$category->setImage($imageName);  
 } else {  
 $category->setImage($prevImageName);  
 }  
 $this->em->flush();  
 }  
}

<?php  
  
  
namespace App\Service;  
  
  
use App\Entity\Comments;  
use App\Entity\Parts;  
use App\Entity\User;  
use App\Repository\PartsRepository;  
use DateTimeImmutable;  
use Doctrine\ORM\EntityManagerInterface;  
use JetBrains\PhpStorm\Pure;  
  
class CommentsService  
{  
  
 private EntityManagerInterface $em;  
 private PartsRepository $partsRepository;  
  
 */\*\*  
 \* FanficService constructor.  
 \* @param EntityManagerInterface $em  
 \* @param PartsRepository $partsRepository  
 \*/* #[Pure] public function \_\_construct  
 (  
 EntityManagerInterface $em,  
 PartsRepository $partsRepository,  
 )  
 {  
 $this->em = $em;  
 $this->partsRepository = $partsRepository;  
 }  
  
 public function createComment(string $text, int $id, User $user)  
 {  
 $comment = new Comments();  
 $parts = $this->partsRepository->findOneBy(['id' => $id]);  
 $comment->setText($text);  
 $comment->setCreatedAt(new DateTimeImmutable());  
 $comment->addPart($parts);  
 $comment->setUser($user);  
 $this->em->persist($comment);  
 $this->em->flush();  
 }  
  
 public function deleteComment(Comments $comment)  
 {  
 $this->em->remove($comment);  
 $this->em->flush();  
 }  
  
  
}

<?php  
  
  
namespace App\Service;  
  
use App\Entity\BaseEntity;  
use App\Entity\Fandom;  
use Doctrine\ORM\EntityManagerInterface;  
use Exception;  
use JetBrains\PhpStorm\Pure;  
use Symfony\Component\Form\FormInterface;  
  
class FandomService extends BaseEntityService  
{  
  
 private EntityManagerInterface $em;  
 private FileUploader $uploader;  
  
 */\*\*  
 \* FanficService constructor.  
 \* @param EntityManagerInterface $em  
 \* @param FileUploader $uploader  
 \*/* #[Pure] public function \_\_construct  
 (  
 EntityManagerInterface $em,  
 FileUploader $uploader  
 )  
 {  
 parent::*\_\_construct*($em);  
 $this->em = $em;  
 $this->uploader = $uploader;  
 }  
  
 */\*\*  
 \* @throws Exception  
 \*/* public function isFandomInstance(BaseEntity $baseEntity): bool  
 {  
 if ($baseEntity instanceof Fandom) {  
 return true;  
 }  
 throw new Exception('Entity is not instanceof Fandom');  
 }  
  
 */\*\*  
 \* @param FormInterface $form  
 \* @throws Exception  
 \*/* public function create(FormInterface $form): void  
 {  
 $fandom = $form->getData();  
 if ($this->isFandomInstance($fandom)) {  
 $image = $form->get('image')->getData();  
 if ($image) {  
 $imageName = $this->uploader->upload($image);  
 */\*\* @var Fandom $fandom \*/* $fandom->setImage($imageName);  
 }  
 $this->em->persist($fandom);  
 $this->em->flush();  
 }  
 }  
  
 */\*\*  
 \* @param string $prevImageName  
 \* @param FormInterface $form  
 \* @throws Exception  
 \*/* public function updateWithImage(string $prevImageName, FormInterface $form): void  
 {  
 */\* @var Fandom $fandom \*/* $fandom = $form->getData();  
 if ($this->isFandomInstance($fandom)) {  
 $image = $form->get('image')->getData();  
 if ($image) {  
 if ($prevImageName) {  
 $this->uploader->removeImage($prevImageName);  
 }  
 $imageName = $this->uploader->upload($image);  
 $fandom->setImage($imageName);  
 } else {  
 $fandom->setImage($prevImageName);  
 }  
 $this->em->flush();  
 }  
 }  
  
 */\*\*  
 \* @param Fandom|BaseEntity $fandom  
 \* @throws Exception  
 \*/* public function delete(Fandom|BaseEntity $fandom): void  
 {  
 if ($this->isFandomInstance($fandom)) {  
 if ($fandom->getImage()) {  
 $this->uploader->removeImage($fandom->getImage());  
 }  
 $this->em->remove($fandom);  
 $this->em->flush();  
 }  
 }  
}

<?php  
  
namespace App\Service;  
  
use Symfony\Component\Filesystem\Exception\IOException;  
use Symfony\Component\Filesystem\Exception\IOExceptionInterface;  
use Symfony\Component\Filesystem\Filesystem;  
use Symfony\Component\HttpFoundation\File\Exception\FileException;  
use Symfony\Component\HttpFoundation\File\File;  
use Symfony\Component\HttpFoundation\File\UploadedFile;  
use Symfony\Component\String\Slugger\SluggerInterface;  
  
class FileUploader  
{  
 private string $targetDirectory;  
 private SluggerInterface $slugger;  
  
 public function \_\_construct($targetDirectory, SluggerInterface $slugger)  
 {  
 $this->targetDirectory = $targetDirectory;  
 $this->slugger = $slugger;  
 }  
  
 public function upload(UploadedFile $file): string  
 {  
 $originalFilename = *pathinfo*($file->getClientOriginalName(), *PATHINFO\_FILENAME*);  
 $safeFilename = $this->slugger->slug($originalFilename);  
 $fileName = $safeFilename . '-' . *uniqid*() . '.' . $file->guessExtension();  
  
 $file->move($this->getTargetDirectory(), $fileName);  
 return $fileName;  
 }  
  
 public function getTargetDirectory(): string  
 {  
 return $this->targetDirectory;  
 }  
  
 public function removeImage(string $fileName)  
 {  
 $fileSystem = new Filesystem();  
 $fileImagePath = $this->getTargetDirectory() . $fileName;  
 try {  
 $fileSystem->remove(new File($fileImagePath));  
 } catch (IOExceptionInterface $exception) {  
 dd('error');  
 }  
 }  
}

<?php  
  
  
namespace App\Service;  
  
  
use App\Entity\Comments;  
use App\Entity\Likes;  
use App\Entity\Parts;  
use App\Entity\User;  
use App\Repository\LikesRepository;  
use App\Repository\PartsRepository;  
use App\Repository\StoryRepository;  
use DateTimeImmutable;  
use Doctrine\ORM\EntityManagerInterface;  
use JetBrains\PhpStorm\Pure;  
  
class LikesService  
{  
  
 private EntityManagerInterface $em;  
 private StoryRepository $storyRepository;  
 private LikesRepository $likesRepository;  
  
