

Prise en main de Symfony 5.4

Doctrine

Contenu



Objectifs :

Continuer le TD Doctrine Structure : Développer une petite application qui va gérer un les navires.
Versionner le projet
Créer des contrôleurs
Créer et valider des formulaires

Environnement :

- ✓ Serveur apache : celui de wamp
- ✓ Base de données : mysql en local (wamp)
- ✓ IDE : netbeans
- ✓ virtualHost : navire.sio
- ✓ PHP 8.1
- ✓ Symfony 5.4
- ✓ Mysql 8.xx



Vous allez maintenant développer l'application.

Partie 1 : USER STORY 1

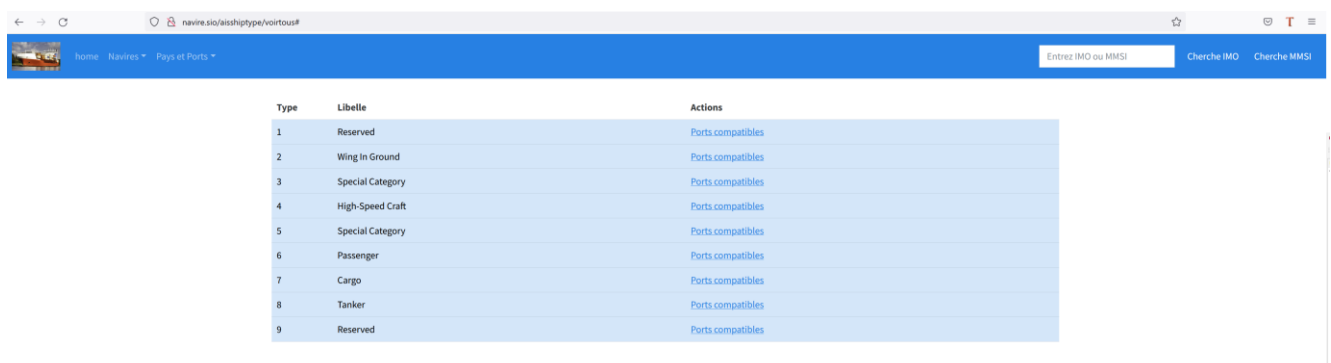
En tant que gestionnaire
je voudrais pouvoir lister les types de Navire
afin de pouvoir visualiser les ports où les navires de ce type sont susceptibles d'être accueillis

1. Lister les types de navire

Pour réaliser cette mission, vous aurez à :

- ✓ Créer le contrôleur AisShipTypeController
- ✓ Implémenter la méthode voirTous de la classe contrôleur AisShipTypeController
- ✓ Créer la classe formulaire AisShipTypeType
- ✓ Créer le template aishiptype\voirtous.html.twig

Résultat à obtenir :



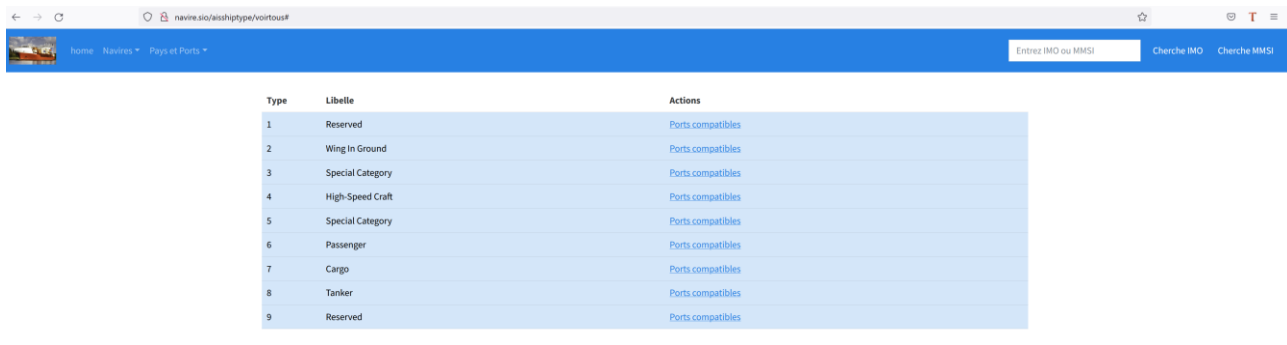
Type	Libelle	Actions
1	Reserved	Ports compatibles
2	Wing In Ground	Ports compatibles
3	Special Category	Ports compatibles
4	High-Speed Craft	Ports compatibles
5	Special Category	Ports compatibles
6	Passenger	Ports compatibles
7	Cargo	Ports compatibles
8	Tanker	Ports compatibles
9	Reserved	Ports compatibles

Contrôleur : AisShipTypeController :

- ✓ Routes préfixées par : *aishiptype/*

✓ Noms des routes préfixés par *aisshiptype_*

Exemple de rendu :



Type	Libelle	Actions
1	Reserved	Ports compatibles
2	Wing In Ground	Ports compatibles
3	Special Category	Ports compatibles
4	High-Speed Craft	Ports compatibles
5	Special Category	Ports compatibles
6	Passenger	Ports compatibles
7	Cargo	Ports compatibles
8	Tanker	Ports compatibles
9	Reserved	Ports compatibles

2. La méthode voirTous de la classe AisShipTypeController

Elle doit s'exécuter sur la route *aisshiptype/voirtous* dont le nom est *aisshiptype_voirtous*

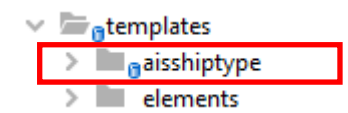
Elle va récupérer tous les types de navires et appeler le template *aisshiptype/voirtous.html.twig*

```
#[Route('/aisshiptype', name: 'aisshiptype')]
class AisShipTypeController extends AbstractController
{
    #[Route('/voirtous', name: 'voirtous')]
    public function voirTous(AisShipTypeRepository $repoAisShipType): Response
    {
        $aisShipTypes=$repoAisShipType->findAll();
        return $this->render('aisshiptype/voirtous.html.twig', [
            'aisShipTypes' => $aisShipTypes,
        ]);
    }
}
```

