BTS SIO - Option SLAM **Marvin Forget-Mandras**

Plateforme de Partage de Cours en PDF

Voir le projet (en prod) : marvinfm.fr/course

0. Contexte du projet

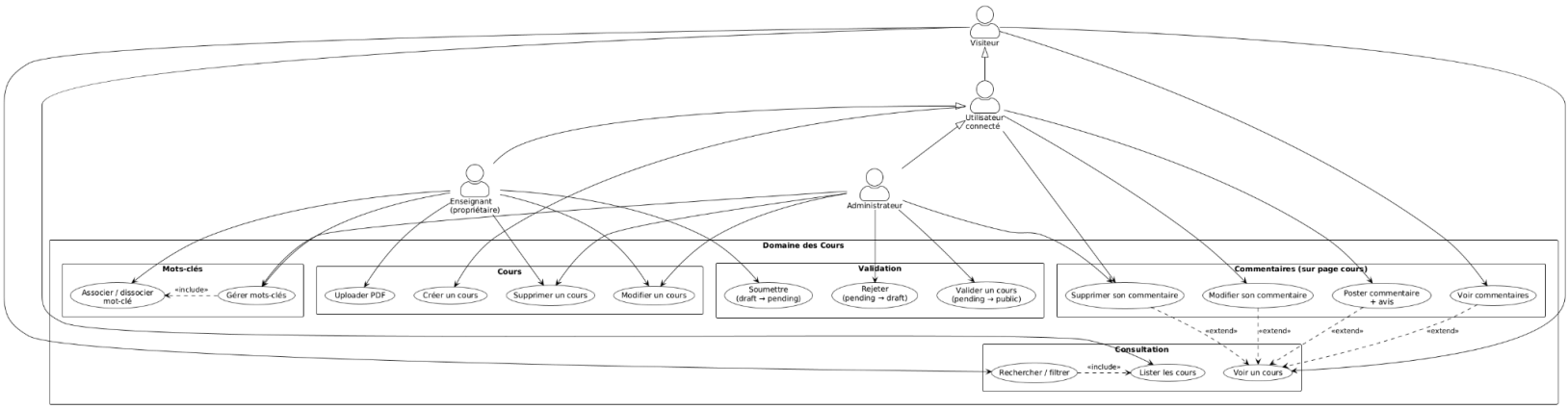
Ce projet s'inscrit dans le cadre d'un recueil de projets web, hébergés sur marvinfm.fr. Il s'agit d'une plateforme web communautaire permettant à des enseignants, formateurs et étudiants de déposer des fichiers PDF (fiches de révision, supports pédagogiques, ressources d'apprentissage, etc.), afin de les partager avec une communauté d'apprenants.

Le projet répond à un besoin réel : centraliser des ressources éducatives variées dans un espace structuré, avec un système de validation éditoriale garantissant la qualité du contenu publié.

