Prise en main de Symfony 5.4
Doctrine
Structure

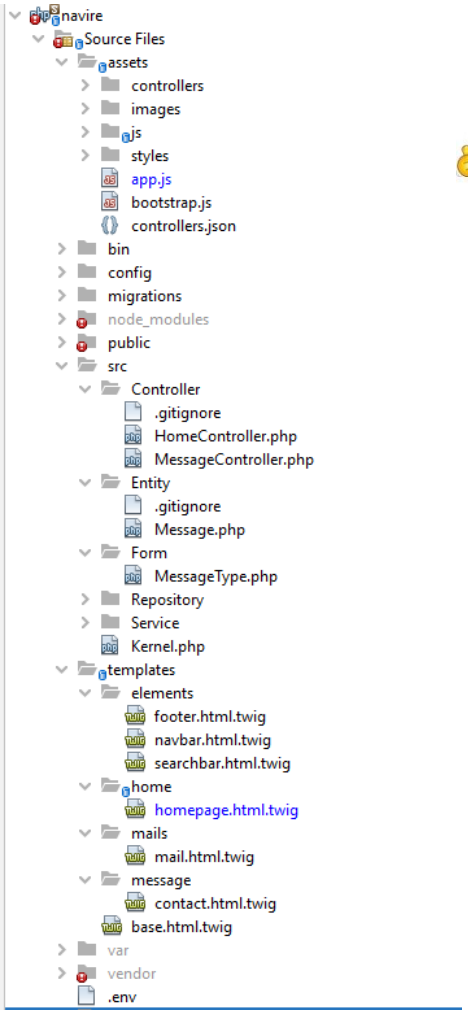
Objectifs :
Développer une petite application qui va gérer un les navires.
Versionner le projet
Valider les concepts vus en cours

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

Partie 1 : POINT ACTUEL SUR LE PROJET

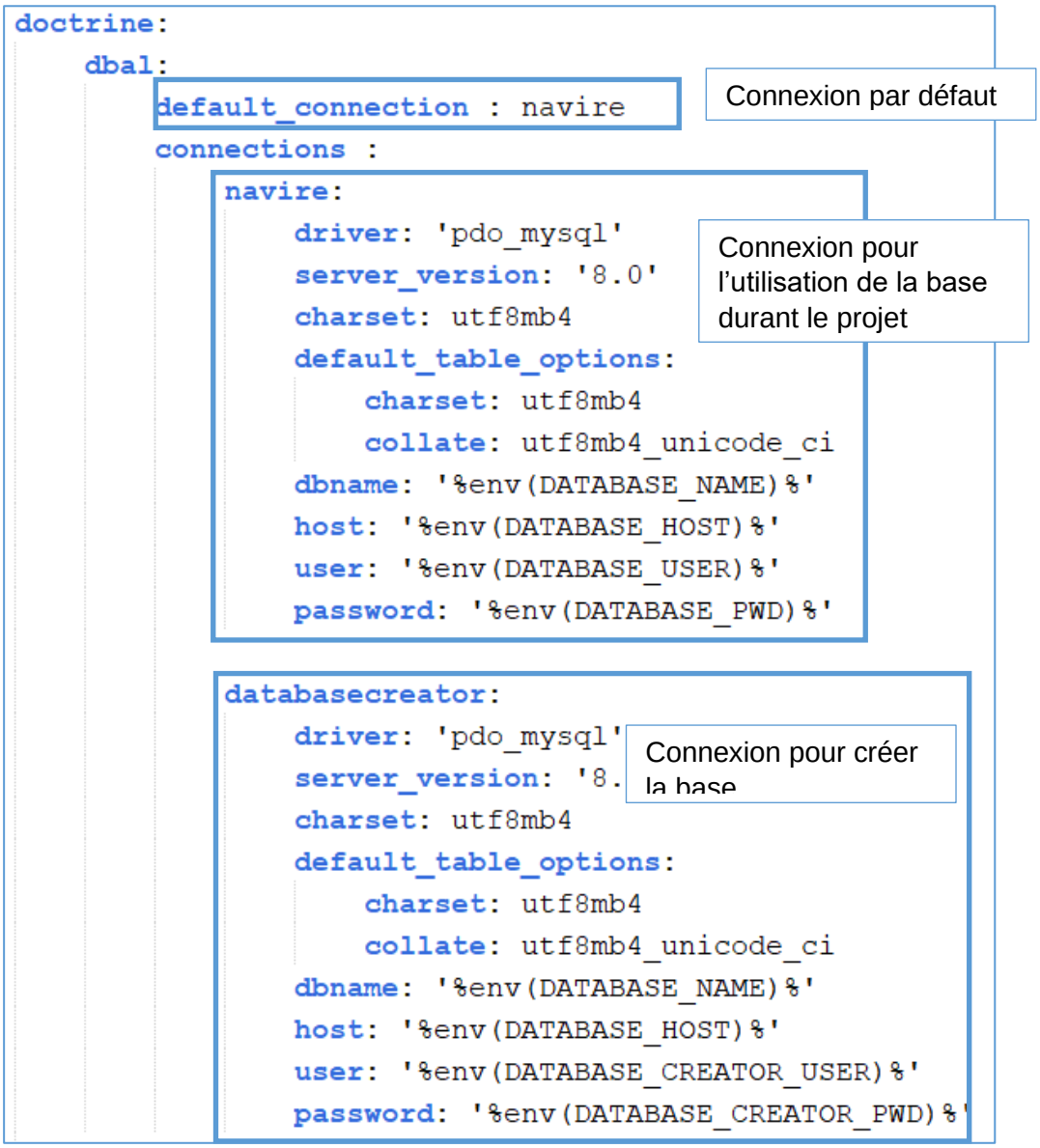
1. Structure du projet :