 */\*\*  
 \* FanficService constructor.  
 \* @param EntityManagerInterface $em  
 \* @param StoryRepository $storyRepository  
 \* @param LikesRepository $likesRepository  
 \*/* #[Pure] public function \_\_construct  
 (  
 EntityManagerInterface $em,  
 StoryRepository $storyRepository,  
 LikesRepository $likesRepository,  
 )  
 {  
 $this->em = $em;  
 $this->storyRepository = $storyRepository;  
 $this->likesRepository = $likesRepository;  
 }  
  
 public function addLike(int $storyId, User $user)  
 {  
  
 $story = $this->storyRepository->findOneBy(['id' => $storyId]);  
 $like = $this->likesRepository->findOneBy(['user' => $user]);  
 if ($like === null) {  
 $like = new Likes();  
 $like->setUser($user);  
 $like->addStory($story);  
 $this->em->persist($like);  
 $this->em->flush();  
 }else{  
 $this->em->remove($like);  
 $this->em->flush();  
 }  
 return $story->getLikes()->count();  
  
 }  
  
 public function deleteLike(Comments $comment)  
 {  
 $this->em->remove($comment);  
 $this->em->flush();  
 }  
  
  
}

<?php  
  
namespace App\Service;  
  
use App\Entity\Category;  
use App\Entity\User;  
use App\Interfaces\CategoryServiceInterface;  
use App\Interfaces\MailServiceInterface;  
use App\Repository\UserRepository;  
use App\Security\EmailVerifier;  
use Doctrine\ORM\EntityManagerInterface;  
use Symfony\Bridge\Twig\Mime\TemplatedEmail;  
use Symfony\Component\Mailer\Exception\TransportExceptionInterface;  
use Symfony\Component\Mime\Address;  
  
class MailService implements MailServiceInterface  
{  
  
 private EntityManagerInterface $em;  
 private UserRepository $userRepository;  
  
 */\*\*  
 \* FanficService constructor.  
 \* @param EntityManagerInterface $em  
 \* @param UserRepository $userRepository  
 \*/* public function \_\_construct(  
 EntityManagerInterface $em,  
 UserRepository $userRepository,  
 )  
 {  
 $this->em = $em;  
 $this->userRepository = $userRepository;  
 }  
  
  
 */\*\*  
 \* @throws TransportExceptionInterface  
 \*/* public function sendEmail(User $user, EmailVerifier $emailVerifier): void  
 {  
 $emailVerifier->sendEmailConfirmation('app\_verify\_email', $user,  
 (new TemplatedEmail())  
 ->from(new Address('sendercourse@gmail.com', 'Sender mail bot'))  
 ->to($user->getEmail())  
 ->subject('Please Confirm your Email')  
 ->htmlTemplate('registration/confirmation\_email.html.twig')  
 );  
 }  
  
 public function emailConfirm(int $id): void  
 {  
 $user = $this->userRepository->find($id);  
 if ($user !== null) {  
 $user->setEmailIsVerified(true);  
 $this->em->persist($user);  
 $this->em->flush();  
 }  
 }  
}

<?php  
  
namespace App\Service;  
  
  
use App\Entity\BaseEntity;  
use App\Entity\Parts;  
use App\Entity\Story;  
use App\Entity\StoryParts;  
use App\Interfaces\PartsServiceInterface;  
use App\Repository\StoryPartsRepository;  
use Doctrine\ORM\EntityManagerInterface;  
use Doctrine\ORM\NonUniqueResultException;  
use Symfony\Component\Form\FormInterface;  
  
class PartService extends BaseEntityService implements PartsServiceInterface  
{  
  
 private EntityManagerInterface $em;  
 private StoryPartsRepository $storyPartsRepository;  
  
 */\*\*  
 \* FanficService constructor.  
 \* @param EntityManagerInterface $em  
 \* @param StoryPartsRepository $storyPartsRepository  
 \*/* public function \_\_construct  
 (  
 EntityManagerInterface $em,  
 StoryPartsRepository $storyPartsRepository,  
 )  
 {  
 $this->em = $em;  
 $this->storyPartsRepository = $storyPartsRepository;  
 }  
  
 public function createPart(Story $story, Parts $parts): void  
 {  
 $storyPart = new StoryParts();  
 $storyPart->setStory($story);  
 $storyPart->setParts($parts);  
 $story->addStoryPart($storyPart);  
 $this->em->persist($storyPart);  
 $this->em->persist($parts);  
 $this->em->flush();  
 }  
  
 */\*\*  
 \* @param Parts $parts  
 \*/* public function editPart(Parts $parts): void  
 {  
 $this->em->flush();  
 }  
  
 */\*\*  
 \* @param Parts|BaseEntity $parts  
 \* @throws NonUniqueResultException  
 \*/* public function delete(Parts|BaseEntity $parts): void  
 {  
 $storyParts = $this->storyPartsRepository->findByPartsId($parts->getId());  
 $this->em->remove($storyParts);  
 $this->em->remove($parts);  
 $this->em->flush();  
 }  
  
 */\*\*  
 \* @throws NonUniqueResultException  
 \*/* public function reorderParts(mixed $jsonData, Story $story): void  
 {  
 foreach ($jsonData as $key => $value) {  
 $part\_id = *json\_decode*($value)->part;  
 $story\_part = $this->storyPartsRepository->findByStoryIdAndByPartId($story->getId(), $part\_id);  
 $story\_part->setOrderNumber($key);  
 $this->em->persist($story\_part);  
  
 }  
 $this->em->flush();  
 }  
}

<?php  
  
  
namespace App\Service;  
  
use App\Entity\BaseEntity;  
use App\Entity\Category;  
use App\Entity\Fandom;  
use App\Entity\Review;  
use App\Entity\Story;  
use App\Entity\User;  
use DateTimeImmutable;  
use Doctrine\ORM\EntityManagerInterface;  
use Exception;  
use JetBrains\PhpStorm\Pure;  
use Symfony\Component\Form\FormInterface;  
  
class ReviewsService  
{  
  
 private EntityManagerInterface $em;  
  
 */\*\*  
 \* FanficService constructor.  
 \* @param EntityManagerInterface $em  
 \*/* #[Pure] public function \_\_construct  
 (  
 EntityManagerInterface $em,  
 )  
 {  
 $this->em = $em;  
 }  
  
 public function createReview(Review $review, Story $story, User $user)  
 {  
 $review->setCreatedAt(new DateTimeImmutable());  
 $review->addStory($story);  
 $review->setUser($user);  
 $this->em->persist($review);  
 $this->em->flush();  
 }  
  
 public function deleteReview(Review $review)  
 {  
 $this->em->remove($review);  
 $this->em->flush();  
 }  
  