3. la classe formulaire AisShipTypeType

Vous la créez avec la commande `symfony make:form`

Vous Renommer le dossier templates :



Elle sera très simple car susceptible d'être utilisée dans plusieurs environnements :

```
public function buildForm(FormBuilderInterface $builder, array $options): void
{
    $builder
        ->add('aisShipType', TextType::class)
        ->add('libelle' , TextType::class)
    ;
}
```

4. le template aisshiptype\voirtous.html.twig

```
{% extends 'base.html.twig' %}
{% block title %}Ship types{% endblock %}

{% block body %}
    <div class="container">
        <table class="table table-hover">
            <thead>
                <tr>
                    <th scope="col" style="width: 100px;">Type</th>
                    <th scope="col">Libelle</th>
                    <th scope="col">Actions</th>
                </tr>
            </thead>
            <tbody>
                {% for type in aisShipTypes %}
                    <tr class="table-primary">
                        <td>{{ type.aisshiptype }}</td>
                        <td>{{ type.libelle }}</td>
                        <td><a href="#">Ports compatibles</a></td>
                    </tr>
                {% endfor %}
            </tbody>
        </table>
    </div>
{% endblock %}
```

Pour l'instant le lien est mort

En cliquant sur le lien :

Indicateur	Nom	Pays	Voie
KRNP	Busan New Port	Corée du Sud	Corée du Sud
CUCC	Cayo Coco	Cuba	Cuba
GIBB	Gibraltar	Gibraltar	Gibraltar
TRST	Istanbul	Turquie	Turquie
SKOP	Koper	Slovenie	Slovenie
FRLEH	Le Havre	France	France
USLDB	Long Beach	Etats-Unis	Etats-Unis
USMIA	Miami	Etats-Unis	Etats-Unis
CNNSA	Nansha	Chine	Chine
USNYC	New York	Etats-Unis	Etats-Unis
CNNGB	Ningbo	Chine	Chine
NGPHC	Port Harcourt	Nigeria	Nigeria
BONAS	Port de NASSAU	Bahamas	Bahamas
ITRAN	Ravenna	Italie	Italie

5. La liste des ports compatibles

Vous aurez à

Rajouter la méthode contrôleur *portsCompatibles* dans la classe *AisShipTypeController*.

```
#[Route('/portscompatibles', name: 'portscompatibles')]
public function portsCompatibles(Request $request, AisShipTypeRepository $repo ): Response {
    $aisShipType= $repo->find($request->get('id'));
    return $this->render('aisshiptype/portscompatibles.html.twig', [
        'aisShipType' => $aisShipType,
    ]);
}
```

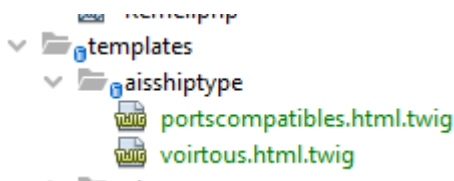
✓ Modifier le lien *Actions* dans la vue *voirtous.html.twig*

```
<td>{{ type.libelle }}</td>
<td> <a href= "{{ path('aisshiptype portscompatibles', {id: type.id}) }}">Ports compatibles</a></td>
```



Ce lien va chercher la route *aisshiptype_portscompatibles* en passant en GET l'id du type de navire recherché.

Rajouter la vue *portscompatibles.html.twig* :



Liste triée alphabétiquement sur le port par twig

Résultats attendus :

Entrez IMO ou MMSI

Ports compatibles
Type de navire : 1 Reserved
Aucun port disponible pour ce type de navire

Ou

Accueil Navires Pays et Ports Entrez IMO ou MMSI Cherche IMO

Ports compatibles
Type de navire : 3 Special Category

Indicatif	Nom	Pays	Voir
GIGIB	Gibraltar	Gibraltar	Gibraltar
FRYNE	La Seyne Sur Mer	France	France
FRLEH	Le Havre	France	France
PTLIS	Lisboa	Portugal	Portugal
USNYC	New York	Etats-Unis	Etats-Unis
CNSHG	Shanghai	Chine	Chine
FRTLN	Toulon	France	France

Pour l'instant le lien est mort



Vous mettrez à jour les liens du menu !!!

Partie 2 : USER STORY 2

En tant que gestionnaire
je voudrais pouvoir lister les navires
afin de pouvoir modifier certaines caractéristiques.



Cette user story va se faire en deux étapes :
Lister les navires en fournissant le lien pour la modification
Afficher un navire en mode édition

Vous aurez à créer le Contrôleur : NavireController :

- ✓ Routes préfixées par : *navire/*
- ✓ Noms des routes préfixés par *navire_*

1. Lister les navires

Nom de la méthode Contrôleur : voirTous
Nom du template : navire\voirtous.html.twig

Résultat final :

Pays et Ports ▼

Entrez IMO ou MMSI

Chercher

IMO	Nom	MMSI	Type	Destination	ETA	Actions
9007491	CLUB MED 2	227194000	Tanker	FRTLN (Toulon)	26-11-2022 20:40	Editer
9744001	SYMPHONY OF THE SEAS	311000660	Passenger	CUCCC (Cayo Coco)	27-12-2022 03:10	Editer
9193680	MSC TIA	255806080	Cargo	SIKOP (Koper)	31-12-2022 02:40	Editer
9502910	MAERSK EMERALD	563090400	Cargo	USLGB (Long Beach)	21-12-2022 01:40	Editer
9755933	MSC DIANA	636017433	Cargo	CNSHG (Shangai)	29-12-2022 11:40	Editer
9280366	CONFIDENCE	636012164	Tanker	NGPHC (Port Harcourt)	05-01-2023 12:40	Editer

