Test WeBreathe

Etape par étape du test :
Pour vous connecter en tant qu'admin : test@test.fr mot de passe : test
Sommaire :

Création du projet symfony :

Se mettre dans le bon dossier où on veut créer le projet
symfony new --full NomDuProjet

Lancer le serveur symfony
symfony serve

Création de la première page ''home''
Pour créer une classe :
php bin/console make :controller home

si la commande make n'est pas reconnue :
composer require symfony/maker-bundle –dev
composer require doctrine/annotations

Cela crée automatiquement
la classe ''home'' dans Controller
et la page twig associé à ce controller dans le dossier templates

---

Partie FRONT :
Création dans le fichier base.html.twig la barre de navigation au dessus du **block body**
Afin de pouvoir directement se servir de cette même barre pour appeler nos pages !
Ainsi que la création du footer en dessous du **block body**
Cela permet de générer la barre de navigation et le footer sur toutes les pages que l'ont appelera par la suite.

Utilisation des routes :
Dans le contrôle de la page home on retrouve :
Ce qui est affiché dans l'url

```
/**
     * @Route("/", name="home")
     */
```

Et le name qui nous sert à appeler la page en liaison avec ce controller , il suffit par exemple de mettre ce name dans la barre de navigation de la façon suivante pour accèder à la page voulu :

```
<a class="navlink active" ariacurrent="page" href="{{path('home')}}">Accueil</
a>
```

On peut ensuite préparer nos futures pages et renouveller cette opération

Partie Back :

Création d'une entité

Pour créer une entité : php bin/console make:entity
Donner le nom de l'entité
Puis le nom de chaque propriété(champs) de l'entité exemple : name
Le type de champ exemple : string
Si le champ peu etre null ou non

Ensuite
Dans le fichier**.env** dans **vendor**
Décommenter la ligne et personnaliser la route suivante :
DATABASE_URL="mysql://root@127.0.0.1:3306/testWeBreathe?serverVersion=mariadb-10.4.14"
Cela permet de faire la relation entre PHP MyAdmin et Symfony pour la création d'une base de donnée

Et commenter la ligne suivante car nous n'en avons pas besoin:
#DATABASE_URL="postgresql://db_user:db_password@127.0.0.1:5432/db_name?serverVersion=13&charset=utf8"

Création d'une base de donnée

Pour créer la base de donnée :
php bin/console doctrine:database:create

Pour mettre à jour la base de donnée :
php bin/console doctrine:schema:update --force

Pour voir la base de donnée : go sur Xampp
Dans la ligne Mysql => **admin**
Aller sur le projet concerné
Cliquer sur la **Table** puis l'onglet **Structure**

Création d'un formulaire (Ajout de Modules):

On crée le controller et la page où l'on veut afficher le formulaire :
Php bin/console make:controller NomDeLaPage (ici , **addModule**)

Et ensuite on crée un controller **form** avec la commande :
php bin/console make:form addModule_form
Puis on indique le nom exact de l'**entité (**ici Module**)**
Cela nous crée à partir de l'entité, un formulaire que l'on va pouvoir manipuler et utiliser afin d'ajouter nos modules

```php
<?php

namespace App\Form;

use App\Entity\Module;
use Symfony\Component\Form\AbstractType;
use Symfony\Component\Form\FormBuilderInterface;
use Symfony\Component\OptionsResolver\OptionsResolver;

class AddModuleFormType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options)
    {
        $builder
            ->add('name')
            ->add('number')
            ->add('description')
            ->add('type')
            ->add('temperature')
            ->add('duree_fonctionnement')
            ->add('donnees_envoyees')
            ->add('etat_de_marche')
        ;
    }

    public function configureOptions(OptionsResolver $resolver)
    {
        $resolver->setDefaults([
            'data_class' => Module::class,
        ]);
    }
}
```

Dans le **controller** de la page **addModule** on lie le formulaire afin de l'afficher dans la page que l'on souhaite :

```php
<?php

namespace App\Controller;

use App\Entity\Module;
use App\Form\AddModuleFormType;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;

class AddModuleController extends AbstractController
{
    /**
     * @Route("/add/module", name="add_module")
     */
        public function index(Request $request):Response{

            $form = $this->createForm(AddModuleFormType::class, $Module);

        return $this->render('add_module/index.html.twig', [
            'AddModuleFormType' => $form->createView(),
        ]);
    }
}
```

Puis dans la page en question où l'on veut afficher le formulaire:
On appelle ce formulaire comme ceci afin de l'afficher :

```
{% extends 'base.html.twig' %}

{% block title %}Hello AddModuleController!{% endblock %}

{% block body %}

<div class="container">
<h1>Ajouter un module</h1>

    {{ form_start(AddModuleFormType) }}
        {{ form_widget(AddModuleFormType) }}<!-- input-->
        <button class="btn btn-primary m-4">Ajouter le module</button>
    {{ form_end(AddModuleFormType) }}
</div>
{# J'affiche mon formulaire #}

{% endblock %}
```

Pour donner un thème bootstrap au formulaire on va dans le fichier **twig .yaml** qui se trouve dans **config/packages** puis on rajoute la ligne **form_themes** :

```
twig:
    default_path: '%kernel.project_dir%/templates'
    form_themes:
        - bootstrap_4_layout.html.twig
```

Ajout du produit à la base de données grâce au formulaire :