Mon projet est fait en utilisant le webpack-encore

2. Création de la base de données

Fichier config/packages/doctrine.yaml :



Fichier .env :

```
# Mysql projet Navire
DATABASE_URL=mysql://db_user:db_password@127.0.0.1:3306/db_name
### navire
DATABASE_USER=usernavire
DATABASE_PWD=usernavire
DATABASE_NAME=naviresymfo
DATABASE_HOST=localhost
### databasecreator
DATABASE_CREATOR_USER=databasecreator
DATABASE_CREATOR_PWD=databasecreator
### end of projet Navire
```

Partie BD

```
### symfony/mailer ###
# Sendinblue
MAILER_DSN=smtp://contact@              :861              @smtp-relay.sendinblue.com:587
# end ofSendinblue
```

Partie mail

Rappel des ordres de création des utilisateurs de la BD :
databasecreator (voir TP Présentation)

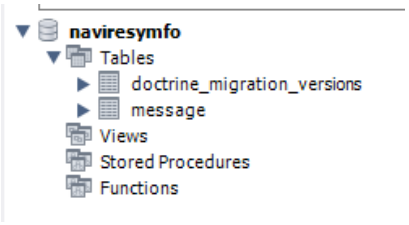
```
create user 'databasecreator'@'localhost' identified by 'databasecreator';
grant create , drop on *.* to 'databasecreator'@'localhost';
```

usernavire (voir TP Formulaire) :

```
create user 'usernavire'@'localhost' identified by 'usernavire';
grant all privileges on naviresymfo.* to 'usernavire'@'localhost';
```

La création de la BD :
symfony console doctrine:database:create --connection:databasecreator