Pour l'instant le lien est mort

2. Modification d'un navire

Vous aurez à :

- ✓ Créer le formulaire : classe *NavireType*
- ✓ Créer la méthode contrôleur *editer* de la class *NavireController*
- ✓ Créer la vue twig *edit.html.twig*

Le résultat :

Pays et Ports ▼

E

IMO

9007491

Nom du Navire

CLUB MED 2

MMSI

227194000

Indicatif VHF

FNIR

Longueur

187

Largeur

20

Tiran d'eau

5.2

Type de navire

Tanker ▼

Pavillon

France ▼

Destination

Toulon ▼

Estimated Time Arrival

26 / 11 / 2022 20 : 40 : 38

retour

Modifier

Classe *NavireType* :

```
public function buildForm(FormBuilderInterface $builder, array $options) {
    $builder
        ->add('imo', TextType::class)
        ->add('nom', TextType::class)
        ->add('mmsi', TextType::class)
        ->add('indicatifAppel', TextType::class)
        ->add('eta', DateTimeType::class, ['widget'=>'single_text'])
        ->add('longueur', IntegerType::class)
        ->add('largeur', IntegerType::class)
        ->add('tirantdeau', NumberType::class, array(
            'scale' => 1,
        ))
        ->add('aisShipType', EntityType::class, [
            'class'=>AisShipType::class,
            'choice_label'=>'libelle'
        ])
        ->add('pavillon', EntityType::class, [
            'class'=> Pays::class,
            'choice_label'=>'nom'
        ])
        ->add('destination', EntityType::class, [
            'class'=> Port::class,
            'choice_label'=>'nom'
        ])
    }
}
```



Observez les différents types de sous-formulaires et leurs paramètres. N'oubliez pas les use qui vont bien !!!

la méthode contrôleur `edit` de la class `NavireController`

```
#[Route('/editor/{id}', name: 'editor')]
public function edit(int $id, Request $request, NavireRepository $repoNavire, EntityManagerInterface $em): Response {
    $navire= $repoNavire->find($id);

    $form = $this->createForm(navireType::class, $navire);
    $form->handleRequest($request);

    if ($form->isSubmitted() && $form->isValid()) {
        $navire = $form->getData();
        $em->persist($navire);
        $em->flush();
        return $this->redirectToRoute('home');
    }
    return $this->render('navire/edit.html.twig', [
        'form' => $form->createView(),
    ]);
}
```



Pas de nouveautés ici :
On récupère l'objet navire à éditer,
on l'affiche,
on récupère les informations
et on persiste en BD

la vue twig `edit.html.twig` (extrait...)

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

{% block title %}{{ parent() }}Edit Navire!{% endblock %}

{% block body %}
<div class="container">
  {{ form_start(form) }}
  <div class="row">
    <div class="col-2" id="some-custom-id">
      {{ form_label(form.imo, 'IMO') }}
      {{ form_widget(form.imo, {'attr': {'placeholder': 'IMO Number', 'readonly': 'true'}}) }}
    </div>

    <div class="col-2" id="some-custom-id">
      {{ form_label(form.nom, 'Nom du Navire') }}
      {{ form_widget(form.nom, {'attr': {'placeholder': 'Saisir le nom du navire'}}) }}
    </div>
  </div>
  <div class="row">
    <div class="col-2" id="some-custom-id">
      {{ form_label(form.mmsi, 'MMSI') }}
      {{ form_widget(form.mmsi, {'attr': {'placeholder': 'MMSI Number'}}) }}
    </div>
  </div>
  {{ form_end(form) }}
</div>
```

....

```
<div class="row">
  <div class="col-2" id="some-custom-id">
    {{ form_label(form.destination, 'Destination') }}
    {{ form_widget(form.destination) }}
  </div>
  <div class="col-2" id="some-custom-id">
    {{ form_label(form.eta, 'Estimated Time Arrival') }}
    {{ form_widget(form.eta) }}
  </div>
</div>
<br>
<a type='button' class='btn btn-success' style='width:100px' href="{{ path('navire_voirtous') }}"> retour </a>
<button type='submit' style='width:100px' class='btn btn-success'>Modifier</button>
{{ form_end(form) }}
{% endblock %}
```



Pourquoi a-t-on prévu des attributs placeholder ?
Pourquoi le champs IMO est en lecture seule ?

Pourquoi a-t-on mis un bouton retour qui redirige vers la liste des navires ?

Bonus : afficher la position du navire en temps réel :



L'essentiel est d'afficher la position du navire
Vous afficherez les données en temps réel alors que les
données de la BD ne sont pas à jour !!!
(il fallait payer)

navire.sio/navire/editer/1

Pays et Ports ▾ Entrez IMO ou MMSI

IMO	Nom du Navire	
9007491	CLUB MED 2	
MMSI	Indicatif VHF	
227194000	FNIR	
Longueur	Largeur	Tirant d'eau
187	20	5.2
Type de navire	Pavillon	
Tanker ▾	France ▾	
Destination	Estimated Time Arrival	
Toulon ▾	26 / 11 / 2022 20 : 40 : 38	

retour Modifier

VesselFinder.com Embed this map | iOS | Android

10 nm

Next port GP FAE
ETA Mar 02, 08:00
IMO 9007491
Speed 5.9 kn

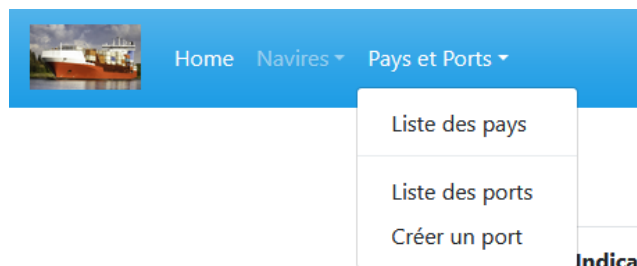
Track More info

Lat: 16.04767 Lon: -62.64768
16° 02' 52" -62° 38' 51"

© OpenStreetMap contributors

Partie 3 : USER STORY 3

En tant que responsable
je voudrais pouvoir créer des ports
afin de pouvoir enregistrer le trafic maritime vers le nouveau port



Vous aurez à créer

- ✓ Le contrôleur PortController
- ✓ Le formulaire portType
- ✓ Le template port/edit.html.twig

L'interface pourrait ressembler à ceci :

Indicatif

Nom du port

Indicatif international

Saisir le nom du port

Pays

Types de navires acceptés

Aruba (Ile d')

☐ Reserved
☐ Wing In Ground
☐ Special Category
☐ High-Speed Craft
☐ Special Category
☐ Passenger
☐ Cargo
☐ Tanker
☐ Petit bateau

Créer



Vous n'aurez rien à faire à ce niveau-là pour saisir les escales et les navires attendus !

Voici à quoi pourrait ressembler le formulaire PortType :