Pour ça on modifie le controller de la page du formulaire comme ceci :

```php
<?php

namespace App\Controller;

use App\Entity\Module;
use App\Form\AddModuleFormType;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;

class AddModuleController extends AbstractController
{
    /**
     * @Route("/add/module", name="add_module")
     */

        public function create(Request $request):Response{

            $Module = new Module();

            $form = $this->createForm(AddModuleFormType::class, $Module);
            $form->handleRequest($request);
            if($form->isSubmitted() && $form->isValid()){

     //On ajoute l'objet dans la BDD
     //On récupere le service doctrine qui permet de gérer la base de données
             $entityManager = $this->getDoctrine()->getManager();
             //On doit mettre l'objet en attente
             $entityManager->persist($Module);
             //On exécute la requete
             $entityManager->flush();

             return $this->redirecttoRoute('modules');
            //Puis on redirige faire la page modules
            }

        return $this->render('add_module/index.html.twig', [
            'AddModuleFormType' => $form->createView(),
        ]);
    }
}
```

Afficher tout les modules

Pour retourner tout les modules dans la page Modules , dans le controller de cette page nous créons la requete pour aller chercher tout ce qu'il y a dans la table module de notre base de données .

```php
namespace App\Controller;

use App\Entity\Module;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;

class ModulesController extends AbstractController
{
    /**
     * @Route("/modules", name="modules")
     */
    public function index(Request $request): Response
    {
        $repository = $this->getDoctrine()->getRepository(Module::class);
        $Modules = $repository->findAll();


        return $this->render('modules/index.html.twig', [
            'modules' => $Modules,
        ]);
    }
}
```

Puis dans la page Modules on crée une **boucle for … in**
Afin d'afficher pour chaque élément une ''card'' de chaque module de notre base de données
Tout en renseignant les données que l'on souhaite afficher ces ''card''

```
{% extends 'base.html.twig' %}

{% block title %}Hello ModulesController!{% endblock %}

{% block body %}

<div class="homeCover">
    <h1 class="col-lg-8">Vos modules</h1>
    <p class="m-
5">Depuis plusieurs années, les objets connectés à internet se multiplient, c'
est pourquoi nous proposons des solutions efficaces afin que vous puissiez gér
er et visualiser au mieux vos modules.</p>
</div>

    <div class="AllModule">

        {% for module in modules %}
            <div class="card" style="width: 18rem;">
                <li class="list-group-item etat">
                    <div class="circleGreen"></div>
                    <div class="circleOrange"></div>
                    <div class="circleRed"></div>
                </li>
                <img src="{{asset('img/module/montre.jpg')}}" class="card-img-
top" alt="...">
                <div class="card-body">
                    <h5 class="card-
title">Module n°{{module.number}} : {{module.name}}</h5>
                    <p class="card-text">{{module.type}}</p>
                    <p class="card-text">{{module.description}}</p>
                    <a href="#" class="btn btn-primary">Go somewhere</a>
                </div>
            </div>
        {% endfor %}

    </div>


{% endblock %}
```

Ajout d'un message de succès :

Afin d'afficher un message de succès pour les futurs ajouts / suppressions/modification etc
On met en place un système pour que , quand on rajoute `$this->addFlash` aux endroits qui nous interessent , on afficher un message de succès pour chaque action.

Dans AddModuleController :
En dessous de :

```
$entityManager->flush();
```

Ajout de :

```
 $this->addFlash
('success', 'votre module '.$Module>getName().'a bien été ajouté');
```

Dans base.html.twig : au dessus du block body

```
    {% for label, messages in app.flashes %}
        {% for message in messages %}
            <div class="alert alert-{{ label }} mt-4">
                {{ message }}
            </div>
        {% endfor %}
    {% endfor %}
```

Pour cet exemple un message de succès s'affichera une fois l'ajout fait !

Ajout image :

Ajout de la possibilité de mettre une image concernant le module :

mise a jour d'une entity :
php bin/console make:entity **(Si on veut rajouter un champ)**
et remettre EXACTEMENT le nom de l'entité que l'on veut changer ici (Module)
et ajouter le champ : image
puis pour mettre a jour : **php bin/console doctrine:schema:update --force**

ajout du champs image dans AddModuleFormType avfin que le champ soit aussi dans le formulaire

```
 public function buildForm(FormBuilderInterface $builder, array $options)
    {
        $builder
            ->add('name')
            ->add('number')
            ->add('description')
            ->add('type')
            ->add('temperature')
            ->add('duree_fonctionnement')
            ->add('donnees_envoyees')
            ->add('etat_de_marche')
            ->add('image', FileType::class, [])

;

    }
```

Dans **services.yaml**
Le lien qui nous servira pour indiquer où vont se stocker les images

```
parameters:
    upload_directory: '%kernel.project_dir%/public/img/module'
```

on rajoute cette partie dans le controller de la page de l'ajout

```
//_____________________UPLOAD IMAGE_____________________________________
            /** @var UploadedFile $image */
            $image= $form->get('image')->getData();
            if($image){
                $filename = uniqid().'.'.$image->guessExtension();
                $image->move($this-
>getParameter('upload_directory'), $filename);
                $Module->setImage($filename);
            }else{
                $Module->setImage('default.png');
            }
```

