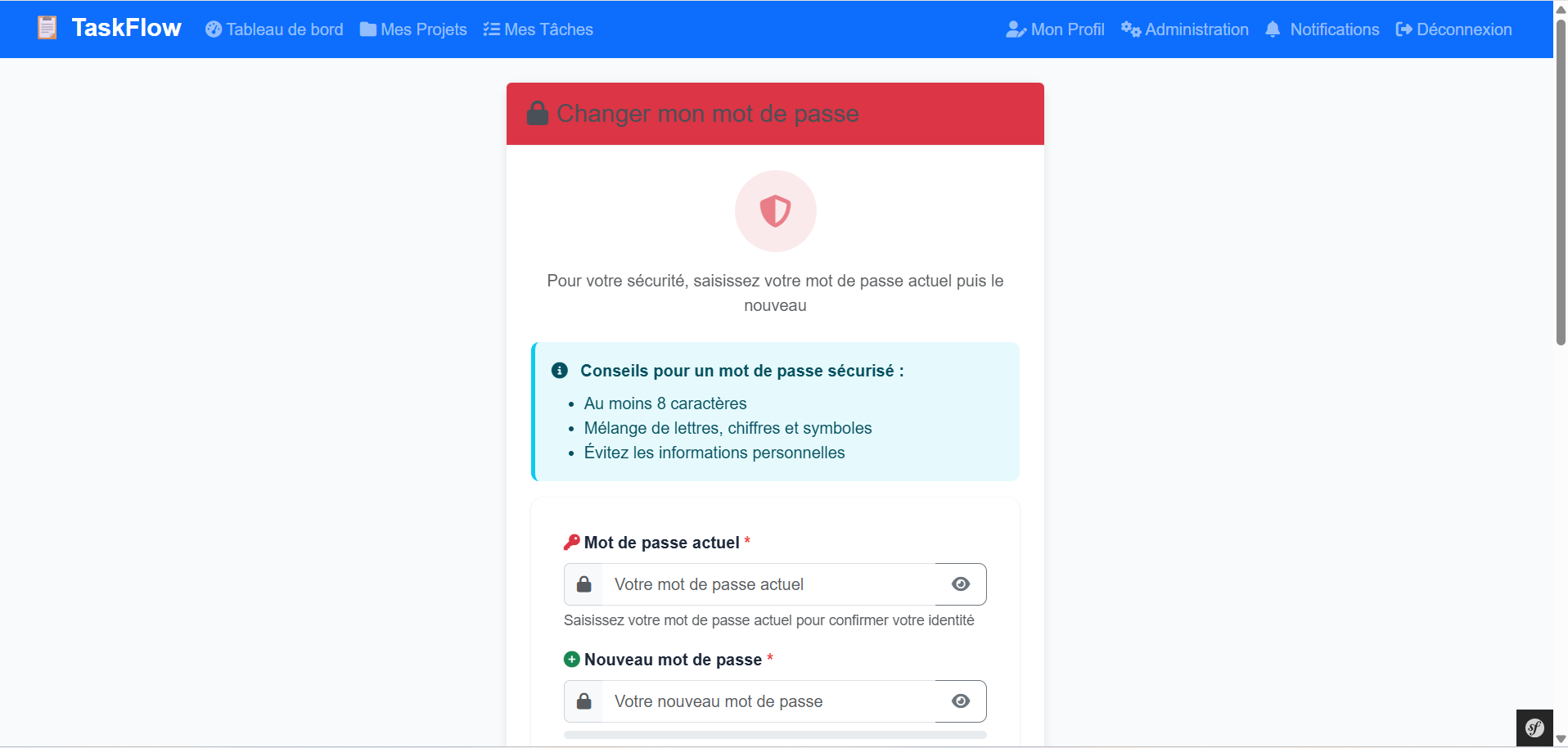
* + Page historique complet