```
public function buildForm(FormBuilderInterface $builder, array $options): void {
    $builder
        ->add('nom', TextType::class)
        ->add('indicatif', TextType::class)
        ->add('pays', EntityType::class, [
            'class' => Pays::class,
            'choice_label' => 'nom',
            'expanded' => false,
            'multiple' => false,
        ])
        ->add('types', EntityType::class, [
            [
                'class' => AisShipType::class,
                'choice_label' => 'libelle',
                'expanded' => true,
                'multiple' => true,
            ]
        ])
    };
}
```

Et la classe portController :

```
#[Route('/creer', name: 'creer')]
public function creer(Request $request, EntityManagerInterface $manager): Response {
    $port = new Port();
    //$Paysrepo->findAll()
    $form = $this->createForm(PortType::class, $port);
    $form->handleRequest($request);
    if ($form->isSubmitted() && $form->isValid()) {
        $manager->persist($port);
        $manager->flush();
        return $this->redirectToRoute('home');
    }
    return $this->render('port/edit.html.twig', [
        'form' => $form->createView(),
    ]);
}
```

Quant au template twig, en voici un extrait :

```
<div class="row">
    <div class="col-2" id="some-custom-id">
        {{ form_label(form.pays, 'Pays') }}
        {{ form_widget(form.pays) }}
    </div>

    <div class="col-2" id="some-custom-id">
        {{ form_label(form.types, 'Types de navires acceptés') }}

        {{ form_widget(form.types) }}
    </div>
</div>
</br>
<button type='submit' class='btn btn-success'>Créer</button>
{{ form_end(form) }}
{% endblock %}
```

Vous allez donc créer le port suivant :

Indicatif	Nom du port
<input type="text" value="ESBCN"/>	<input type="text" value="BARCELONA"/>
Pays	Types de navires acceptés
<input type="text" value="Espagne"/>	<input type="checkbox"/> Reserved <input type="checkbox"/> Wing In Ground <input type="checkbox"/> Special Category <input checked="" type="checkbox"/> High-Speed Craft <input type="checkbox"/> Special Category <input checked="" type="checkbox"/> Passenger <input checked="" type="checkbox"/> Cargo <input checked="" type="checkbox"/> Tanker <input type="checkbox"/> Reserved

Cliquez sur Créer et revenez sur votre home page.



Voyez maintenant la base de données

	25	Qingdao	CNQDG	29
	26	Barcelona	ESBCN	54
*	NULL	NULL	NULL	NULL

Le port a bien été créé avec l'id 26.

Vérifiez maintenant les types de navires acceptés avec la requête suivante :

```
3 • select * from naviresymfo.porttypecompatible where idport=26;
```

Result Grid

	idport	idaisshtype
▶	26	4
	26	6
	26	7
	26	8
*	NULL	NULL



C'est exactement ce qui avait été saisi !!!
Cool !!!...



Il y a quand même un souci : vous avez remarqué que la liste des pays n'est pas triée ... C'est gênant quand on doit choisir UN pays dans autant de pays non triés

Pays

- Aruba (Ile d')
- Aruba (Ile d')
- Angola
- Anguilla (Ile d')
- Albanie
- Aland (Iles d')
- Antilles Néerlandaises
- Emirats Arabes Unis
- Argentine
- Kerguelen (Iles)

Pour avoir une liste triée, vous allez donc exploiter tout le potentiel de symfony

L'idée est de récupérer la liste des pays triée sur le nom du pays..... et là on est obligés de passer par une requête DQL juste pour effectuer un tri.....

Vous allez donc modifier :

- ✓ Le repository PaysRepository
- ✓ Le formulaire PortType

Le repository PaysRepository en rajoutant la méthode getPaysTrieSurNom(). On n'a pas le choix, on doit passer par l'API QueryBuider.

```
public function getPaysTrieSurNom()
{
    return $this->createQueryBuilder('p')
        ->orderBy('p.nom', 'ASC');
}
```

Le formulaire PortType : au lieu de récupérer l'entity Pays comme elle se présente dans la BD, on va récupérer l'entity ... triée sur le nom.



On va intégrer une requête DQL directement dans l'objet FormBuilder :