Page Edit

Création d'une page pour editer le module que l'on veut

Php bin/console make :controller edit

```
class EditController extends AbstractController
{
    /**
     * @Route("/module/modifier/{id}", name="edit")
     */
    public function edit(Request $request, Module $Module):Response{
        //On crée le formulaire a partir de AddProductFormType (dans le dossie
r form)
        //symfony rempli l'objet $OneProduct avec les données du formulaire
        // grâce à la request
        $form = $this->createForm(AddModuleFormType::class, $Module);
        //Permet de lié le formulaire à la requete (pour récuperer le $_POST)
        $form->handleRequest($request);
        if($form->isSubmitted() && $form->isValid()){
             $image= $form->get('image')->getData();
        if($image){
            $defaultImages = ['default.png'];
            if($Module->getImage() && !in_array($Module-
>getImage(), $defaultImages )) {
                // FileSystem permet de manipuler les fichiers
                $fs = new Filesystem();
                // On supprime l'ancienne image
                $fs->remove($this-
>getParameter('upload_directory').'/'.$Module->getImage());
            }

            $filename = uniqid().'.'.$image->guessExtension();
            $image->move($this->getParameter('upload_directory'), $filename);
            $Module->setImage($filename);
        }
      //On ajoute l'objet dans la BDD
      //On récupere le service doctrine qui permet de gérer la base de données
        $this->getDoctrine()->getManager()->flush();

        //_______Redirection et message de succes_____________________
         $this->addFlash('success', 'votre module a bien été modifié');
         return $this->redirecttoRoute('modules');
        }
        return $this->render('edit/index.html.twig', [
            'AddModuleFormType' => $form->createView(),
            'Module' => $Module
        ]);
    }
}
```

Supprimer un module :

php bin/console make:controller delete

Pour supprimer un module , on crée un controller où on fait cette requete à chaque fois que l'on appellera la route de ce controller via un bouton supprimer .
Cela supprimera le module en question en affichant ensuite un message de confirmation
Puis le retour vers la page module

dans le controller delete :

```
namespace App\Controller;

use App\Entity\Module;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;

class DeleteController extends AbstractController
{
    /**
     * @Route("/module/supprimer/{id}", name="delete")
     */
    public function delete(Module $Module): Response
    {
        $entityManager = $this->getDoctrine()->getManager();
        $entityManager->remove($Module);
        $entityManager->flush();

        $this->addFlash('danger', 'Votre module a bien été supprimé');
        return $this->redirecttoRoute('modules');

    }
}
```

Sur le bouton supprimer :

```
<a href="{{path('delete', {id: module.id})}}" class="btn btn-
primary">Supprimer</a>
```

Création des users

Php bin/console make:user **(4 x entrées puisqu'on garde les valeurs par défaut)**

Création des fixtures :
composer require doctrine/doctrine-fixtures-bundle --dev
Créer un fichier **AppFixtures.php** dans src/DataFixtures

Permet d'installer le service Fixture qui permet de remplir la base de donnée

php bin/console doctrine:schema:update –force
pour mettre la base de données à jour et prendre en compte la table ''user''

dans appFixtures :

```php
namespace App\DataFixtures;

use App\Entity\User;
use Doctrine\Bundle\FixturesBundle\Fixture;
use Doctrine\Persistence\ObjectManager;
use Symfony\Component\Security\Core\Encoder\UserPasswordEncoderInterface;

class AppFixtures extends Fixture
{
    private $passwordEncoder;
    //On crée la variable
    //Puis on apelle et on stock le service dans la variable
    public function __construct(UserPasswordEncoderInterface  $passwordEncoder
){
        $this->passwordEncoder = $passwordEncoder;
    }

    public function load(ObjectManager $manager)
    {
        $user = new User();
        $user->setEmail('test@test.fr');
        $user->setPassword($this->passwordEncoder-
>encodePassword($user, 'test'));
        $user->setRoles(['ROLE_ADMIN']);
        $manager->persist($user);

        $manager->flush();
    }
}
```

php bin/console doctrine:fixtures:load
Cela permet de charger la base de données
puis faire **Yes**

Authentification :

php bin/console make:auth

1 Login form authenticator on sélectionne le fait de se connecter avec une authentification
On donne un nom de classe
On laisse le nom par défaut du controller
Et on dit yes pour générer une route pour la déconnection

Cette commande crée :

- Une template pour se co
- Un controller
- Et a mis a jour le fichier **sécurity.yaml** nottamment la route pour se délog

On supprimer la ligne

```
 throw new \Exception('TODO: provide a valid redirect inside '.__FILE__);
```

du fichier **AppLoginAuthenticator** du dossier **sécurity**

et remplacer par

```
 return new RedirectResponse($this->urlGenerator->generate('home'));
```

afin de rediriger sur l'accueil une fois logger

On rajoute ces quelques lignes dans la barre de navigation
Afin d'afficher le nom du user si il est connecté , sinon on marque « mon compte ».

```
{% if app.user %}
     {{ app.user.username }}
{% else %}
     mon compte
{% endif %}
```

On crée le bouton de connexion

```
<li><a class="dropdown-item" href="{{ path('app_login') }}">Connexion</a></li>
```

On crée le bouton de déconnexion

```
<li><a class="dropdown-item" href="{{ path('app_logout') }}">
Déconnexion</a></li>
```

Tout en veillant à mettre la route vers les controller qui correspondent

Inscription

php bin/console make:registration-form

la premiere ligne veille à ce qu'on ne puisse pas s'enregistrer 2 fois (on a mis yes)
la deuxieme sert a demander l'envoie de mail pour confirmer l'inscription (on a mis no)
la troisieme sert a connecter l'utilisateur dès l'inscription

cela crée un controller , une templates et un formulaire et a mis à jour l'entité User

rajout de la route dans la barre nav

```
<li><a class="dropdown-item" href="{{ path('app_register') }}">
Inscription</a></li>
```

Dans le controller du formulaire ici RegistrationFormType on a ceci :

```
  ->add('plainPassword', PasswordType::class, [
                // instead of being set onto the object directly,
                // this is read and encoded in the controller
                'mapped' => false,
                'constraints' => [
                    new NotBlank([
                        'message' => 'Please enter a password',
                    ]),
                    new Length([
                        'min' => 6,
                        'minMessage' => 'Your password should be at least {{ l
imit }} characters',
                        // max length allowed by Symfony for security reasons
                        'max' => 4096,
                    ]),
                ],
            ])
        ;
```