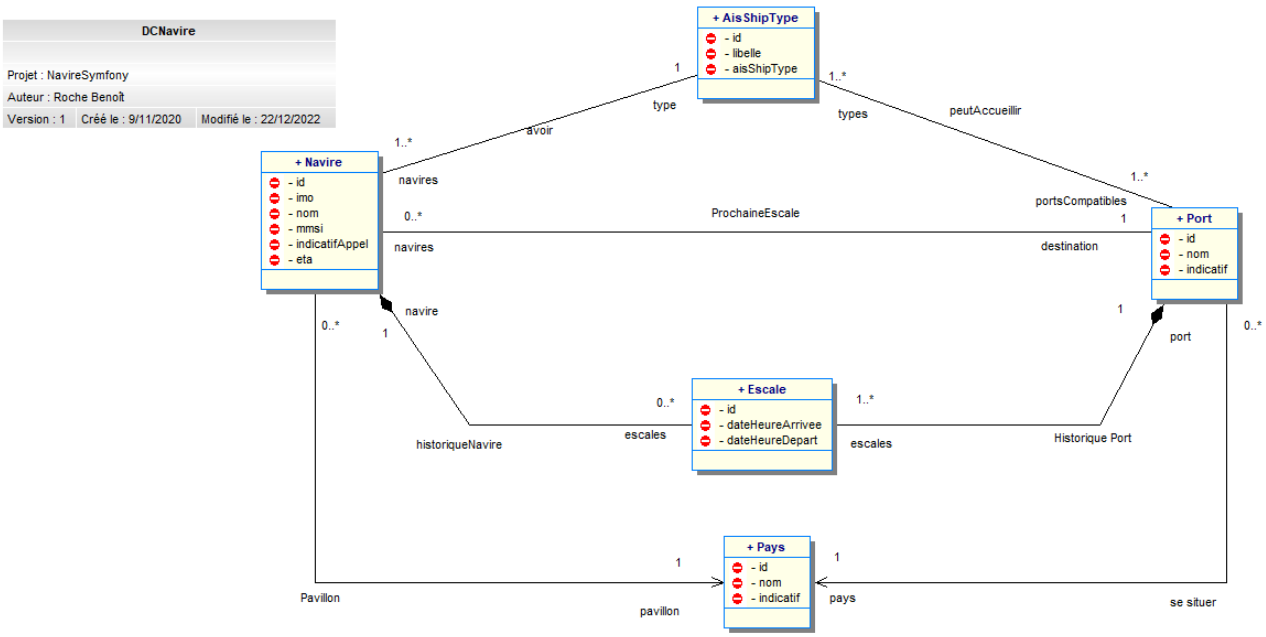
La base de données après le TP formulaire vue par l'utilisateur *usernavire* :



Partie 2 : LE PROJET

1. Diagramme de classes

Vous avez déjà réalisé le diagramme de classes qui a été validé par le product owner :

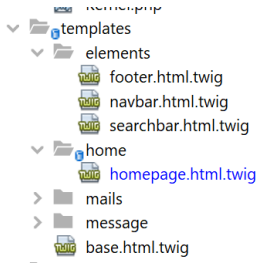


2. Contraintes

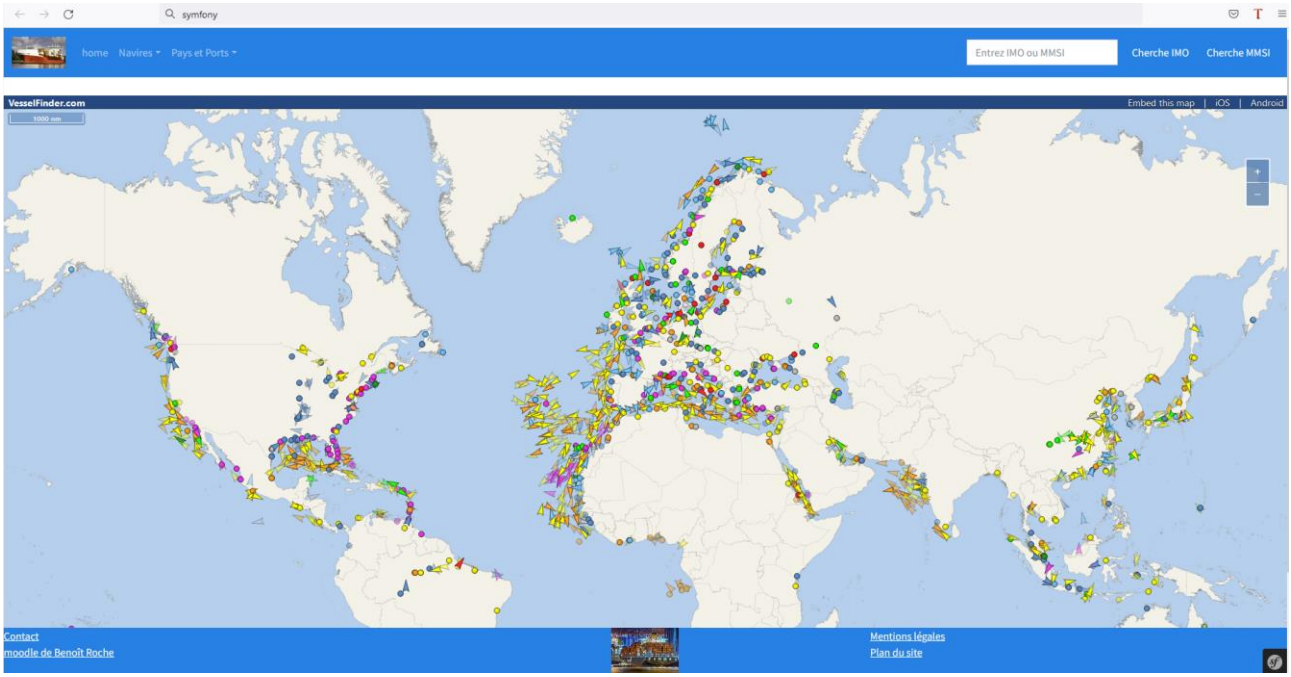
Vous fournirez systématiquement :