```
->add('pays', EntityType::class, [
    'class' => Pays::class,
    'choice_label' => 'nom',
    'expanded' => false,
    'multiple' => false,
    'query_builder' => function (PaysRepository $paysRepo) {
        return $paysRepo->createQueryBuilder('p')
            ->orderBy('p.nom', 'ASC');
    },
])
->add('types', EntityType::class,
```

Partie 4 : USER STORY 4

En tant que gestionnaire,
je voudrais pouvoir visualiser les types de navires susceptibles d'être accueillis dans un port
afin de vérifier les itinéraires des navires

1. Liste des ports

Vous allez commencer par visualiser la liste des ports :

URL : <http://navire.sio/port/voirtous>

Contrôleur : PortController

Nom de la route : port_voirtous



La liste des ports sera triée **par twig** par ordre croissant des noms

indicatif	nom	indicatif pays	pays	Actions
ESBCN	Barcelona	ESP	Espagne	Types de navires compatibles
KRBNP	Busan New Port	KOR	Corée du Sud	Types de navires compatibles
CUCCC	Cayo Coco	CUB	Cuba	Types de navires compatibles
GIGIB	Gibraltar	GIB	Gibraltar	Types de navires compatibles
TRIST	Istanbul	TUR	Turquie	Types de navires compatibles
SIKOP	Koper	SVN	Slovénie	Types de navires compatibles
FRYNE	La Seyne Sur Mer	FRA	France	Types de navires compatibles
FRLEH	Le Havre	FRA	France	Types de navires compatibles
PTLIS	Lisboa	PRT	Portugal	Types de navires compatibles
BSCOC	Little Stirrup Cay	BHS	Bahamas	Types de navires compatibles

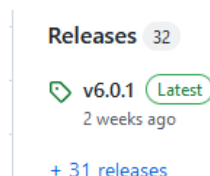


Vue la quantité de ports dans le monde, en plus d'avoir la liste triée sur le nom, on pourrait faire de la pagination !

2. Pagination

Nous allons pour cela utiliser le bundle *Knplabs/KnpPaginatorBundle*

En allant sur le dépôt de ce bundle, vous voyez que la version 6 existe.



Cette version ne fonctionne pas en symfony 5 mais elle fonctionne en symfony 6.

v6.0.0

- Remove all previously deprecated code
- Remove support for Bootstrap <4
- Drop support for php 7 and Symfony 4/5

En parcourant les changelog, on déduit qu'il faut la version 5.4 minimum qui prend en compte PHP8 et symfony 5 :

v5.4.0

- Added support for php 8
- Removed support for EOled php 7.2
- Added support for Material design in sortable

Vous installerez donc ce bundle en précisant la version minimum :

symfony composer require knplabs/knp-paginator-bundle:5.4

Il sera donc nécessaire d'installer la version 5

<https://github.com/KnpLabs/KnpPaginatorBundle/releases?page=2>

```
Generating optimized autoload files
74 packages you are using are looking for funding.
Use the `composer fund` command to find out more!

Symfony operations: 2 recipes (bd4c892d5f19f2c983f32c46cabe8e69)
- Configuring symfony/translation (>=5.3): From github.com/symfony/recipes:main
- Configuring knplabs/knp-paginator-bundle (>=v5.4.0): From auto-generated recipe
Executing script cache:clear [OK]
Executing script assets:install public [OK]

What's next?

Some files have been created and/or updated to configure your new packages.
Please review, edit and commit them: these files are yours.

No security vulnerability advisories found
```

Vous commencerez à modifier votre classe PortController :

```
[Route('/voirtous', name: 'voirtous')]
public function voirTous(Request $request, EntityManagerInterface $manager, PortRepository $repo, PaginatorInterface $paginator): Response {
    $data = $repo->findAll();
    $ports = $paginator->paginate($data,
        $request->query->getInt('page', 1),
        10
    );
    return $this->render('port/voirTous.html.twig', [
        'ports' => $ports,
    ]);
}
```

```
$data = $repo->findAll();
$ports = $paginator->paginate($data,
    $request->query->getInt('page', 1),
    10
);
```

Les arguments de la méthode paginate de la classe PaginatorInterface

- ✓ La source de données
- ✓ Le numéro de page à afficher, par défaut c'est 1 : la première page
- ✓ Le nombre d'éléments par page

Vous modifierez ensuite le template voir tous.html.twig

```
</tbody>
</table>
<div class="d-flex justify-content-center">
    {% do ports.setPageRange(2) %}
    {{ knp_pagination_render(ports, 'blocs/pagination.html.twig') }}
</div>
{% else %}
    <h2 class="h3 mb-3 font-weight-normal text-center">Pas de ports pour ce pays</h2>
{% endif %}
```

Et enfin vous ajouterez le dossier bloc dans le dossier templates dans lequel vous créerez le fichier pagination.html.twig que vous trouverez sur moodle et que vous étudierez.

Testez et naviguez dans les pages.:

Liste des ports

indicatif	nom	indicatif pays	pays	Actions
CUCC	Cayo Coco	CUB	Cuba	Types de navires compatibles
TRIST	Istanbul	TUR	Turquie	Types de navires compatibles
FRYNE	La Seyne Sur Mer	FRA	France	Types de navires compatibles
FRLEH	Le Havre	FRA	France	Types de navires compatibles
PTLIS	Lisboa	PRT	Portugal	Types de navires compatibles
BSCOC	Little Stirrup Cay	BHS	Bahamas	Types de navires compatibles
USMIA	Miami	USA	Etats-Unis	Types de navires compatibles
BSNAS	Port de NASSAU	BHS	Bahamas	Types de navires compatibles
PTSET	Setubal	PRT	Portugal	Types de navires compatibles
FRTLN	Toulon	FRA	France	Types de navires compatibles

« Previous 1 2 Next »



C'est super, mais il y a un souci.... Les ports sont triés mais seulement à l'intérieur d'une page, pas dans la globalité.

Nous allons de voir modifier pas mal de choses pour avoir un tri global....

L'idée est de créer une requête DQL triée dans le repository du port et ensuite de faire ce que l'on a fait déjà faire ...

Création de la requête DQL : Nous allons créer la méthode `getPortTrieSurNom` dans la classe `PortRepository` :

```
class PortRepository extends ServiceEntityRepository {

    public function __construct(ManagerRegistry $registry) { ...3 lines }

    /**
     * Fonction qui va trier les ports sur le nom du port
     * @return array : tableau des ports trié sur le nom du port
     */
    public function getPortTrieSurNom() : Array {
        return $this->createQueryBuilder('p')
            ->OrderBy('p.nom', 'ASC')
            ->getQuery()->getResult();
    }
}
```

Le tri dans le template `port/voirtous.html.twig` devient inutile puisque les données arrivent triées directement sur le nom du port :