Que l'on change en cela =>
Pour avoir la confirmation de mot de passe

```
->add('plainPassword', RepeatedType::class, [
                // instead of being set onto the object directly,
                // this is read and encoded in the controller
                'mapped' => false,
                'type'=> PasswordType::class,
                'first_options' =>['label' => 'Mot de passe'],
                'second_options' =>['label' => 'Confirmer mot de passe'],

                'constraints' => [
                    new NotBlank([
                        'message' => 'Veuillez entrez votre mot de passe ',
                    ]),
                    new Length([
                        'min' => 4,
                        'minMessage' => 'Votre mot de passe doit contenir au m
oins {{ limit }} caractères',
                        // max length allowed by Symfony for security reasons
                        'max' => 4096,
                    ]),
```

Dans le User.php (l'entité) on peut changer la phrase si un email est déjà pris : (Vers la ligne 12)

```
 * @UniqueEntity(fields={"email"}, message="Il y a déjà un compte avec cet ema
il")
```

Sécurité des routes USER / NON USER / ADMIN dans sécurity.yaml

```
access_control:
        # - { path: ^/admin, roles: ROLE_ADMIN }
        # - { path: ^/profile, roles: ROLE_USER }
        - { path: ^/add/product, roles: ROLE_ADMIN }
        - { path: ^/add/module, roles: ROLE_USER }
        - { path: ^/modules, roles: ROLE_USER }
```

Changement du toggle dans la navBar pour que les user , non user et admin voient ce qu'ils sont destiné à voir en fonction de leur statut

```
{% if is_granted('ROLE_ADMIN') %}
      <li><a class="dropdown-item" href="#">
      Gérer mes équipements (DashBoard)</a></li>
{% endif %}

{% if app.user %}
    <li><a class="dropdown-item" href="{{path('add_module')}}">
      Ajouter d'un module</a></li>
    <li><a class="dropdown-item" href="{{ path('app_logout') }}">
      Déconnexion</a></li>
{% else %}
     <li><a class="dropdown-item" href="{{ path('app_login') }}">
      Connexion</a></li>
     <li><a class="dropdown-item" href="{{ path('app_register') }}">
      Inscription</a></li>
{% endif %}
```

Changement de formulaire d'ajout

pour mettre un select ainsi que deux barres afin d'ajuster le nombre d'heure de fonctionnement

```php
<?php

namespace App\Form;

use App\Entity\Module;
use Symfony\Component\Form\AbstractType;
use Symfony\Component\Form\Extension\Core\Type\ChoiceType;
use Symfony\Component\Form\Extension\Core\Type\FileType;
use Symfony\Component\Form\Extension\Core\Type\RangeType;
use Symfony\Component\Form\FormBuilderInterface;
use Symfony\Component\OptionsResolver\OptionsResolver;

class AddModuleFormType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options)
    {
        $builder
            ->add('name')
            ->add('number')
            ->add('description')
            ->add('type', ChoiceType::class, [
                'choices' => [
                  //'Affiche' => 'Value',
                    'Montre' => 1,
                    'Chauffage' => 2,
                    'Prise' => 3,
                    'Assistant Vocal' => 4,
                    'Caméra' => 5,
                ],
            ])

            ->add('temperature', RangeType::class,[
                'attr' => [
                    'min' => 0,
                    'max' => 30,
                    'class' => 'p-0',
                ]
            ])

            ->add('duree_fonctionnement', RangeType::class,[
                'attr' => [
                    'min' => 0,
                    'max' => 24,
                    'class' => 'p-0',
                ]
```

```
            ])
            ->add('donnees_envoyees')
            ->add('etat_de_marche')
            ->add('image', FileType::class, [
                  'mapped' => false,
                  'required' => false
                  ])
        ;
    }

    public function configureOptions(OptionsResolver $resolver)
    {
        $resolver->setDefaults([
            'data_class' => Module::class,
        ]);
    }
}
```

Javascript

Javascript pour personnaliser la barre de température ainsi que la barre de temps de fonctionnement
Afin de voir la valeur de la position de la barre de réglage

```
$('#add_module_form_temperature').after('<div id="result">'+$('#add_module_for
m_temperature').val()+'°C</div>')

$('#add_module_form_temperature').on('input', function() {

    $('#result').remove();
   //je récupere la valeur du input et l'ajoute directement en dessous de celu
i-ci
    $(this).after('<div id="result">'+$(this).val()+'°C</div>')
});



$('#add_module_form_duree_fonctionnement').after('<div id="result2">'+$('#add_
module_form_duree_fonctionnement').val()+'H</div>')

$('#add_module_form_duree_fonctionnement').on('input', function() {

    $('#result2').remove();
    $(this).after('<div id="result2">'+$(this).val()+'H</div>')
});
```

Dashboard

Création d'un Dashboard, qui est une autre solution pour manipuler les modules

composer require easycorp/easyadmin-bundle
php bin/console make:admin:dashboard (entréeX2)

dans la barre de navigation

```
<li><a class="dropdown-item" href="{{path('admin')}}">
Gérer mes équipements (DashBoard)</a></li>
```

Dans Security.yaml

```
   access_control:
        # - { path: ^/admin, roles: ROLE_ADMIN }
        # - { path: ^/profile, roles: ROLE_USER }
        - { path: ^/add/product, roles: ROLE_ADMIN }
        - { path: ^/add/module, roles: ROLE_USER }
        - { path: ^/modules, roles: ROLE_USER }
        - { path: ^/easyadmin, roles: ROLE_ADMIN}
```

Puis on crée ce qu'on appelle les CRUD afin de gérer nos entités , ou plus précisément ce qu'il y a dans nos tables MySQL , on peut Créer , Lire , Modifier et Supprimer les éléments d'une table .
php bin/console make:admin:crud
choisir l'entité que l'on veut gerer
2x entrée