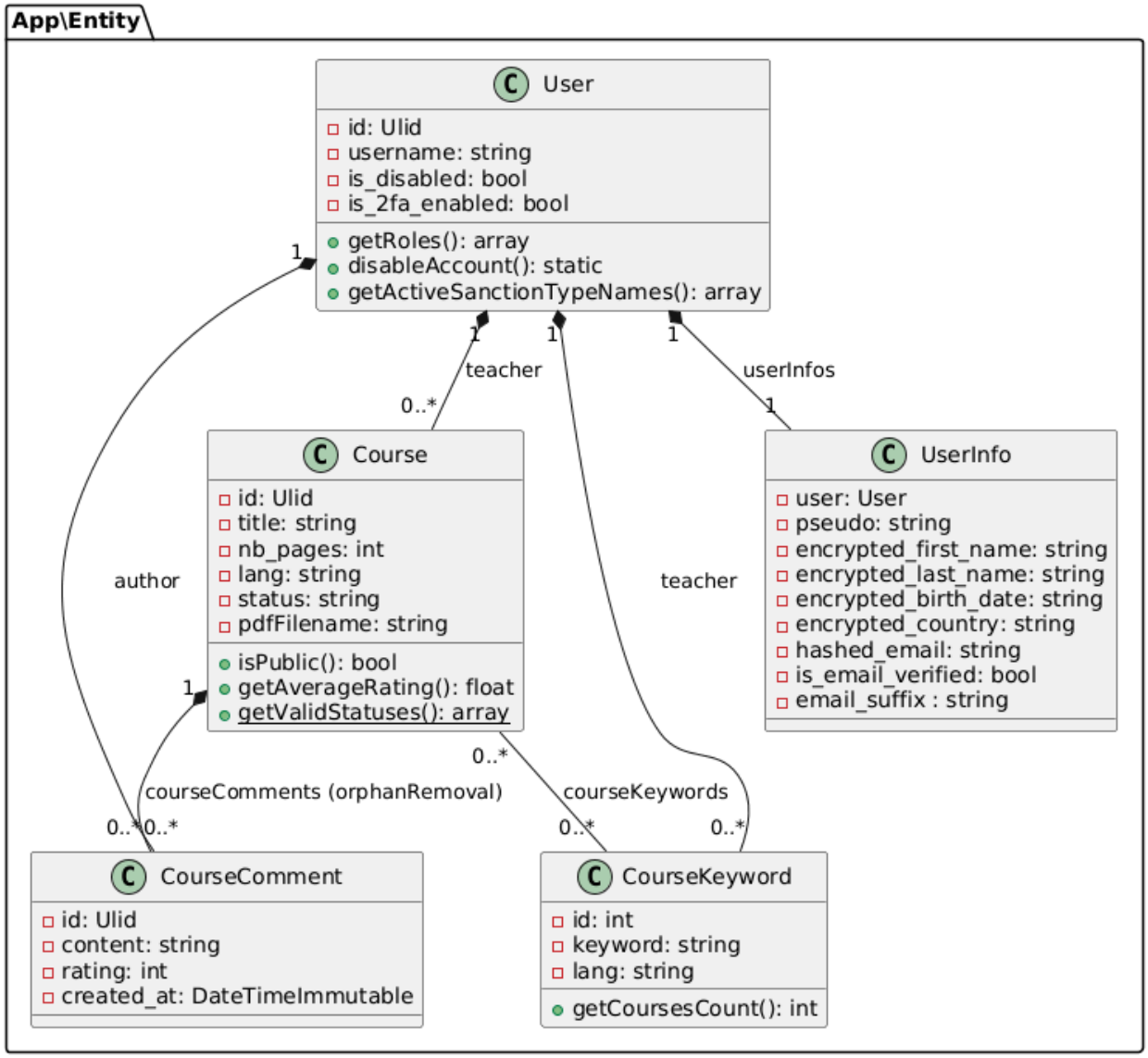
> Objectif principal : mettre en place une gestion complète du cycle de vie d'un cours, depuis sa création par un enseignant jusqu'à sa publication validée par un administrateur, en passant par l'upload du fichier PDF, la gestion des mots-clés et les commentaires de la communauté.

1. Conception

1.1 Diagramme de cas d'utilisation



1.2 Diagramme de classes



1.3 Architecture (Controllers)

Nom du controller	Namespace	Description du controller
BaseController.php	App\Controller	Controller de base du projet : tous les controllers passent par lui. Il gère le système de messages flash et le système de traductions, notamment grâce à “renderWithTranslations()”.
Contrôleurs du domaine principal		
ActuController.php	App\Controller	Controller qui gère les pages liées au domaine de l’actualité, il utilise les services de gestion des actu (Exemple : ActuRssService.php) et des commandes (Exemple : ActuWarmCacheCommand.php).
AdminController.php	App\Controller	Gère le panel admin, avec un système similaire à SettingController (AdminTableField a un fonctionnement similaire à AccountField) *Se référer à SettingController.*
HomeController.php	App\Controller	Gère la page d’accueil, la page de présentation des services, la politique de confidentialité, les conditions d'utilisation ou encore la page de contact.
LanguageController.php	App\Controller	Permet de changer la langue d’affichage après avoir choisi à travers la popup des langues. La liste des langues se trouve dans FmConfig.
LoginTwoFactorController.php	App\Controller	Gère la page d’A2F : - Permet d’envoyer un email (L’utilisateur doit obligatoirement renseigner son email); - Permet de rentrer le code reçu par email. Si l’A2F est activé et que l’utilisateur vient de se connecter, il sera forcément redirigé vers cette page. S’il n’est pas en mesure de rentrer le code, il peut se déconnecter.
PasswordResetController.php	App\Controller	Gère deux pages : - Page d’envoie de lien de réinitialisation par email (nécessite de renseigner son email ET son username) - Page de modification de mot de passe (accessible via le lien de réinitialisation, avec un token)