  
}

<?php  
  
namespace App\Service;  
  
use App\Entity\BaseEntity;  
use App\Entity\Story;  
use App\Entity\User;  
use Doctrine\ORM\EntityManagerInterface;  
use Exception;  
use Symfony\Component\Form\FormInterface;  
use Symfony\Component\Security\Core\User\UserInterface;  
  
class StoryService extends BaseEntityService  
{  
  
 private EntityManagerInterface $em;  
 private User $user;  
 private FileUploader $uploader;  
  
 */\*\*  
 \* FanficService constructor.  
 \* @param EntityManagerInterface $em  
 \* @param FileUploader $uploader  
 \*/* public function \_\_construct  
 (  
 EntityManagerInterface $em,  
 FileUploader $uploader,  
 )  
 {  
 $this->em = $em;  
 $this->uploader = $uploader;  
 }  
  
 */\*\*  
 \* @throws Exception  
 \*/* public function isStoryInstance(BaseEntity $baseEntity): bool  
 {  
 if (!$baseEntity instanceof Story) {  
 throw new Exception('Entity is not instanceof Story');  
 }  
 return true;  
 }  
  
 */\*\*  
 \* @param User|UserInterface $user  
 \*/* public function setUser(UserInterface|User $user): void  
 {  
 $this->user = $user;  
 }  
  
 */\*\*  
 \* @param FormInterface $form  
 \* @throws Exception  
 \*/* public function create(FormInterface $form): void  
 {  
 $story = $form->getData();  
 if ($this->isStoryInstance($story)) {  
 $story->setAuthor($this->user);  
 }  
 $this->em->persist($story);  
 $this->em->flush();  
 }  
  
 */\*\*  
 \* @param FormInterface $form  
 \*/* public function update(FormInterface $form): void  
 {  
  
 $this->em->flush();  
 }  
  
 */\*\*  
 \* @param Story|BaseEntity $story  
 \*/* public function delete(Story|BaseEntity $story): void  
 {  
 $this->em->remove($story);  
 $this->em->flush();  
 }  
}

<?php  
  
  
namespace App\Service;  
  
use App\Entity\BaseEntity;  
use App\Entity\User;  
use App\Interfaces\UserServiceInterface;  
use App\Utils\Base64FileExtractor;  
use App\Utils\UploadedBase64File;  
use Doctrine\ORM\EntityManagerInterface;  
use Exception;  
use JetBrains\PhpStorm\Pure;  
use Symfony\Component\Form\FormInterface;  
use Symfony\Component\PasswordHasher\Hasher\UserPasswordHasherInterface;  
  
class UserService extends BaseEntityService implements UserServiceInterface  
{  
  
 private EntityManagerInterface $em;  
 private UserPasswordHasherInterface $passwordHasher;  
 private FileUploader $uploader;  
 private Base64FileExtractor $base64FileExtractor;  
  
 */\*\*  
 \* FanficService constructor.  
 \* @param EntityManagerInterface $em  
 \* @param UserPasswordHasherInterface $passwordHasher  
 \* @param FileUploader $uploader  
 \* @param UploadedBase64File $file  
 \*/* #[Pure] public function \_\_construct(  
 EntityManagerInterface $em,  
 UserPasswordHasherInterface $passwordHasher,  
 FileUploader $uploader,  
 Base64FileExtractor $base64FileExtractor,  
 )  
 {  
 parent::*\_\_construct*($em);  
 $this->em = $em;  
 $this->passwordHasher = $passwordHasher;  
 $this->uploader = $uploader;  
 $this->base64FileExtractor = $base64FileExtractor;  
 }  
  
 */\*\*  
 \* @throws Exception  
 \*/* public function isUserInstance(BaseEntity $baseEntity): bool  
 {  
 if (!$baseEntity instanceof User) {  
 throw new Exception('Entity is not instanceof User');  
 }  
 return true;  
 }  
  
 */\*\*  
 \* @param FormInterface $form  
 \* @throws Exception  
 \*/* public function create(FormInterface $form): void  
 {  
 $user = $form->getData();  
 if ($this->isUserInstance($user)) {  
 $user->setPassword($this->passwordHasher->hashPassword($user, $user->getPassword()));  
 $this->em->persist($user);  
 $this->em->flush();  
 }  
 }  
  
 */\*\*  
 \* @param FormInterface $form  
 \* @throws Exception  
 \*/* public function update(FormInterface $form): void  
 {  
 $user = $form->getData();  
 if ($this->isUserInstance($user)) {  
 $user->setPassword($this->passwordHasher->hashPassword($user, $user->getPassword()));  
 $this->em->flush();  
 }  
 }  
  
 */\*\*  
 \* @param User|BaseEntity $user  
 \* @throws Exception  
 \*/* public function delete(User|BaseEntity $user): void  
 {  
 if ($this->isUserInstance($user)) {  
 $this->em->remove($user);  
 $this->em->flush();  
 }  
 }  
  
 public function updateAvatarAjax($file, User $user)  
 {  
 $prevImageName = $user->getImage();  
 $base64Image = $this->base64FileExtractor->extractBase64String($file);  
 $imageFile = new UploadedBase64File($base64Image, "blabla");  
  
  
  
  
 $imageName = $this->uploader->upload($imageFile);  
 $user->setImage($imageName);  
 $this->em->flush();  
 }  
  
 public function updateWithImage(string $prevImageName, FormInterface $form): void  
 {  
 */\* @var User $user \*/* $user = $form->getData();  
 if ($this->isUserInstance($user)) {  
 $image = $form->get('image')->getData();  
 if ($image) {  
 if ($prevImageName) {  
 $this->uploader->removeImage($prevImageName);  
 }  
 $imageName = $this->uploader->upload($image);  
 $user->setImage($imageName);  
 } else {  
 $user->setImage($prevImageName);  
 }  
 $this->em->flush();  
 }  
 }  
}