Et dans le DashboardController

```
 yield MenuItem::linkToCrud('Modules', 'fas fa-list', Module::class);
```

(composer update easycorp/easyadmin-bundle) au cas ou si besoin

Ajout catégorie

php bin/console make:entity puis **entrée** en on indique le nom'**ModuleCategory'**
on rajoute un champ **CategoryName** et c'est tout

ensuite pour mettre jour la bdd :
php bin/console doctrine:schema:update --force

Ensuite on ajoute le champ catégory à notre entité **Module** afin que l'on puisse renseigner chaque catégorie pour chaque module en prenant soin de renseigner qu'il y a une relation de **ManyToOne** avec l'entité **ModuleCategory**

```
PS C:\Users\Mikie\Desktop\Test-WeBreathe\TestWeBreathe> php bin/console make:entity

 Class name of the entity to create or update (e.g. FierceChef):
 > Module

 Your entity already exists! So let's add some new fields!

 New property name (press <return> to stop adding fields):
 > Category

 Field type (enter ? to see all types) [string]:
 > ?

Main types
  * string
  * text
  * boolean
  * integer (or smallint, bigint)
  * float

Relationships / Associations
  * relation (a wizard will help you build the relation)
  * ManyToOne
  * OneToMany
  * ManyToMany
  * OneToOne

Array/Object Types
  * array (or simple_array)
  * json
  * object
  * binary
  * blob

Date/Time Types
  * datetime (or datetime_immutable)
  * datetimetz (or datetimetz_immutable)
  * date (or date_immutable)
  * time (or time_immutable)
  * dateinterval
```

```
Other Types
  * ascii_string
  * decimal
  * guid
  * json_array
  * uuid
  * ulid


 Field type (enter ? to see all types) [string]:
 > ManyToOne

 What class should this entity be related to?:
 > ModuleCategory

 Is the Module.Category property allowed to be null (nullable)? (yes/no) [yes]:
 > no

 Do you want to add a new property to ModuleCategory so that you can access/update Module
objects from it - e.g. $moduleCategory->getModules()? (yes/no) [yes]:
 >

 A new property will also be added to the ModuleCategory class so that you can access the related
Module objects from it.

 New field name inside ModuleCategory [modules]:
 >

 Do you want to activate orphanRemoval on your relationship?
 A Module is "orphaned" when it is removed from its related ModuleCategory.
 e.g. $moduleCategory->removeModule($module)

 NOTE: If a Module may *change* from one ModuleCategory to another, answer "no".

 Do you want to automatically delete orphaned App\Entity\Module objects (orphanRemoval)?
(yes/no) [no]:
 > yes

 updated: src/Entity/Module.php
 updated: src/Entity/ModuleCategory.php

 Add another property? Enter the property name (or press <return> to stop adding fields):
 >

  Success!


 Next: When you're ready, create a migration with php bin/console make:migration

PS C:\Users\Mikie\Desktop\Test-WeBreathe\TestWeBreathe>
```

Supprimer le getter et setter et la variable ‘‘type’’ de Module puisqu’on utilisera le champ **category**

Dans **appfixtures** on crée un tableau avec des noms de catégorie

```
$categories = ['Montre', 'Chauffage', 'Prise', 'Assistant Vocal','Caméra'];

        foreach ($categories as $key => $category){
            $ModuleCategory = new ModuleCategory();
            $ModuleCategory->setCategoryName($category);
            $this->addReference('stuff-'.$key, $ModuleCategory);
            $manager->persist($ModuleCategory);
        }
```

Et ensuite on ajoute le champ **Category** a la place de type dans le formulaire d’ajout

```
->add('Category', EntityType::class, [
                'class' => ModuleCategory::class,
                'choice_label' => 'CategoryName',
            ])

            // ->add('Category', null, [
            //          'choice_label' => 'CategoryName',
            //          'expanded' => true
            //      ])
```

Pour afficher la catégorie dans chaque card de module , on fait comme ceci :

```
<p class="card-text">Type : {{module.category.CategoryName}}</p>
```

Afin de remettre a neuf notre base de donnée
On la supprime
php bin/console doctrine:database:drop –force
On la recrée
php bin/console doctrine:database:create
On met a jours notre base de donnée
php bin/console doctrine:schema:update –force
On relance nos fixtures
php bin/console doctrine:fixtures:load

Trie par categorie

php bin/console make:controller Category

affichage du menu de catégorie dans la page module
Pour ça on récuperer tout les éléments dans la table module
dans le controller :

```php
<?php

namespace App\Controller;

use App\Entity\Module;
use App\Repository\ModuleCategoryRepository;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;

class ModulesController extends AbstractController
{
    /**
     * @Route("/modules", name="modules")
     */
    public function index(ModuleCategoryRepository $moduleCategoryRepository):
 Response
    {
        $repository = $this->getDoctrine()->getRepository(Module::class);
        $Modules = $repository->findAll();

        $category = $moduleCategoryRepository->findAll();


        return $this->render('modules/index.html.twig', [
            'modules' => $Modules,
            'categories'=> $category
        ]);
    }
}
```

Dans le twig

```twig
{% for category in categories %}
       <li class="list-group-item"><a href="
      {{ path('category', {'id':category.id}) }}">{{ category.CategoryName }}
      </a></li>
{% endfor %}
```