RegistrationController.php	App\Controller	Permet de créer un compte, en 3 parties : - Username et password - Email et pseudo - Validation
SecurityController.php	App\Controller	Gère la connexion et la déconnexion des utilisateurs
SettingsController.php	App\Controller	Permet de gérer son compte, avec un système de pages (Chaque page fonctionne grâce à un tableau d'objet AccountField; Un AccountField représente un champ et tous ses paramètres, par exemplenotamment le titre)
TicketController.php	App\Controller	Permet de créer des tickets pour poser des questions, reporter un bug ou encore faire une demande au staff.
UserNoteController.php	App\Controller	Permet de gérer (CRUD) des notes chiffrées, à trouver dans les paramètres utilisateurs.
UserReviewController.php	App\Controller	Permet de laisser un avis avec une note sur le site.
Contrôleurs du domaine forum (en développement, pas encore opérationnel)		
ForumController.php	App\Controller\Forum	Projet de forum avec des branches publiques et privées.
Contrôleurs du domaine des cours		
CourseCrudController.php	App\Controller\Course	Permet de gérer (CRUD) les cours.
CourseCommentController.php	App\Controller\Course	Permet de gérer (CRUD) les commentaires.
CourseUploadController.php	App\Controller\Course	Permet l'upload de PDF en ligne de manière sécurisée.
CourseStatusController.php	App\Controller\Course	Permet d'attribuer, modifier ou supprimer le statut d'un cours (draft, pending, public, deleted).
CourseKeywordController.php	App\Controller\Course	Permet de gérer (CRUD) les mots-clés.
Contrôleurs d'APIs		
ActuCacheController.php	App\Controller\Api	Permet d'actualiser manuellement les flux RSS du domaine d'actualités. Nécessite un accès administrateur.

2. Technologies utilisées

Technologie	Rôle	Version / Détail
PHP 8.4	Langage serveur principal	Typage strict, attributs natifs
Symfony 7	Framework MVC back-end	Routing, Forms, Security, Flash
Doctrine ORM	Abstraction base de données	Entités, Repository, EntityManager
Twig	Moteur de templates	Rendu HTML côté serveur
TypeScript	Scripting front-end typé	Interactions UI, validation client
HTML5 / CSS3	Structure et style des vues	Responsive, BEM
Hostinger	Hébergement de production	Linux, Nginx, MySQL
ULID	Identifiants d'entités	Symfony\Uid\Ulid

Symfony a été retenu pour sa robustesse, son écosystème éprouvé (Security Component, Form Component) et sa facilité d'intégration avec Doctrine ORM. L'usage de TypeScript côté front garantit une meilleure maintenabilité des scripts d'interaction par rapport à du JavaScript natif non typé.

3. Architecture et organisation du code

Le module de gestion des cours est découpé en cinq contrôleurs distincts, chacun responsable d'un périmètre fonctionnel précis. Cette approche respecte le principe de responsabilité unique (SRP) issu des principes SOLID.

3.1 Héritage et services transversaux

Tous les contrôleurs héritent d'un BaseController maison qui encapsule deux comportements communs : l'authentification (via AuthenticationService) et la traduction des messages flash (via TranslationService). Cela évite la duplication de code entre contrôleurs.

```
// Extrait : CourseCrudController.php
final class CourseCrudController extends BaseController
{
   public function __construct(
        AuthenticationService $authService,
        TranslationService $trans,
        private CourseValidationService $validationService,
        private CoursePermissionService $permissionService
   ) {
        parent::__construct($authService, $trans);
   }
}
```

Les services métier (CourseValidationService, CoursePermissionService, PdfFileService, etc.) sont injectés dans les constructeurs des contrôleurs concernés.