<?php  
  
  
namespace App\Voter;  
  
  
use App\Entity\User;  
use App\Service\Constants;  
use Symfony\Component\Security\Core\Authentication\Token\TokenInterface;  
use Symfony\Component\Security\Core\Authorization\Voter\Voter;  
use Symfony\Component\Security\Core\Security;  
use Symfony\Component\Security\Core\User\UserInterface;  
  
  
class AdminVoter extends Voter  
{  
 private Security $security;  
 public const *VIEW* = 'VIEW';  
  
 public function \_\_construct(Security $security)  
 {  
 $this->security = $security;  
 }  
  
 protected function supports(string $attribute, $subject): bool  
 {  
 return $subject instanceof User && *in\_array*(  
 $attribute,  
 [self::*VIEW*],  
 true);  
 }  
  
 protected function voteOnAttribute($attribute, $subject, TokenInterface $token): bool  
 {  
 $user = $this->security->getUser();  
 if ($user === null) {  
 return false;  
 }  
 return match ($attribute) {  
 self::*VIEW* => $this->canView(),  
 default => false,  
 };  
 }  
  
 private function canView(): bool  
 {  
 return $this->isAdmin();  
 }  
  
 private function isAdmin(): bool  
 {  
 return $this->security->isGranted(Constants::*ROLE\_ADMIN*);  
 }  
  
  
}

<?php  
  
namespace App\Controller;  
  
use App\Entity\Story;  
use App\Repository\CategoryRepository;  
use App\Repository\CharacterRepository;  
use App\Repository\FandomRepository;  
use App\Repository\MpaaRatingRepository;  
use App\Repository\StatusRepository;  
use App\Repository\StoryRepository;  
use App\Repository\TagRepository;  
use App\Repository\UserRepository;  
use App\Voter\AdminVoter;  
use Knp\Component\Pager\PaginatorInterface;  
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;  
use Symfony\Component\HttpFoundation\Request;  
use Symfony\Component\HttpFoundation\Response;  
use Symfony\Component\Routing\Annotation\Route;  
  
class AdminController extends AbstractController  
{  
 private UserRepository $userRepository;  
 private FandomRepository $fandomRepository;  
 private CategoryRepository $categoryRepository;  
 private StatusRepository $statusRepository;  
 private TagRepository $tagRepository;  
 private CharacterRepository $characterRepository;  
 private StoryRepository $storyRepository;  
 private MpaaRatingRepository $mpaaRatingRepository;  
 private PaginatorInterface $paginator;  
  
 public function \_\_construct  
 (  
 UserRepository $userRepository,  
 FandomRepository $fandomRepository,  
 CategoryRepository $categoryRepository,  
 StatusRepository $statusRepository,  
 TagRepository $tagRepository,  
 CharacterRepository $characterRepository,  
 StoryRepository $storyRepository,  
 MpaaRatingRepository $mpaaRatingRepository,  
 PaginatorInterface $paginator,  
 )  
 {  
 $this->userRepository = $userRepository;  
 $this->fandomRepository = $fandomRepository;  
 $this->categoryRepository = $categoryRepository;  
 $this->statusRepository = $statusRepository;  
 $this->tagRepository = $tagRepository;  
 $this->characterRepository = $characterRepository;  
 $this->storyRepository = $storyRepository;  
 $this->mpaaRatingRepository = $mpaaRatingRepository;  
 $this->paginator = $paginator;  
 }  
  
  
 #[Route('/admin/user', name: 'admin\_user')]  
 public function indexUser(Request $request): Response  
 {  
 $this->denyAccessUnlessGranted(AdminVoter::VIEW, $this->getUser(), "Access Denied. Only for Admins");  
 $query = $this->userRepository->queryFindAll();  
 $pagination = $this->paginator->paginate(  
 $query,  
 $request->query->getInt('page', 1),  
 $request->query->getInt('limit', 7),  
 );  
  
 return $this->render('admin/indexUser.html.twig', [  
 'pagination' => $pagination  
 ]);  
 }  
  
 #[Route('/admin/fandom', name: 'admin\_fandom')]  
 public function indexFandom(Request $request): Response  
 {  
 $this->denyAccessUnlessGranted(AdminVoter::VIEW, $this->getUser(), "Access Denied. Only for Admins");  
 $query = $this->fandomRepository->queryFindAll();  
 $pagination = $this->paginator->paginate(  
 $query,  
 $request->query->getInt('page', 1),  
 $request->query->getInt('limit', 7),  
 );  
  
 return $this->render('admin/indexFandom.html.twig', [  
 'pagination' => $pagination  
 ]);  
 }  
  
 #[Route('/admin/category', name: 'admin\_category')]  
 public function indexCategory(Request $request): Response  
 {  
 $this->denyAccessUnlessGranted(AdminVoter::VIEW, $this->getUser(), "Access Denied. Only for Admins");  
 $query = $this->categoryRepository->queryFindAll();  
 $pagination = $this->paginator->paginate(  
 $query,  
 $request->query->getInt('page', 1),  
 $request->query->getInt('limit', 7),  
 );  
  
 return $this->render('admin/indexCategory.html.twig', [  
 'pagination' => $pagination,  
 ]);  
 }  
  
 #[Route('/admin/status', name: 'admin\_status')]  
 public function indexStatuses(Request $request): Response  
 {  
 $this->denyAccessUnlessGranted(AdminVoter::VIEW, $this->getUser(), "Access Denied. Only for Admins");  
 $query = $this->statusRepository->queryFindAll();  
 $pagination = $this->paginator->paginate(  
 $query,  
 $request->query->getInt('page', 1),  
 $request->query->getInt('limit', 7),  
 );  
  
 return $this->render('admin/indexStatus.html.twig', [  
 'pagination' => $pagination,  
 ]);  
 }  
  
 #[Route('/admin/tag', name: 'admin\_tag')]  
 public function indexTags(Request $request): Response  
 {  
 $this->denyAccessUnlessGranted(AdminVoter::VIEW, $this->getUser(), "Access Denied. Only for Admins");  
 $query = $this->tagRepository->queryFindAll();  
 $pagination = $this->paginator->paginate(  
 $query,  
 $request->query->getInt('page', 1),  
 $request->query->getInt('limit', 7),  
 );  
  
 return $this->render('admin/indexTag.html.twig', [  
 'pagination' => $pagination,  
 ]);  
 }  
  
 #[Route('/admin/character', name: 'admin\_character')]  
 public function indexCharacter(Request $request): Response  
 {  
 $this->denyAccessUnlessGranted(AdminVoter::VIEW, $this->getUser(), "Access Denied. Only for Admins");  
 $query = $this->characterRepository->queryFindAll();  
 $pagination = $this->paginator->paginate(  
 $query,  
 $request->query->getInt('page', 1),  
 $request->query->getInt('limit', 7),  
 );  
  
 return $this->render('admin/indexCharacter.html.twig', [  
 'pagination' => $pagination,  
 ]);  
 }  
  