Dans le module repository on crée une méthode pour appeler les modules de la catégorie voulu

```
    public function FindCategory($id){
        //On selectionne tout les champs de la table module
        //Select * FROM Module

        //comme on est dans le repository de Module on choppe la table module
automatiquement
        //en lui donnant l'alias "m"
        $qb = $this->createQueryBuilder('m') // m = module
                    ->innerJoin('m.Category', 'mc')
//On prend le champ Category de l'entité Module ,
                    //en lui donnant l'alias "mc"
                    ->andWhere('mc = :id')
                    ->setParameter('id', $id);

        dump($qb->getQuery()->getResult());
        return $qb->getQuery()->getResult();

     }
```

Dans le catégoryController

```php
<?php

namespace App\Controller;

use App\Entity\Module;
use App\Entity\ModuleCategory;
use App\Repository\ModuleCategoryRepository;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;

class CategoryController extends AbstractController
{
    /**
     * @Route("/category/{id}", name="category")
     */
    public function index($id, Request $request, ModuleCategoryRepository $mod
uleCategoryRepository): Response
    {

        //la on appelle la table Module
        $repository = $this->getDoctrine()->getRepository(Module::class);

        //appelle la méthode
        $ModuleByCategory = $repository->FindCategory($id);


        //Affiche du menu de catégorie
        $categories = $moduleCategoryRepository->findAll();




        return $this->render('category/index.html.twig', [
            'categories'=> $categories,
            'ModuleByCategory' => $ModuleByCategory

        ]);
    }
}
```

Dans le twig de catégorie :

```
{% extends 'base.html.twig' %}

{% block title %}Hello CategoryController!{% endblock %}

{% block body %}

    {% for category in categories %}
            <li class="list-group-item">
                  <a href="{{ path('category', {'id':category.id}) }}">
                        {{ category.CategoryName }}
                  </a>
            </li>
    {% endfor %}


    {% for module in ModuleByCategory %}
       <div class="card" style="width: 18rem;">
                <li class="list-group-item etat">
                    <div class="circleGreen"></div>
                    <div class="circleOrange"></div>
                    <div class="circleRed"></div>
                </li>
                <img src="{{asset('img/module/' ~ module.image)}}"
                        class="card-img-top" alt="...">
                <div class="card-body">
                    <h5 class="card-title">Module n°{{module.number}}</h5>
                    <h5 class="card-title">{{module.name}}</h5>
                    <p class="card-text">
                        Type : {{module.category.CategoryName}}</p>
                    <p class="card-text">{{module.description}}</p>
                    <a href="{{path('edit', {id: module.id})}}"
                       class="btn btn-primary">Modifier</a>
                    <a href="{{path('delete', {id: module.id})}}"
                       class="btn btn-primary">Supprimer</a>
                </div>
            </div>
    {% endfor %}


{% endblock %}
```

Méthode Marche / arrêt

```php
    /**
     * @Route("/module/fonctionne/{id}", name="marche")
     */
    public function EtatMarche($id,Module $Module): Response
    {
        $entityManager = $this->getDoctrine()->getManager();
        $repository = $entityManager->getRepository(Module::class)->find($id);
        $repository->setEtatDeMarche("1");
        $entityManager->flush();

        $this->addFlash('sucess', 'Votre module fonctionne');
        return $this->redirecttoRoute('modules');
    }
    /**
     * @Route("/module/arret/{id}", name="arret")
     */
    public function EtatArret($id,Module $Module): Response
    {
        $entityManager = $this->getDoctrine()->getManager();
        $repository = $entityManager->getRepository(Module::class)->find($id);
        $repository->setEtatDeMarche("0");
        $entityManager->flush();

        $this->addFlash('danger', 'Votre module a bien été arrêté');
        return $this->redirecttoRoute('modules');

    }
```

Bouton Marche / Arrêt /Disfonction

Dans les cards de Modules voici ce que l'on fait on affiche soit le bouton arret soit le bouton marche en fonction de son état de fonctionnement.

```
{% if module.etatDeMarche == 1 %}
     <a href="{{path('arret', {id: module.id})}}"
             class="btn btn-primary">Arret</a>
    <a href="{{path('disfonction', {id: module.id})}}"
             class="btn btn-primary">Disfonction</a>
{% endif %}


{% if module.etatDeMarche == 0 %}
    <a href="{{path('marche', {id: module.id})}}"
             class="btn btn-primary">Marche</a>
    <a href="{{path('disfonction', {id: module.id})}}"
             class="btn btn-primary">Disfonction</a>
{% endif %}



{% if module.etatDeMarche == 2 %}
     <a href="{{path('marche', {id: module.id})}}"
             class="btn btn-primary">Marche</a>
     <a href="{{path('arret', {id: module.id})}}"
             class="btn btn-primary">Arret</a>
{% endif %}
```

Simulation disfonctionnement

Pour simuler un état de disfonctionnement , nous créons un bouton nommé Disfonction qui appelera la méthode qui va setter l'état de marche du module dans un état de disfonctionnement.

En ajoutant un message pour indiquer le changement d'état de disfonctionnement
Et rediriger vers la route actuel

```
/**
     * @Route("/module/disfonction/{id}", name="disfonction")
     */
    public function EtatDisfonction($id,Module $Module, Request $request): Res
ponse
    {

        $entityManager = $this->getDoctrine()->getManager();

        $repository = $entityManager->getRepository(Module::class)->find($id);
        $repository->setEtatDeMarche("2");
        dump($repository);

        $entityManager->flush();


       $entityManager->flush();

        $this->addFlash(
      'danger', 'Attention, votre module rencontre un disfonctionnement');

        $CurrentPage = $request->headers->get('referer');
        return $this->redirect($CurrentPage);
        //return $this->redirecttoRoute('modules');

    }
```