4. Fonctionnement — Cycle de vie d'un cours

Un cours suit un workflow en plusieurs étapes, matérialisé par un champ statut en base de données. Les transitions possibles sont les suivantes :

```
draft → (upload PDF) → pending → public
                                ↘ draft (rejet admin)
```

4.1 Création d'un cours (draft)

Lorsqu'un enseignant crée un cours, celui-ci est automatiquement mis en statut draft. Le contrôleur vérifie d'abord que l'utilisateur n'est pas sous sanction active (SanctionAccessService), puis instancie le formulaire Symfony et délègue la validation à CourseValidationService.

Après la création, l'utilisateur est automatiquement redirigé vers la page d'upload du PDF, ce qui guide naturellement le flux de création.

```
// CourseCrudController.php — méthode new()
$course = new Course();
$course->setCreatedAt(new \DateTimeImmutable());
$course->setStatus('draft');
$course->setTeacher($this->getUser());

if ($form->isSubmitted()) {
   $errors = $this->validationService->validateCourseData($course,
$entityManager);

   if (!empty($errors)) {
        $this->addValidationErrors($errors);
   } elseif ($form->isValid()) {
         $entityManager->persist($course);
         $entityManager->flush();
        $this->addTranslatedFlash('success', 'create_success');
         return $this->redirectToRoute('app_course_upload', ['id' =>
$course->getId()]);
   }
}w
```

4.2 Upload du fichier PDF

Le CourseUploadController gère le dépôt et le remplacement du PDF. La validation du fichier (type MIME, taille, intégrité) est déléguée au PdfFileService. Si le cours était déjà public, son statut repasse en draft pour une nouvelle relecture.

```
// CourseUploadController.php
$validationErrors = $this->pdfService->validatePdfFile($pdf);

if (!empty($validationErrors)) {
    $this->addValidationErrors($validationErrors);
   return $this->renderWithTranslations('course/upload.html.twig',
[...]);
}

$this->pdfService->savePdfFile($pdf, $course);

if ($course->getStatus() === 'public') {
    $course->setStatus('draft'); // Retour en relecture si modification
}
$entityManager->flush();
```

4.3 Soumission pour relecture et publication

Une fois le PDF déposé, l'enseignant peut soumettre son cours. Le CourseStatusController vérifie que le cours est bien en statut draft, que les champs obligatoires sont renseignés (titre, description, PDF), et valide le token CSRF avant de passer le statut à pending. L'administrateur dispose ensuite de deux actions : publier (public) ou rejeter (retour à draft).

```
// CourseStatusController.php — méthode submit()
if ($course->getStatus() !== 'draft') {
   $this->addTranslatedFlash('error', 'cannot_modify');
   return $this->redirectToRoute('app_course_show', ['id' =>
$course->getId()]);
}

if (!$course->getTitle() || !$course->getDescription() ||
!$course->getPdfFilename()) {
   $this->addTranslatedFlash('error', 'form_invalid');
   return $this->redirectToRoute('app_course_show', ['id' =>
$course->getId()]);
}

if ($this->isCsrfTokenValid('submit' . $course->getId(),
$request->getPayload()->getString('_token'))) {
   $course->setStatus('pending');
   $entityManager->flush();
   $this->addTranslatedFlash('success', 'submit_for_review');
}
```

4.4 Édition et réinitialisation du statut

La méthode edit() du CourseCrudController capture les données originales du cours avant de traiter le formulaire. Si des champs publics (titre, description, nombre de pages, langue) ont été modifiés sur un cours public, le statut repasse automatiquement en draft pour protéger l'intégrité éditoriale.

```
// CourseCrudController.php — méthode edit()
$originalData = [
   'title'       => $course->getTitle(),
   'description' => $course->getDescription(),
    'nb_pages'   => $course->getNbPages(),
    'lang'       => $course->getLang(),
];

// [...] traitement du formulaire [...]
```

```
if ($this->hasCourseDataChanged($course, $originalData) &&
$course->getStatus() === 'public') {
   $course->setStatus('draft'); // Force une nouvelle relecture
}
```

5. Mesures de sécurité

La sécurité est prise en compte à plusieurs niveaux : contrôle d'accès, protection contre les falsifications de requêtes, isolation des données et gestion des sanctions.