 #[Route('/admin/mpaaRating', name: 'admin\_mpaaRating')]  
 public function indexMpaaRating(Request $request): Response  
 {  
 $this->denyAccessUnlessGranted(AdminVoter::*VIEW*, $this->getUser(), "Access Denied. Only for Admins");  
 $query = $this->mpaaRatingRepository->queryFindAll();  
 $pagination = $this->paginator->paginate(  
 $query,  
 $request->query->getInt('page', 1),  
 $request->query->getInt('limit', 7),  
 );  
 return $this->render('admin/indexMpaaRating.html.twig', [  
 'pagination' => $pagination,  
 ]);  
 }  
  
 #[Route('/admin/story', name: 'admin\_story')]  
 public function indexStory(Request $request): Response  
 {  
 $this->denyAccessUnlessGranted(AdminVoter::VIEW, $this->getUser(), "Access Denied. Only for Admins");  
 $query = $this->storyRepository->queryFindAll()->getQuery();  
 $pagination = $this->paginator->paginate(  
 $query,  
 $request->query->getInt('page', 1),  
 $request->query->getInt('limit', 7),  
 );  
 return $this->render('admin/indexStory.html.twig', [  
 'pagination' => $pagination,  
 ]);  
 }  
}

<?php  
  
namespace App\Controller;  
  
use App\Repository\UserRepository;  
use Knp\Component\Pager\Paginator;  
use Knp\Component\Pager\PaginatorInterface;  
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;  
use Symfony\Component\HttpFoundation\Request;  
use Symfony\Component\HttpFoundation\Response;  
use Symfony\Component\Routing\Annotation\Route;  
  
class AuthorController extends AbstractController  
{  
 private UserRepository $userRepository;  
 private PaginatorInterface $paginator;  
  
 public function \_\_construct  
 (  
 UserRepository $userRepository,  
 PaginatorInterface $paginator  
 )  
 {  
 $this->userRepository = $userRepository;  
 $this->paginator = $paginator;  
 }  
  
 #[Route('/author', name: 'author')]  
 public function index(Request $request): Response  
 {  
 $query = $this->userRepository->queryFindAll();  
 $pagination = $this->paginator->paginate(  
 $query,  
 $request->query->getInt('page', 1),  
 $request->query->getInt('limit', 7),  
 );  
  
 return $this->render('author/index.html.twig', [  
 'pagination' => $pagination  
 ]);  
 }  
}

<?php  
  
namespace App\Controller;  
  
use App\Entity\BaseEntity;  
use App\Interfaces\BaseEntityServiceInterface;  
use App\Service\BaseEntityService;  
use App\Voter\AdminVoter;  
use Exception;  
use Knp\Component\Pager\PaginatorInterface;  
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;  
use Symfony\Component\Form\AbstractType;  
use Symfony\Component\Form\Extension\Core\Type\SubmitType;  
use Symfony\Component\Form\FormInterface;  
use Symfony\Component\HttpFoundation\RedirectResponse;  
use Symfony\Component\HttpFoundation\Request;  
use Symfony\Component\HttpFoundation\Response;  
  
  
class BaseController extends AbstractController  
{  
 private BaseEntityServiceInterface $baseEntityService;  
 private PaginatorInterface $paginator;  
  
 public function \_\_construct  
 (  
 BaseEntityService $baseEntityService,  
 PaginatorInterface $paginator,  
 )  
 {  
 $this->baseEntityService = $baseEntityService;  
 $this->paginator = $paginator;  
 }  
  
 public function setService(BaseEntityServiceInterface $baseEntityService)  
 {  
 $this->baseEntityService = $baseEntityService;  
 }  
  
 public function create(Request $request, string $formType, string $redirectRoute, string $template): Response  
 {  
 $this->denyAccessUnlessGranted(AdminVoter::VIEW, $this->getUser(), "Access Denied. Only for Admins");  
 $form = $this->createForm($formType);  
 $form->handleRequest($request);  
 if ($form->isSubmitted() && $form->isValid()) {  
 $this->baseEntityService->create($form);  
 return new RedirectResponse($this->generateUrl($redirectRoute));  
 }  
 return $this->render($template, [  
 'form' => $form->createView(),  
 ]);  
 }  
  
 public function update(Request $request, string $formType, BaseEntity $entity, string $redirectRoute, string $template): Response  
 {  
 $this->denyAccessUnlessGranted(AdminVoter::*VIEW*, $this->getUser(), "Access Denied. Only for Admins");  
 $form = $this->createForm($formType, $entity);  
 $form->handleRequest($request);  
 if ($form->isSubmitted() && $form->isValid()) {  
 $this->baseEntityService->update($form);  
 return new RedirectResponse($this->generateUrl($redirectRoute));  
 }  
 return $this->render($template, [  
 'form' => $form->createView()  
 ]);  
 }  
  
 public function showWithPagination(Request $request, $query, string $template, ?int $page = 1, ?int $limit = 10): Response  
 {  
 $pagination = $this->paginator->paginate(  
 $query,  
 $request->query->getInt('page', $page),  
 $request->query->getInt('limit', $limit),  
 );  
  
 return $this->render($template, [  
 'stories' => $pagination,  
 ]);  
 }  
  
  
 public function delete(BaseEntity $baseEntity, string $redirectRoute): Response  
 {  
 $this->denyAccessUnlessGranted(AdminVoter::VIEW, $this->getUser(), "Access Denied. Only for Admins");  
 $this->baseEntityService->delete($baseEntity);  
 return new RedirectResponse($this->generateUrl($redirectRoute));  
 }  
  
 */\*\*  
 \* @throws Exception  
 \*/* public function deleteWithConfirmation  
 (  
 Request $request,  
 BaseEntity $entity,  
 string $successRedirect,  
 string $cancelRedirect,  
 ): Response  
 {  
 $DELETE = 'DELETE';  
 $CANCEL = 'CANCEL';  
  
 $this->denyAccessUnlessGranted(AdminVoter::VIEW, $this->getUser(), "Access Denied. Only for Admins");  
  
 $form = $this->createFormBuilder()  
 ->add($DELETE, SubmitType::class, ['label' => 'Delete'])  
 ->add($CANCEL, SubmitType::class, ['label' => 'Cancel'])  
 ->getForm();  
  