Voyant d'état

Après avoir fait la méthode on crée des voyants qui s'afficheront en fonction de l'état du module, a savoir Marche /Arrêt /Disfonctionnement.

```
{% for module in modules %}

          <div class="card cardModule col-lg-2" >

              {% if module.etatDeMarche == 1 %}
                  <div class="etattitle">
                      <li class=" etat">
                          <div class="circleGreen"></div>
                          <div class="circleVoid"></div>
                          <div class="circleVoid"></div>
                      </li>
                      <h6 class="card-title">Module n°{{ module.id}}</h6>
                  </div>
              {% endif %}
              {% if  module.etatDeMarche == 0 %}
                  <div class="etattitle">
                      <li class=" etat">
                          <div class="circleVoid"></div>
                          <div class="circleVoid"></div>
                          <div class="circleRed"></div>
                      </li>
                      <h6 class="card-title">Module n°{{ module.id}}</h6>
                  </div>
              {% endif %}
              {% if module.etatDeMarche == 2 %}
                  <div class="etattitle">
                      <li class=" etat">
                          <div class="circleVoid"></div>
                          <div class="circleOrange"></div>
                          <div class="circleVoid"></div>
                      </li>
                      <h6 class="card-title">Module n°{{ module.id}}</h6>
                  </div>
              {% endif %}
```

Historique de fonctionnement

On crée le controller et la page où sera afficher l'historique de fonctionnement.
php bin/console make:controller HistoriqueDeFonctionnement

Puis on crée l'entité **Historique** avec le champ **commentaire**
php bin/console make:entity
Class name of the entity to create or update (e.g. FiercePopsicle):
> **Historique**
New property name (press <return> to stop adding fields):
> **Commentaire**

Mise à jour de la base de donnée
php bin/console doctrine:schema:update --force

Pour ça on utilise nos méthode marche / arrêt / disfonctionnement afin d'envoyer à la base de données les infos dès qu'il y a un changement d'état
Dans **ModulesController.php**
On ajoute aux méthodes , le fait de créer un commentaire à chaque changement d'état de fonctionnement . Pour l'état de marche :

```
   /**
     * @Route("/module/fonctionne/{id}", name="marche")
     */
  public function EtatMarche($id, Module $Module, Request $request): Response
    {
        //-------------------------------------------------------
        $entityManager = $this->getDoctrine()->getManager();
        $repository = $entityManager->getRepository(Module::class)->find($id);
        $repository->setEtatDeMarche("1");
        //dump($repository);
        //-------------------------------------------------------
        $Commentaire = $entityManager->getRepository(
                        Module::class)->find($id);
        $idModuleCommentaire = $Commentaire->getId();
        $NameModuleCommentaire = $Commentaire->getName();

        $commentaire = new Historique();
        $commentaire-
>setCommentaire(" Le module n° $idModuleCommentaire : $NameModuleCommentaire
 fonctionne correctement ");
        $entityManager->persist($commentaire);

        //-------------------------------------------------------
        $entityManager->flush();
        $this->addFlash('success', 'Votre module fonctionne');

        $CurrentPage = $request->headers->get('referer');

        return $this->redirect($CurrentPage);
    }
```

Pour l'état d'arrêt :

```
/**
     * @Route("/module/arret/{id}", name="arret")
     */
    public function EtatArret($id, Module $Module, Request $request): Response
    {

        $entityManager = $this->getDoctrine()->getManager();
        $repository = $entityManager->getRepository(Module::class)->find($id);
        $repository->setEtatDeMarche("0");

        //-------------------------------------------------------------
        $Commentaire = $entityManager->getRepository(
                        Module::class)->find($id);

        $idModuleCommentaire = $Commentaire->getId();
        $NameModuleCommentaire = $Commentaire->getName();

        $commentaire = new Historique();
        $commentaire->setCommentaire
(" ⚠ Attention ! Le module n° $idModuleCommentaire : $NameModuleCommentaire s'
est arrêté ! ");
        $entityManager->persist($commentaire);

        //-------------------------------------------------------------


        $entityManager->flush();
        $this->addFlash('danger', 'Votre module a bien été arrêté');

        $CurrentPage = $request->headers->get('referer');

        return $this->redirect($CurrentPage);
        // return $this->redirecttoRoute('modules');

    }
```

Pour l'état de disfonctionnement :

```
/**
     * @Route("/module/disfonction/{id}", name="disfonction")
     */
    public function EtatDisfonction($id, Module $Module, Request $request): Re
sponse
    {

        $entityManager = $this->getDoctrine()->getManager();
        $repository = $entityManager->getRepository(Module::class)->find($id);
        $repository->setEtatDeMarche("2");


        //-------------------------------------------------------
        $Commentaire = $entityManager->getRepository(
                        Module::class)->find($id);

        $idModuleCommentaire = $Commentaire->getId();
        $NameModuleCommentaire = $Commentaire->getName();


        $commentaire = new Historique();
        $commentaire->setCommentaire
(" □ Le module n° $idModuleCommentaire : $NameModuleCommentaire est dans un ét
at de disfonctionnement");
        $entityManager->persist($commentaire);

        //-------------------------------------------------------

        $entityManager->flush();

        $this->addFlash(
      'danger', 'Attention, votre module rencontre un disfonctionnement');

        $CurrentPage = $request->headers->get('referer');
        return $this->redirect($CurrentPage);
    }
}
```

Affichage de l'historique dans twig :

```
{% extends 'base.html.twig' %}

{% block title %}Historique{% endblock %}

{% block body %}
    {# {% for commentaire in commentaires %}
       {{ commentaire.commentaire }} <br>
    {% endfor %} #}
    <div class="homeCover">
        <h1 class="col-lg-8">Historique de fonctionnement des modules : </h1>
    </div>

   <div class="container">
        <table class="table table-dark table-striped">
            <thead>
                <tr>
                    <th scope="col">n°</th>
                    <th scope="col">Commentaire</th>
                </tr>
            </thead>
            <tbody>
                {% for commentaire in commentaires %}
                    <tr>
                        <th scope="row">{{ commentaire.id }}</th>
                        <td>{{ commentaire.commentaire }}</td>
                    </tr>
                {% endfor %}
            </tbody>
        </table>
   </div>

{% endblock %}
```