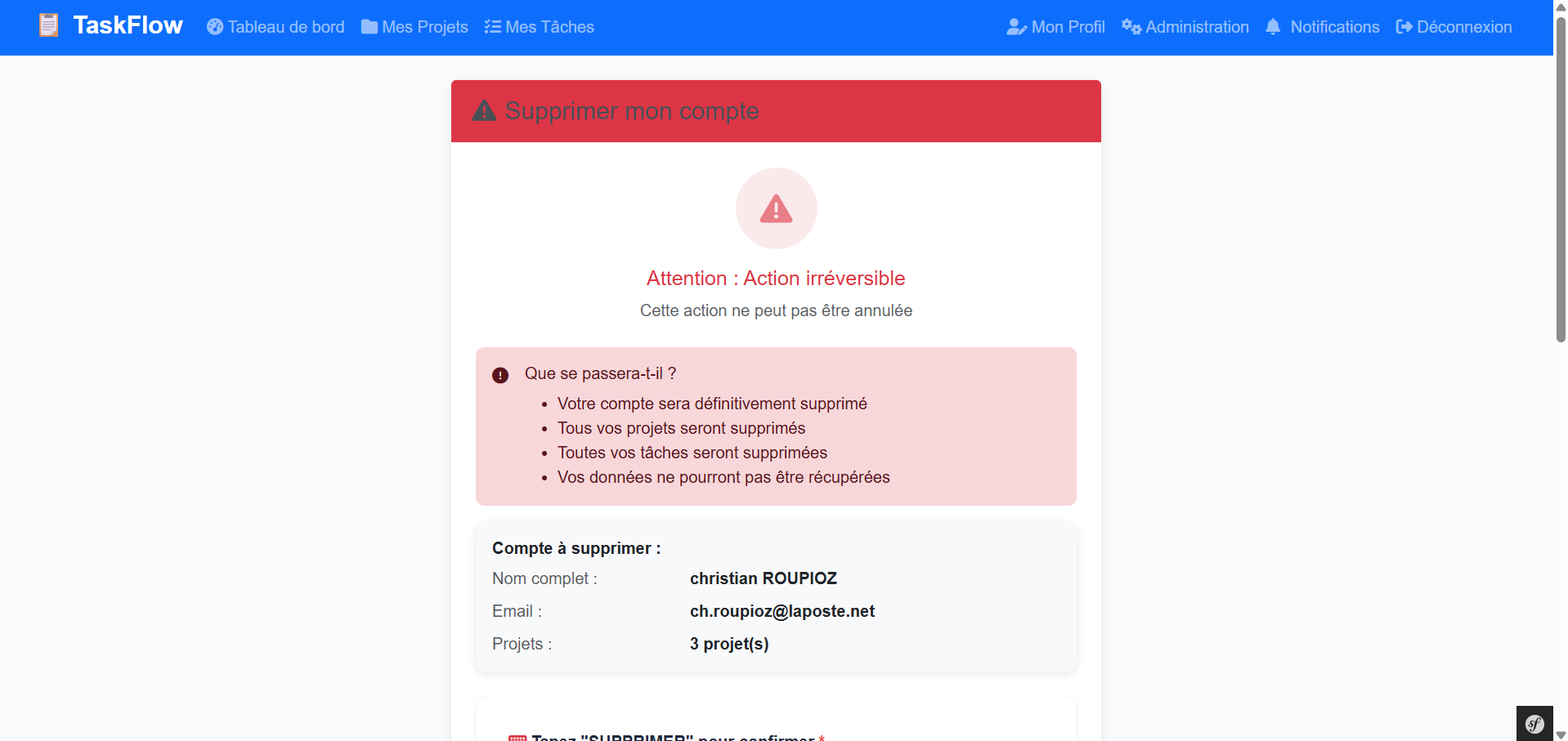
1. **Profil utilisateur**
   * Informations du profil