 $form->handleRequest($request);  
  
 if ($form->isSubmitted() && $form->isValid()) {  
 $nextAction = $form->get($DELETE)->isClicked()  
 ? $DELETE  
 : $CANCEL;  
  
 switch ($nextAction) {  
 case $DELETE:  
 {  
 $this->baseEntityService->delete($entity);  
 return new RedirectResponse($this->generateUrl($successRedirect));  
 }  
 case $CANCEL:  
 {  
 return new RedirectResponse($this->generateUrl($cancelRedirect));  
 }  
 default:  
 throw new Exception('Unexpected value');  
 }  
 }  
  
 return $this->render('layouts/deleteConfirmation.html.twig', [  
 'form' => $form->createView(),  
 'deleteButton' => $DELETE,  
 'cancelButton' => $CANCEL,  
 ]);  
 }  
  
}

<?php  
  
namespace App\Controller;  
  
use App\Entity\Category;  
use App\Form\CategoryType;  
use App\Interfaces\CategoryServiceInterface;  
use App\Repository\CategoryRepository;  
use App\Repository\StoryRepository;  
use App\Service\CategoryService;  
use App\Voter\AdminVoter;  
use Doctrine\ORM\EntityManagerInterface;  
use Exception;  
use Knp\Component\Pager\PaginatorInterface;  
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;  
use Symfony\Component\HttpFoundation\File\Exception\FileException;  
use Symfony\Component\HttpFoundation\RedirectResponse;  
use Symfony\Component\HttpFoundation\Request;  
use Symfony\Component\HttpFoundation\Response;  
use Symfony\Component\Routing\Annotation\Route;  
use Symfony\Component\String\Slugger\SluggerInterface;  
  
class CategoryController extends AbstractController  
{  
 private BaseController $baseController;  
 private CategoryService $categoryService;  
 private CategoryRepository $categoryRepository;  
 private PaginatorInterface $paginator;  
  
 public function \_\_construct  
 (  
 BaseController $baseController,  
 CategoryService $categoryService,  
 CategoryRepository $categoryRepository,  
 PaginatorInterface $paginator,  
 )  
 {  
 $this->baseController = $baseController;  
 $this->categoryService = $categoryService;  
 $this->baseController->setService($categoryService);  
 $this->categoryRepository = $categoryRepository;  
 $this->paginator = $paginator;  
 }  
  
 #[Route('/admin/category/create', name: 'category\_create')]  
 public function createCategory(Request $request): Response  
 {  
 return $this->baseController->create(  
 $request,  
 CategoryType::class,  
 'admin\_category',  
 'category/categoryCreate.html.twig',  
 );  
 }  
  
 */\*\*  
 \* @throws Exception  
 \*/* #[Route('/admin/category/update/{id}', name: 'category\_update')]  
 public function updateCategory(Request $request, Category $category): Response  
 {  
 $prevImageName = $category->getImage();  
 $this->denyAccessUnlessGranted(AdminVoter::*VIEW*, $this->getUser(), "Access Denied. Only for Admins");  
 $form = $this->createForm(CategoryType::class, $category);  
 $form->handleRequest($request);  
 if ($form->isSubmitted() && $form->isValid()) {  
 $this->categoryService->updateWithImage($prevImageName, $form);  
 return new RedirectResponse($this->generateUrl('admin\_category'));  
 }  
 return $this->render('category/categoryCreate.html.twig', [  
 'form' => $form->createView(),  
 'prevImageName' => $prevImageName  
 ]);  
 }  
  
 */\*\*  
 \* @throws Exception  
 \*/* #[Route('/admin/category/delete/{id}', name: 'category\_delete')]  
 public function deleteCategory(Request $request, Category $category): Response  
 {  
  
 return $this->baseController->deleteWithConfirmation($request,  
 $category,  
 'admin\_category',  
 'admin\_category',  
 );  
 }  
  
 #[Route('/category', name: 'category')]  
 public function index(Request $request): Response  
 {  
 $query = $this->categoryRepository->queryFindAll();  
 $pagination = $this->paginator->paginate(  
 $query,  
 $request->query->getInt('page', 1),  
 $request->query->getInt('limit', 7),  
 );  
  
 return $this->render('category/index.html.twig', [  
 'pagination' => $pagination  
 ]);  
 }  
  
  
}

<?php  
  
namespace App\Controller;  
  
use App\Entity\User;  
use App\Form\UserCreateType;  
use App\Repository\StoryPartsRepository;  
use App\Repository\StoryRepository;  
use App\Repository\UserRepository;  
use App\Service\UserService;  
use App\Voter\AdminVoter;  
use Knp\Component\Pager\PaginatorInterface;  
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;  
use Symfony\Component\Form\CallbackTransformer;  
use Symfony\Component\Form\Extension\Core\Type\FileType;  
use Symfony\Component\Form\Extension\Core\Type\SubmitType;  
use Symfony\Component\HttpFoundation\JsonResponse;  
use Symfony\Component\HttpFoundation\RedirectResponse;  
use Symfony\Component\HttpFoundation\Request;  
use Symfony\Component\HttpFoundation\Response;  
use Symfony\Component\Routing\Annotation\Route;  
  
class UserProfileController extends AbstractController  
{  
 private StoryRepository $storyRepository;  
 private PaginatorInterface $paginator;  
 private UserService $userService;  
  
 public function \_\_construct  
 (  
 StoryRepository $storyRepository,  
 PaginatorInterface $paginator,  
 UserService $userService,  
 )  
 {  
 $this->storyRepository = $storyRepository;  
 $this->paginator = $paginator;  
 $this->userService = $userService;  
 }  
  
 #[Route('/user/{id}', name: 'user\_profile')]  
 public function index(User $user, UserRepository $userRepository): Response  
 {  
  
 return $this->render('user\_profile/index.html.twig', [  
 'user' => $user,  
 ]);  
 }  
  
 #[Route('/user/avatar/{id}', name: 'user\_avatar\_update', methods: ['GET', 'POST'])]  
 public function updateAvatar(Request $request, User $user): Response  
 {  
 if ($request->isXmlHttpRequest()) {  
 $this->userService->updateAvatarAjax($request->request->get('image'), $user);  
 return new JsonResponse(1);  
 }  
 $prevImageName = $user->getImage();  
 $form = $this->createFormBuilder($user)  
 ->add('image', FileType::class,[  
 'label'=>'Изображение'  
 ]);  
  
 $form->get('image')->addModelTransformer(new CallBackTransformer(  
 function ($imageUrl) {  
 return null;  
 },  
 function ($imageUrl) {  
 return $imageUrl;  
 }  
 ));  
 $form = $form->getForm();  
 return $this->render('user/changeAvatar.html.twig', [  
 'form' => $form->createView(),  
 'prevImageName' => $prevImageName  
 ]);  
 }  
  