Changement des champs du CRUD de module :

Cela permet d'afficher et de personnaliser les champs de chaque module dans le crud :

```php
<?php

namespace App\Controller\Admin;

use App\Entity\Module;
use EasyCorp\Bundle\EasyAdminBundle\Controller\AbstractCrudController;
use EasyCorp\Bundle\EasyAdminBundle\Field\AssociationField;
use EasyCorp\Bundle\EasyAdminBundle\Field\ImageField;
use EasyCorp\Bundle\EasyAdminBundle\Field\NumberField;
use EasyCorp\Bundle\EasyAdminBundle\Field\TextField;
use phpDocumentor\Reflection\Types\Float_;

class ModuleCrudController extends AbstractCrudController
{
    public static function getEntityFqcn(): string
    {
        return Module::class;
    }


    public function configureFields(string $pageName): iterable
    {
        return [

            TextField::new('name'),
            TextField::new('description'),
            NumberField::new('temperature'),
            NumberField::new('duree_fonctionnement'),
            NumberField::new('etat_de_marche'),
            ImageField::new('image')->setBasePath('img/module')-
>setUploadDir('public/img/module/'),
            AssociationField::new('Category','Categorie'),

        ];
    }

}
```

Changement des champs de Crud de historique :

Cela permet d'afficher et de personnaliser les champs de chaque commentraire dans le crud :

```php
<?php

namespace App\Controller\Admin;

use App\Entity\Historique;
use EasyCorp\Bundle\EasyAdminBundle\Controller\AbstractCrudController;
use EasyCorp\Bundle\EasyAdminBundle\Field\TextField;

class HistoriqueCrudController extends AbstractCrudController
{
    public static function getEntityFqcn(): string
    {
        return Historique::class;
    }


    public function configureFields(string $pageName): iterable
    {
        return [
           TextField::new('Commentaire')
        ];
    }

}
```

Changement du lien du logo du Dashboard :

Dans vendor/easycorp/easyadmin-bundle/src/ressources/views/
Cela permettra de revenir sur la page d'accueil plus facilement

Dans Layout :

```
  <div id="header-logo">
 {% block header_logo %}
    {# <a class="logo {{ ea.dashboardTitle|length > 14 ? 'logo-
long' }}" title="{{ ea.dashboardTitle|striptags }}" href="{{ path(ea.dashboard
RouteName) }}">
     {{ ea.dashboardTitle|raw }}
   </a> #}
   <a class="logo {{ ea.dashboardTitle|length > 14 ? 'logo-
long' }}" title="{{ ea.dashboardTitle|striptags }}"
            href="{{ path('home') }}">{{ ea.dashboardTitle|raw }}
        </a>
      {% endblock header_logo %}
</div>
```

Création de 5 modules en dur afin d'avoir un module de chaque catégorie pour le début du test .

```
$module = new Module();
        $module->setName('caméra');
        $module->setDescription('Caméra 360° connecté wifi');
        $module->setEtatDeMarche('2');
        $ModuleCategory = $this->getReference('stuff-4');
        $module->setCategory($ModuleCategory);
        $module->setImage('fixtures/Caméra.PNG');
        $manager->persist($module);

        $module1 = new Module();
        $module1->setName('Montre Galaxy Watch');
        $module1->setDescription('Montre de Michaël');
        $module1->setEtatDeMarche('1');
        $ModuleCategory = $this->getReference('stuff-0');
        $module1->setCategory($ModuleCategory);
        $module1->setImage('fixtures/Montre.PNG');
        $manager->persist($module1);

        $module2 = new Module();
        $module2->setName('Chauffage');
        $module2->setDescription('Chauffage de la chambre');
        $module2->setEtatDeMarche('1');
        $ModuleCategory = $this->getReference('stuff-1');
        $module2->setCategory($ModuleCategory);
        $module2->setImage('fixtures/Chauffage.PNG');
        $manager->persist($module2);

        $module3 = new Module();
        $module3->setName('Prise ');
        $module3->setDescription('Prise de l\'aquarium');
        $module3->setEtatDeMarche('0');
        $ModuleCategory = $this->getReference('stuff-2');
        $module3->setCategory($ModuleCategory);
        $module3->setImage('fixtures/Prise.PNG');
        $manager->persist($module3);


        $module4 = new Module();
        $module4->setName('Amazon echo dot 3');
        $module4->setDescription('assistant vocal du bureau');
        $module4->setEtatDeMarche('1');
        $ModuleCategory = $this->getReference('stuff-3');
        $module4->setCategory($ModuleCategory);
        $module4->setImage('fixtures/Vocal.PNG');
        $manager->persist($module4);
```

Bouton "Back-to-top"

Javascript

```
$(document).ready(function(){


    $("h1, p").delay("1000").fadeIn();

    $("#back-top").hide();


    $(function () {
        $(window).scroll(function () {
            if ($(this).scrollTop() > 200) {
                $('#back-top').fadeIn();
            } else {
                $('#back-top').fadeOut();

            }
        });


        $('a#back-top').click(function () {
            $('body,html').animate({
                scrollTop: 0
            }, 800);
            return false;
        });
    });


});
```

Dans le footer :

```
<a  id="back-top" href="#top"href="">
      <li class="round">
            <i class="fas fa-chevron-up"></i>
      </li>
</a>
```