```
<tbody>
    {% for port in ports %}
        <tr class="table-primary">
            <td>{{ port.indicatif }}</td>
```

Testez : <http://navire.sio/port/voirtous>



Vu le nombre de ports dans le monde il peut être judicieux de trier le résultat en premier sur le nom du pays et en second sur le nom du port

A vous de trouver :

Liste des ports

indicatif	nom	indicatif pays	pays	Actions
BSCOC	Little Stirrup Cay	BHS	Bahamas	Types de navires compatibles
BSNAS	Port de NASSAU	BHS	Bahamas	Types de navires compatibles
CNNSA	Nansha	CHN	Chine	Types de navires compatibles
CNNGB	Ningbo	CHN	Chine	Types de navires compatibles
CNQDG	Qingdao	CHN	Chine	Types de navires compatibles
CNSHG	Shangai	CHN	Chine	Types de navires compatibles
CNYTN	Yantian	CHN	Chine	Types de navires compatibles
CUCCC	Cayo Coco	CUB	Cuba	Types de navires compatibles
ESBCN	Barcelona	ESP	Espagne	Types de navires compatibles
FRYNE	La Seyne Sur Mer	FRA	France	Types de navires compatibles

« Previous 1 2 Next »

3. Les types compatibles

Le lien (bouton *Types de navires compatibles*), va appeler la route *port_typescompatibles* du contrôleur *PortController*. Faudra lui passer, en GET l'id du port afin de pouvoir réaliser la recherche idoine.

La syntaxe :

```
href= "{ path('port_typescompatibles', {id: port.id}) }" } >Types de
```

Ce paramètre crée une variable GET dans l'objet Request.

Nous allons donc créer la méthode contrôleur *typesCompatibles* dans la classe *PortController*

Cette méthode récupérera l'objet port de la classe Port qui fournira la collection des types compatibles :

```
#[Route("/typescompatibles", name: 'typescompatibles')]
public function typesCompatibles(Request $request, PortRepository $repoPort ): Response {
    $port= $repoPort->find($request->get('id'));
    return $this->render('port/typescompatibles.html.twig', [
        'port' => $port,
    ]);
}
```

Récupération de l'id du port passé en GET

Enfin, la vue twig *port/typescompatibles.html.twig* :



Exercice:

Vous la ferez en autonomie.

Voici un exemple, votre template devra afficher strictement les mêmes informations.

Exemple :

indicatif	nom	indicatif pays	pays	Actions
FRLEH	Le Havre	FRA	France	Types de navires compatibles
EDTIN	Toulon	FRA	France	

Types et Ports ▾ Chercher

Types compatibles
Port : FRLEH - Le Havre - France

Id	Nom
3	Special Category
4	High-Speed Craft
5	Special Category
6	Passenger
7	Cargo
8	Tanker



Le résultat est trié sur l'id

Résultat alternatif :

Types compatibles
Port : CNQDG - Qingdao - Chine

Aucun type compatible avec ce port

Partie 5 : LA BARRE DE RECHERCHE

Lors du dernier sprint, une nouvelle User Story a été créée avec une priorité très haute. Vous êtes chargé de mettre en place cette user story :

En tant que gestionnaire
je voudrais pouvoir rechercher un navire par son numéro IMO ou son indicatif MMSI
afin de pouvoir récupérer ses informations et le localiser sur la carte.

Vous allez donc développer le code de la zone de recherche de la navbar :

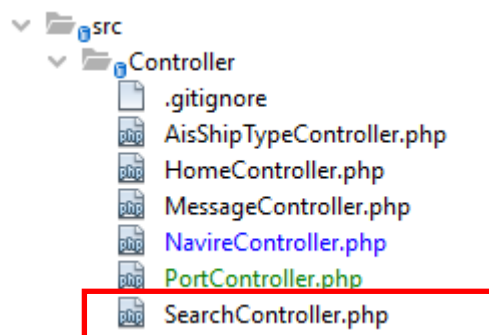
Chercher IMO Chercher MMSI

Pour cela, vous allez :

- ✓ Créer le contrôleur SearchController
- ✓ Créer le template éléments/searchbar.html.twig
- ✓ L'intégrer dans le template éléments/navbar.html.twig

4. Le contrôleur

Créer le contrôleur SearchController avec la commande `symfony make :controller`



N'oubliez pas ... toutes les routes de ce contrôleur
Commencent par /search/xxx
Se nomment search_xxx

Ce contrôleur aura 2 méthodes :

- ✓ La méthode `searchBar()` qui va construire le formulaire de recherche
- ✓ La méthode `handleSearch()` qui va exploiter le retour du formulaire quand l'utilisateur aura cliqué sur l'un des 2 boutons.

La méthode `searchBar`

```
public function searchBar() {
    $form = $this->createFormBuilder()
        ->setAction($this->generateUrl("search_handlesearch"))
        ->add('cherche', TextType::class)
        ->add('envoiimo', SubmitType::class)
        ->add('envoimmsi', SubmitType::class)
        ->getForm();
    return $this->render('elements/searchbar.html.twig', [
        'formSearch' => $form->createView()
    ]);
}
```

.. il s'agit seulement ici de générer le formulaire et d'appeler le template `éléments\searchbar.html.twig`

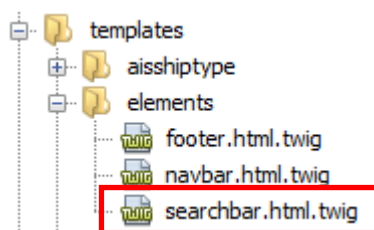
La méthode `handleSearch()`

Elle correspondra au contrôleur qui s'exécutera sur la route `/search/handlesearch`
Cette méthode va :

- ✓ Récupérer la valeur à rechercher (imo ou mmsi)
- ✓ Tester lequel des 2 boutons submit a envoyé le formulaire
- ✓ Effectuer la recherche sur le critère recherché *findOneByimo* ou *findOneBymmsi*
- ✓ Afficher le navire trouvé ou revenir sur la homepage avec un message de navire nont trouvé