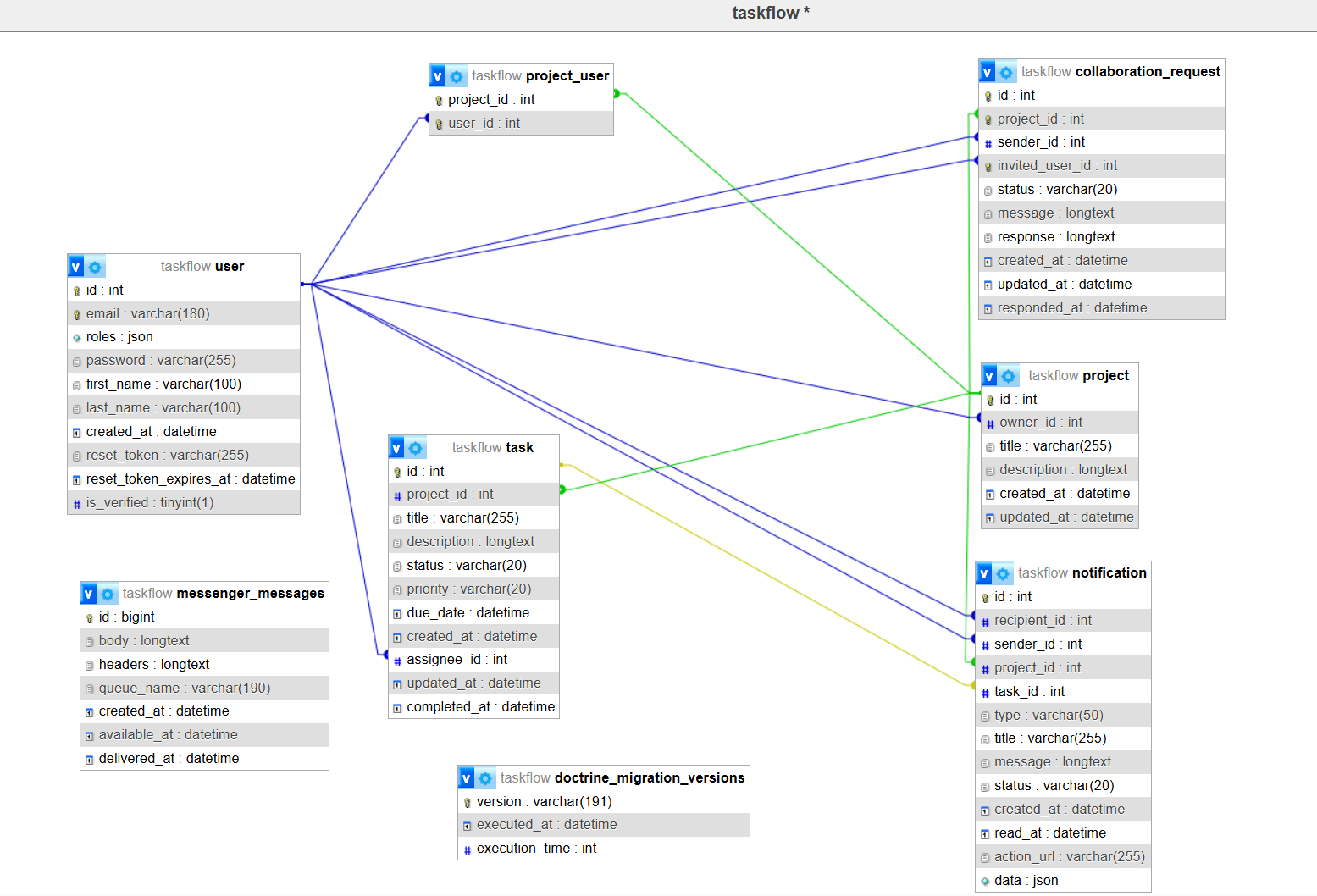


* + Changement de mot de passe



* + Suppression de compte





### Annexe C : Glossaire

**Termes techniques :**

* **MVC** : Model-View-Controller, architecture de séparation des responsabilités
* **ORM** : Object-Relational Mapping, technique de mapping entre objets et base de données
* **CSRF** : Cross-Site Request Forgery, type d'attaque web
* **Voter** : Composant Symfony pour la gestion des autorisations
* **Repository** : Classe responsable de l'accès aux données d'une entité
* **Service** : Classe réutilisable contenant de la logique métier
* **Entity** : Classe représentant une table de base de données
* **Doctrine** : ORM utilisé par Symfony
* **Twig** : Moteur de templates de Symfony
* **AssetMapper** : Système de gestion des assets de Symfony

**Termes métier :**

* **Projet** : Conteneur de tâches liées à un objectif commun
* **Tâche** : Élément de travail avec statut, priorité et échéance
* **Collaborateur** : Utilisateur invité à participer à un projet
* **Assigné** : Utilisateur responsable d'une tâche spécifique
* **Notification** : Message informant l'utilisateur d'un événement
* **Dashboard** : Tableau de bord avec vue d'ensemble
* **Statut** : État d'avancement d'une tâche (À faire, En cours, Terminé)
* **Priorité** : Niveau d'importance d'une tâche (Basse, Moyenne, Haute)

## Bibliographie et Ressources

**Documentation officielle :**

* Symfony Documentation : <https://symfony.com/doc/current/index.html>
* Doctrine ORM : <https://www.doctrine-project.org/projects/doctrine-orm/en/current/>
* Bootstrap 5 : <https://getbootstrap.com/docs/5.3/>
* PHP 8.3 : <https://www.php.net/manual/fr/>

**Ressources d'apprentissage :**

* SymfonyCasts : <https://symfonycasts.com/>
* MDN Web Docs : <https://developer.mozilla.org/>
* PHP The Right Way : <https://phptherightway.com/>

**Outils utilisés :**

* Font Awesome : <https://fontawesome.com/>
* GitHub : <https://github.com/>
* Composer : [https://getcomposer.org/``](https://getcomposer.org/%60%60)