5.1 Contrôle d'accès par rôles (RBAC)

Symfony Security Component est utilisé pour définir des règles d'accès à chaque route via l'attribut #[IsGranted]. Trois niveaux de rôles sont définis : ROLE_USER (enseignant authentifié), ROLE_ADMIN (modérateur), et IS_AUTHENTICATED_REMEMBERED (utilisateur connecté, persistance session).

```
// Accès réservé aux admins uniquement
#[Route('/{id}/publish', name: 'app_course_publish', methods: ['POST'])]
#[IsGranted('ROLE_ADMIN')]
public function publish(...): Response { ... }

// Accès à tout utilisateur authentifié
#[Route('/{id}/submit', name: 'app_course_submit', methods: ['POST'])]
#[IsGranted('ROLE_USER')]
public function submit(...): Response { ... }
```

5.2 Vérification de propriété (ownership)

Au-delà des rôles Symfony, chaque action sensible vérifie que l'utilisateur connecté est bien propriétaire de la ressource. Cela empêche un enseignant de modifier le cours d'un autre enseignant, même s'il dispose du bon rôle.

```
// CourseKeywordController.php
if (
   !$this->isGranted('ROLE_ADMIN') &&
   (!$this->getUser() || $course->getTeacher() !== $this->getUser())
) {
```

```
        $this->addTranslatedFlash('error', 'access_denied');
        return $this->redirectToRoute('app_course_index');
    }
```

Cette même logique est centralisée dans CoursePermissionService pour les opérations de modification et de visualisation, favorisant la réutilisabilité.

5.3 Protection CSRF

Toutes les actions destructives ou critiques (suppression, soumission, publication, rejet, gestion des mots-clés) sont protégées par un token CSRF généré et vérifié par Symfony. Le token est lié à une action précise et à l'identifiant de l'entité concernée, ce qui limite les risques de réutilisation.

```
    // CourseStatusController.php — vérification CSRF
    if ($this->isCsrfTokenValid('publish' . $course->getId(),
        $request->getPayload()->getString('_token'))) {
        $course->setStatus('public');
        $entityManager->flush();

    }
```

5.4 Système de sanctions

Un service dédié, SanctionAccessService, est injecté dans les contrôleurs concernés pour vérifier qu'un utilisateur sanctionné ne peut pas poster de contenu (création de cours, upload, commentaires). Ce système est consulté systématiquement avant toute action de création ou modification.

```
    // CourseCrudController.php — vérification sanction
    if ($this->getUser() instanceof User
        && !$sanctionAccessService->canPostContent($this->getUser())) {
        $this->addTranslatedFlash('error', 'active_sanction');
        return $this->redirectToRoute('app_course_index');
    }
```

5.5 Validation des données (double niveau)

La validation est effectuée à deux niveaux complémentaires. Côté serveur, CourseValidationService centralise les règles métier (unicité du titre, longueur des champs, etc.) indépendamment du formulaire Symfony. Les mots-clés font l'objet d'une validation renforcée : longueur maximale, caractères autorisés via regex, limite de 10 mots-clés par cours, et vérification d'appartenance au bon enseignant.

```
// CourseKeywordController.php — validation des nouveaux mots-clés
foreach ($newKeywordsList as $keyword) {
   if (strlen($keyword) > 16) {
         $this->addTranslatedFlash('error', 'keywords_too_long', [
         '%keyword%' => $keyword,
         '%length%' => strlen($keyword),
         ]);
         return $this->redirectToRoute('app_course_keywords_manage',
$manageRoute);
   }

   if (!preg_match('/^[a-zA-Z0-9À-ſ\-_]+$/', $keyword)) {
         $this->addTranslatedFlash('error', 'keywords_invalid_chars',
['%keyword%' => $keyword]);
         return $this->redirectToRoute('app_course_keywords_manage',
$manageRoute);
   }
}
```