 #[Route('/user/profile/mystories', name: 'user\_my\_stories')]  
 public function myStories(): Response  
 {  
 $stories = $this->storyRepository->getAllUsersStories($this->getUser())->getQuery()->getResult();  
  
 return $this->render('user\_profile/my\_stories.html.twig', [  
 'stories' => $stories,  
 ]);  
 }  
}

<?php  
  
namespace App\Controller;  
  
use App\Entity\User;  
use App\Form\UserCreateType;  
use App\Interfaces\UserServiceInterface;  
use App\Service\UserService;  
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;  
use Symfony\Component\HttpFoundation\Request;  
use Symfony\Component\HttpFoundation\Response;  
use Symfony\Component\Routing\Annotation\Route;  
  
class UserController extends AbstractController  
{  
 private BaseController $baseController;  
  
 public function \_\_construct  
 (  
 UserService $userService,  
 BaseController $baseController,  
 )  
 {  
 $this->baseController = $baseController;  
 $this->baseController->setService($userService);  
 }  
  
 #[Route('/admin/user/create', name: 'user\_create')]  
 public function createUser(Request $request): Response  
 {  
 return $this->baseController->create(  
 $request,  
 UserCreateType::class,  
 'admin\_user',  
 'user/userCreate.html.twig',  
 );  
 }  
  
  
 #[Route('/admin/user/update/{id}', name: 'user\_update')]  
 public function editUser(Request $request, User $user): Response  
 {  
 return $this->baseController->update(  
 $request,  
 UserCreateType::class,  
 $user,  
 'admin\_user',  
 'user/userCreate.html.twig',  
 );  
 }  
  
 #[Route('/admin/user/delete/{id}', name: 'user\_delete')]  
 public function deleteUser(User $user): Response  
 {  
 return $this->baseController->delete(  
 $user,  
 'admin\_user',  
 );  
 }  
}

<?php  
  
namespace App\Controller;  
  
  
use App\Entity\Tag;  
use App\Form\CategoryType;  
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;  
use Symfony\Component\HttpFoundation\Request;  
use Symfony\Component\HttpFoundation\Response;  
use Symfony\Component\Routing\Annotation\Route;  
  
class TagController extends AbstractController  
{  
 private BaseController $baseController;  
  
 public function \_\_construct  
 (  
 BaseController $baseController,  
 )  
 {  
 $this->baseController = $baseController;  
 }  
  
 #[Route('/admin/tag/create', name: 'tag\_create')]  
 public function createTag(Request $request): Response  
 {  
 return $this->baseController->create(  
 $request,  
 new CategoryType(),  
 'admin\_tag',  
 'tag/tagCreate.html.twig',  
 );  
 }  
  
 #[Route('/admin/tag/update/{id}', name: 'tag\_update')]  
 public function updateTag(Request $request, Tag $tag): Response  
 {  
 return $this->baseController->update(  
 $request,  
 new CategoryType(),  
 $tag,  
 'admin\_tag',  
 'tag/tagCreate.html.twig',  
 );  
 }  
  
 #[Route('/admin/tag/delete/{id}', name: 'tag\_delete')]  
 public function deleteTag(Tag $tag): Response  
 {  
 return $this->baseController->delete(  
 $tag,  
 'admin\_tag',  
 );  
 }  
  
}

<?php  
  
namespace App\Controller;  
  
use App\Entity\Category;  
use App\Entity\Character;  
use App\Entity\Comments;  
use App\Entity\Fandom;  
use App\Entity\MpaaRating;  
use App\Entity\Review;  
use App\Entity\Status;  
use App\Entity\Story;  
use App\Entity\Tag;  
use App\Entity\User;  
use App\Form\CommentType;  
use App\Form\ReviewType;  
use App\Form\StoryType;  
use App\Repository\StoryRepository;  
use App\Service\CommentsService;  
use App\Service\ReviewsService;  
use App\Service\StoryService;  
use Doctrine\ORM\NonUniqueResultException;  
use Exception;  
use Knp\Component\Pager\PaginatorInterface;  
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;  
use Symfony\Component\Form\Extension\Core\Type\SubmitType;  
use Symfony\Component\Form\Extension\Core\Type\TextareaType;  
use Symfony\Component\HttpFoundation\JsonResponse;  
use Symfony\Component\HttpFoundation\RedirectResponse;  
use Symfony\Component\HttpFoundation\Request;  
use Symfony\Component\HttpFoundation\Response;  
use Symfony\Component\Routing\Annotation\Route;  
use Symfony\Component\Security\Core\Security;  
  
class StoryController extends AbstractController  
{  
 private BaseController $baseController;  
 private PaginatorInterface $paginator;  
 private StoryRepository $storyRepository;  
 private ReviewsService $reviewsService;  
 private CommentsService $commentsService;  
  
 public function \_\_construct  
 (  
 BaseController $baseController,  
 StoryService $storyService,  
 Security $security,  
 PaginatorInterface $paginator,  
 StoryRepository $storyRepository,  
 ReviewsService $reviewsService,  
 CommentsService $commentsService,  
 )  
 {  
 $this->baseController = $baseController;  
 if ($security->getUser()) {  
 $storyService->setUser($security->getUser());  
 }  
 $this->baseController->setService($storyService);  
 $this->paginator = $paginator;  
 $this->storyRepository = $storyRepository;  
 $this->reviewsService = $reviewsService;  
 $this->commentsService = $commentsService;  
 }  
  
 #[Route('/admin/story/create', name: 'story\_create')]  
 public function createStory(Request $request): Response  
 {  
  
 return $this->baseController->create(  
 $request,  
 StoryType::class,  
 'admin\_story',  
 'story/index.html.twig',  
 );  
 }  
  
 #[Route('/admin/story/update/{id}', name: 'story\_update')]  
 public function updateStory(Request $request, Story $story): Response  
 {  
 return $this->baseController->update(  
 $request,  
 StoryType::class,  
 $story,  
 'admin\_story',  
 'story/index.html.twig',  
 );  
 }  
  
 */\*\*  
 \* @throws Exception  
 \*/* #[Route('/admin/story/delete/{id}', name: 'story\_delete')]  
 public function deleteStory(Request $request, Story $story): Response  
 {  
 return $this->baseController->deleteWithConfirmation(  
 $request,  
 $story,  
 'admin\_story',  
 'admin\_story',  
 );  
 }  
  