```
#[Route('/handlesearch', name: 'handlesearch')]
public function handleSearch(Request $request, NavireRepository $repoNavire): Response {
    $valeur = $request->request->get('form')['cherche'];
    if (isset($request->request->get('form')['envoiimo'])) {
        $messageNotFound = "Le navire d'IMO " . $valeur . " n'existe pas.";
        $navire = $repoNavire->findOneByimo($valeur);
    } else {
        $messageNotFound = "Le navire de MMSI " . $valeur . " n'existe pas.";
        $navire = $repoNavire->findOneBymmsi($valeur);
    }
    if ($navire != null) {
        $url = $this->generateUrl('navire_editor', ["id" => $navire->getId()]);
    } else {
        $this->addFlash('notification', $messageNotFound);
        $url = $this->generateUrl('home');
    }
    return $this->redirect($url);
}
```

5. le template



Il s'agit d'un simple formulaire :

```
<div class="form-inline my-2">
    {% if is_granted('IS_AUTHENTICATED_FULLY') %}
        <h5>{{ app.user.prenom }} {{ app.user.nom }} </h5>
    {% endif %}
    {{ form_start(formSearch) }}

    {{ form_widget(formSearch.cherche, {'attr': {'placeholder': 'Entrez IMO or MMSI'}}) }}
    {{ form_widget(formSearch.envoiimo, {'label': 'Cherche IMO' }) }}
    {{ form_widget(formSearch.envoimmsi, {'label': 'Cherche MMSI' }) }}

    {{ form_end(formSearch) }}
</div>
```

6. L'intégration du formulaire de recherche dans la navbar :

Vous utiliserez la fonction render de twig pour appeler le template de recherche.
Vous modifierez donc le code à la fin du code de la navbar :