5.6 Isolation des données entre enseignants

Lors de l'association de mots-clés existants à un cours, le système vérifie que le mot-clé appartient bien à l'enseignant du cours et qu'il partage la même langue. Cela prévient toute tentative de cross-contamination des données entre utilisateurs.

```
// CourseKeywordController.php
if (
   $keyword->getTeacher() !== $course->getTeacher() ||
   $keyword->getLang() !== $course->getLang()
) {
   $this->addTranslatedFlash('error', 'keywords_not_allowed',
         ['%keyword%' => $keyword->getKeyword()]);
   return $this->redirectToRoute('app_course_keywords_manage',
$manageRoute);
}
```

5.7 Gestion des erreurs base de données

Les blocs try/catch distinguent les violations de contrainte d'unicité (UniqueConstraintViolationException) des autres erreurs Doctrine, afin de fournir un message d'erreur précis à l'utilisateur sans exposer les détails techniques internes.

```
// CourseCrudController.php
try {
   $entityManager->persist($course);
   $entityManager->flush();
   $this->addTranslatedFlash('success', 'create_success');
} catch (\Doctrine\DBAL\Exception\UniqueConstraintViolationException $e)
{
   $this->addTranslatedFlash('error', 'title_exists');
} catch (\Doctrine\DBAL\Exception $e) {
   $this->addTranslatedFlash('error', 'database_error');
}
```

6. Internationalisation - Traductions

Tous les messages affichés à l'utilisateur (flash messages de succès, d'erreur, de validation) passent par un système de traduction centralisé. La méthode addTranslatedFlash(), définie dans le BaseController, prend une clé de traduction et des paramètres optionnels, puis délègue au TranslationService. Les fichiers de traduction spécifiques au domaine cours sont déclarés via la propriété translationFiles dans chaque contrôleur, ce qui permet un chargement ciblé des catalogues de traductions.

```
// Déclaration dans chaque contrôleur
protected array $translationFiles = ['course'];

// Utilisation typique
$this->addTranslatedFlash('success', 'create_success');
$this->addTranslatedFlash('error', 'keywords_too_long', [
   '%keyword%' => $keyword,
   '%length%' => strlen($keyword),
]);

return $this->renderWithTranslations('course/index.html.twig');
```

Cette approche permet d'adapter facilement l'interface à plusieurs langues sans modifier le code des contrôleurs, conformément au principe d'ouverture/fermeture (Open/Closed Principle).

7. Gestion des commentaires

Le CourseCommentController gère l'ensemble du CRUD des commentaires. Plusieurs règles métier sont appliquées :

- Un utilisateur ne peut déposer qu'un seul commentaire par cours (vérification via findOneByUserAndCourse).
- Seul l'auteur du commentaire ou un administrateur peut le modifier ou le supprimer.
- Les utilisateurs sous sanction ne peuvent pas commenter.
- Le formulaire de commentaire n'est affiché sur la page du cours que pour les utilisateurs authentifiés (IS_AUTHENTICATED_REMEMBERED).

```
// CourseCommentController.php — unicité du commentaire
$existingComment = $commentRepo->findOneByUserAndCourse($user, $course);

if ($existingComment) {
   $this->addTranslatedFlash('error', 'course_comment_already_exists');
   return $this->redirectToRoute('app_course_show', ['id' =>
$courseId]);
}
```

La cohérence de l'appartenance du commentaire à son cours est également vérifiée lors de l'édition et de la suppression, en comparant l'identifiant du cours stocké en base avec celui passé dans l'URL, ce qui prévient toute manipulation de paramètre.

8. Bilan et points forts du projet

Points forts

- Architecture modulaire : chaque contrôleur a une responsabilité unique et délègue la logique métier à des services dédiés.
- Sécurité multicouche : RBAC, ownership, CSRF, sanctions, validation serveur.
- Workflow éditorial clair : le système de statuts guide le contenu de sa création à sa publication.
- Internationalisation intégrée dès la conception, sans coupler les messages à la logique applicative.
- Gestion des erreurs différenciées : distinction entre erreurs métier et erreurs techniques, sans exposition d'informations sensibles.

Pistes d'amélioration possibles

- Extraction des transitions de statut vers un composant Workflow Symfony pour rendre le cycle de vie explicite et extensible.
- Ajout d'un système de notifications (email) lors des changements de statut (soumission, publication, rejet).
- Pagination et filtrage côté API (JSON) pour les futurs clients mobiles ou SPA.