 */\*\*  
 \* @throws NonUniqueResultException  
 \*/* #[Route('/readstory/{id}', name: 'story\_read', methods: ['GET'])]  
 public function readStory(Story $story): Response  
 {  
 $proxyStory = $this->storyRepository->getStoryById($story)->getQuery()->getOneOrNullResult();  
  
 return $this->render('story/readStory.html.twig', [  
 'story' => $proxyStory,  
 'likesCount'=>$story->getLikes()->count(),  
 ]);  
 }  
  
 #[Route('/createCommentsForPart/{id}', name: 'comments\_create', methods: ['POST'])]  
 public function readStoryAjax(Request $request): Response  
 {  
 if ($request->isXmlHttpRequest()) {  
 $parametersAsArray = [];  
 if ($content = $request->getContent()) {  
 $parametersAsArray = json\_decode($content, true);  
 }  
 $text = $parametersAsArray['text'];  
 $partsId = $parametersAsArray['partsId'];  
 $storyId = $parametersAsArray['storyId'];  
 $this->commentsService->createComment($text, $partsId, $this->getUser());  
 return new JsonResponse(1);  
 }  
 return new JsonResponse(0);  
 }  
 #[Route('/comment/{id}/delete', name: 'comments\_delete')]  
 public function deleteStoryComment(Comments $comments): Response  
 {  
 $this->commentsService->deleteComment($comments);  
 return new RedirectResponse($this->generateUrl('home'));  
 }  
  
  
 */\*\*  
 \* @throws NonUniqueResultException  
 \*/* #[Route('/story/{id}/reviews', name: 'story/reviews\_read')]  
 public function readStoryReviews(Story $story, Request $request): Response  
 {  
 $proxyStory = $this->storyRepository->getStoryById($story)->getQuery()->getOneOrNullResult();  
 $form = $this->createForm(ReviewType::class);  
  
 $form->handleRequest($request);  
  
 if ($form->isSubmitted() && $form->isValid()) {  
 $this->reviewsService->createReview($form->getData(), $story, $this->getUser());  
 return new RedirectResponse($this->generateUrl('story/reviews\_read', ['id' => $story->getId()]));  
 }  
  
 return $this->render('story/readStoryReviews.html.twig', [  
 'story' => $proxyStory,  
 'form' => $form->createView(),  
 ]);  
 }  
  
 #[Route('/review/{id}/delete', name: 'reviews\_delete')]  
 public function deleteStoryReview(Review $review): Response  
 {  
 $this->reviewsService->deleteReview($review);  
 return new RedirectResponse($this->generateUrl('home'));  
 }  
  
 #[Route('stories/all', name: 'stories/all\_show')]  
 public function showStoriesAll(Request $request): Response  
 {  
 $query = $this->storyRepository->queryFindAll()->getQuery()->getResult();  
 return $this->baseController->showWithPagination($request, $query, 'story/readStories.html.twig');  
 }  
  
 #[Route('stories/category/{id}', name: 'stories/category\_show')]  
 public function showStoriesInCategory(Request $request, Category $category): Response  
 {  
 $query = $this->storyRepository->getStoryByCategoryId($category)->getQuery()->getResult();  
 return $this->baseController->showWithPagination($request, $query, 'story/readStories.html.twig');  
 }  
  
 #[Route('stories/fandom/{id}', name: 'stories/fandom\_show')]  
 public function showStoriesInFandom(Request $request, Fandom $fandom): Response  
 {  
 $query = $this->storyRepository->getStoryByFandomId($fandom)->getQuery()->getResult();  
 return $this->baseController->showWithPagination($request, $query, 'story/readStories.html.twig');  
 }  
  
 #[Route('stories/character/{id}', name: 'stories/character\_show')]  
 public function showStoriesInCharacter(Request $request, Character $character): Response  
 {  
 $query = $this->storyRepository->getStoryByCharacterId($character)->getQuery()->getResult();  
 return $this->baseController->showWithPagination($request, $query, 'story/readStories.html.twig');  
 }  
  
 #[Route('stories/tag/{id}', name: 'stories/tag\_show')]  
 public function showStoriesInTags(Request $request, Tag $tag): Response  
 {  
 $query = $this->storyRepository->getStoryByTagId($tag)->getQuery()->getResult();  
 return $this->baseController->showWithPagination($request, $query, 'story/readStories.html.twig');  
 }  
  
 #[Route('stories/status/{id}', name: 'stories/status\_show')]  
 public function showStoriesInStatus(Request $request, Status $status): Response  
 {  
 $query = $this->storyRepository->getStoryByStatusId($status)->getQuery()->getResult();  
 return $this->baseController->showWithPagination($request, $query, 'story/readStories.html.twig');  
 }  
  
 #[Route('stories/mpaaRating/{id}', name: 'stories/mpaaRating\_show')]  
 public function showStoriesInMpaaRating(Request $request, MpaaRating $mpaaRating): Response  
 {  
 $query = $this->storyRepository->getStoryByMpaaRatingId($mpaaRating)->getQuery()->getResult();  
 return $this->baseController->showWithPagination($request, $query, 'story/readStories.html.twig');  
 }  
 #[Route('stories/author/{id}', name: 'stories/author\_show')]  
 public function showStoriesInAuthors(Request $request, User $user): Response  
 {  
 $query = $this->storyRepository->getStoryByAuthorId($user)->getQuery()->getResult();  
 return $this->baseController->showWithPagination($request, $query, 'story/readStories.html.twig');  
 }  
  
}

<?php  
  
namespace App\Controller;  
  
  
use App\Entity\Status;  
use App\Form\CategoryType;  
use App\Form\StatusType;  
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;  
  
use Symfony\Component\HttpFoundation\Request;  
use Symfony\Component\HttpFoundation\Response;  
use Symfony\Component\Routing\Annotation\Route;  
  
class StatusController extends AbstractController  
{  
 private BaseController $baseController;  
  
 public function \_\_construct  
 (  
 BaseController $baseController,  
 )  
 {  
 $this->baseController = $baseController;  
 }  
  
 #[Route('/admin/status/create', name: 'status\_create')]  
 public function createStatus(Request $request): Response  
 {  
 return $this->baseController->create(  
 $request,  
 new StatusType(),  
 'admin\_status',  
 'status/statusCreate.html.twig',  
 );  
 }  
  
 #[Route('/admin/status/update/{id}', name: 'status\_update')]  
 public function updateStatus(Request $request, Status $status): Response  
 {  
 return $this->baseController->update(  
 $request,  
 new CategoryType(),  
 $status,  
 'admin\_status',  
 'status/statusCreate.html.twig',  
 );  
 }