```

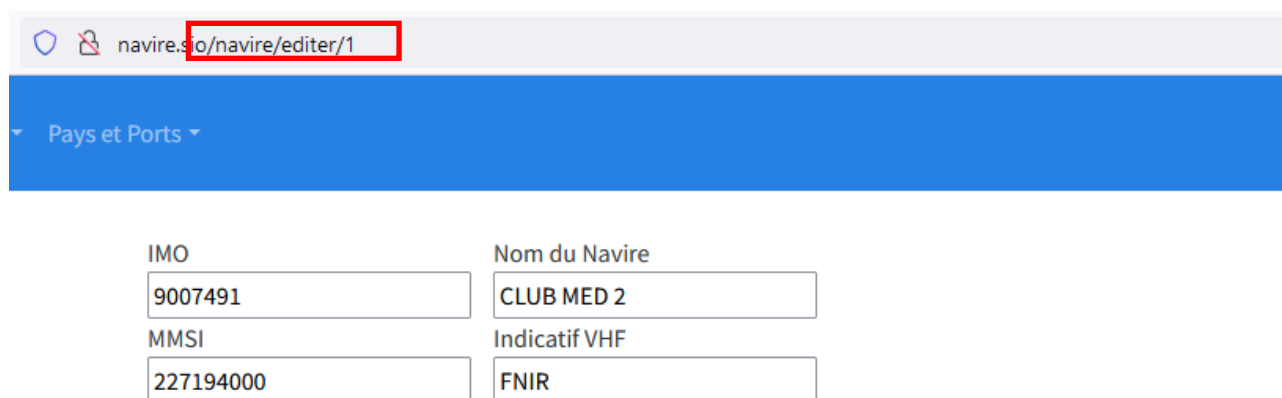
        </ul>
    </div>
    {{ render(controller('App\\Controller\\SearchController::searchBar')) }}
</div>
</nav>

```

Et vous pouvez tester sur le numéro IMO :



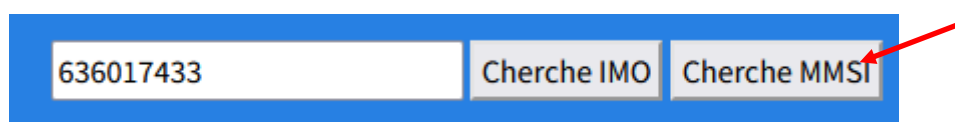
Résultat :



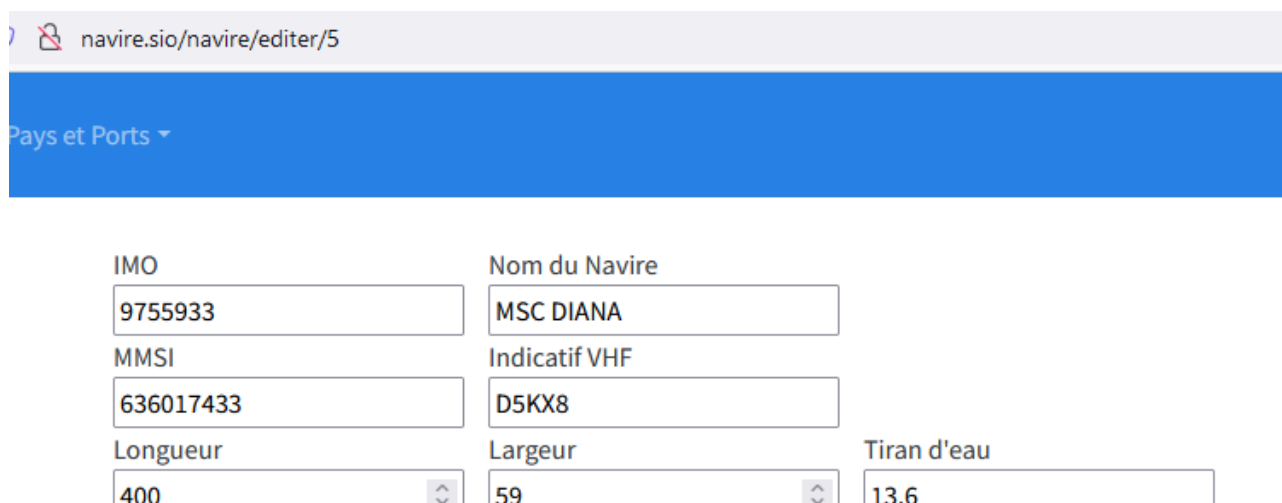
IMO	Nom du Navire
9007491	CLUB MED 2

MMSI	Indicatif VHF
227194000	FNIR

Et sur l'indicatif MMSI :



Et le résultat :



IMO	Nom du Navire
9755933	MSC DIANA

MMSI	Indicatif VHF
636017433	D5KX8

Longueur	Largeur	Tiran d'eau
400	59	13.6

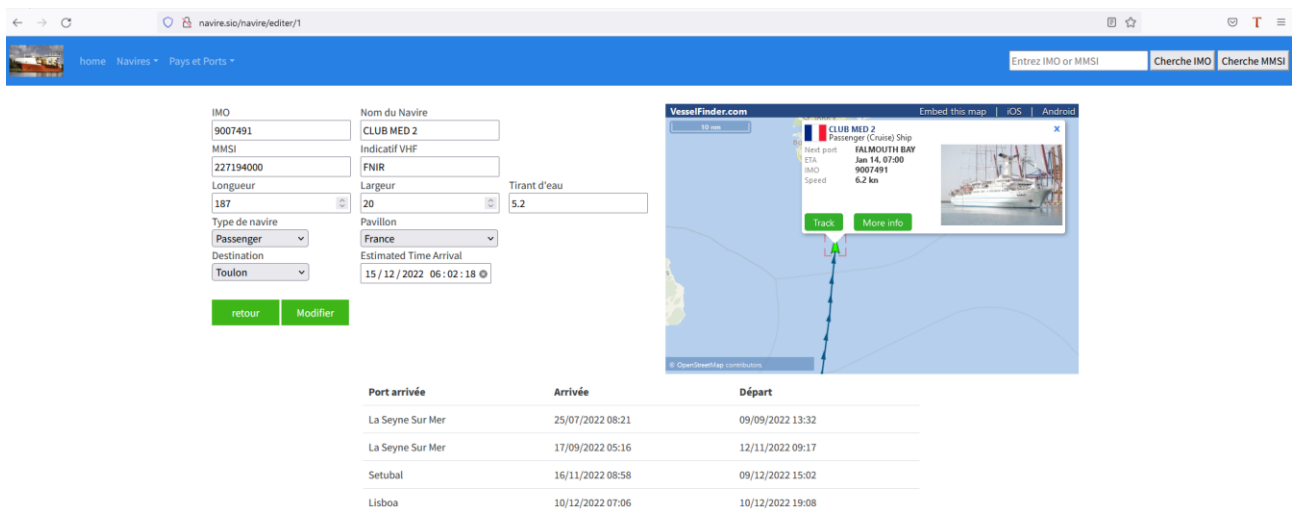
Et si le navire n'existe pas :



7. Suite des user stories

Les user stories suivantes seront à faire en autonomie, l'interface est libre.

En tant que gestionnaire
je voudrais connaître l'historique des escales d'un navire
afin de pouvoir effectuer des statistiques sur ses destinations et garder l'historique de ses routes.



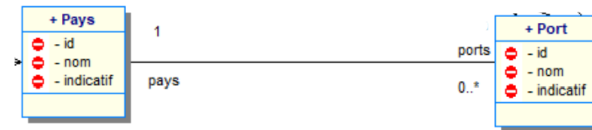
Modifiez juste le template twig voir tous.html.twig
Utilisez tous les attributs de la classe Navire Dont les relations

Et les nouvelles :

En tant que responsable
Je voudrais connaître le nombre de ports par pays
Afin d'étudier la faisabilité d'une liaison



Afin de simplifier la programmation, je vous conseille ici de modifier la relation entre port et pays et de la rendre bidirectionnelle en plus cela n'aura aucune incidence sur la BD



Exemple : <http://navire.sio/pays/voirtous?page=3>

Type	Libelle	Nombre de ports	Ports
BMU	Bermudes (Iles)	0	Liste des ports
BOL	Bolivie	0	Liste des ports
BRA	Brésil	0	Liste des ports
BRB	Barbade (Ile de La)	0	Liste des ports
BRN	Brunei	0	Liste des ports
CAN	Canada	0	Liste des ports
CHE	Suisse	0	Liste des ports
CHL	Chili	0	Liste des ports
CHN	Chine	5	Liste des ports
CIV	Côte d'Ivoire	0	Liste des ports

Liste des ports Pays : Chine : 5 Ports

indicatif	nom	Actions
CNSHG	Shangai	Types de navires compatibles
CNYTN	Yantian	Types de navires compatibles
CNNSA	Nansha	Types de navires compatibles
CNNGB	Ningbo	Types de navires compatibles
CNQDG	Qingdao	Types de navires compatibles



Pour cet affichage, on se sert de ce qui a déjà été fait.

En tant que responsable
Je voudrais connaître le nombre de ports par pays susceptible d'accueillir un type de navire donné
Afin d'étudier la faisabilité d'une liaison

En tant que responsable

Je voudrais connaître la distance en miles entre 2 ports
Afin de planifier les liaisons maritimes

Partie 6 : SYMFONY C'EST COOL



Un conseil
Voir la commande

make:crud

Vous aurez juste à charger une dépendance et c'est parti !!!

... elle devrait aussi vous rendre des services !!!



Bravo pour votre travail,