- ✓ Le nom des tables de la base de données concernée par l'entity
- ✓ Les noms des colonnes de la base de donnée concernées par chaque attribut
- ✓ Le nom des tables de jointures pour les relations ManyToMany
- ✓ Le nom des colonnes clés étrangères
- ✓ Les noms des tables et des colonnes de la base de données seront en ***minuscule***

3. La homepage

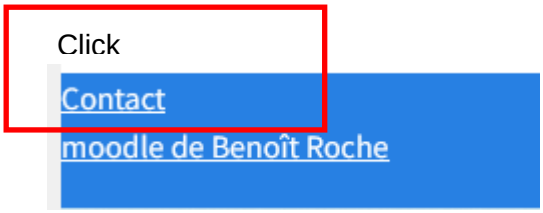


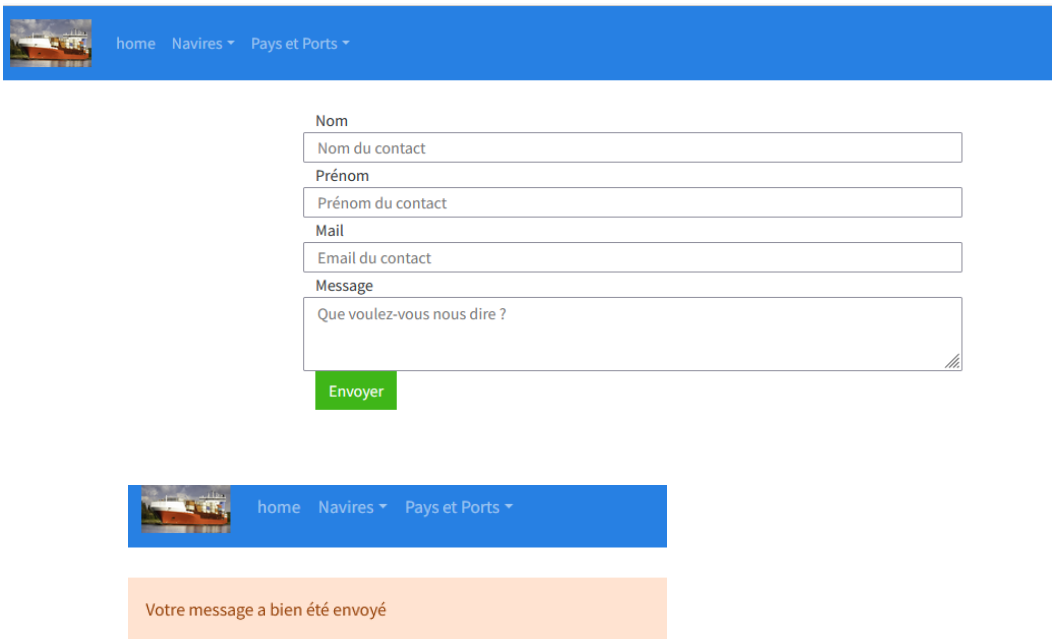

```php

namespace App\Controller;

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

class HomeController extends AbstractController
{
    #[Route('/', name: 'home')]
    public function home(): Response
    {
        return $this->render('home/homepage.html.twig', []);
    }
}

```



4. Le formulaire d'envoi de message

Click





Partie 3 : 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. Diagramme de classes concerné :

2. Entity AisShipType

Contraintes :
L'attibut *aisShipType* représente le premier niveau de classification des navires au niveau international. (voir codification internationale ici). Les valeurs permises vont de 1 à 9.

Vous gèrerez les règle d'intégrité au niveau de l'entity.

Pour la première voici les ordres de création :

```
PS T:\Wampsites\CoursSymfony\navire> symfony console make:entity

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

 created: src/Entity/AisShipType.php
 created: src/Repository/AisShipTypeRepository.php

 Entity generated! Now let's add some fields!
 You can always add more fields later manually or by re-running this command.

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

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

 Can this field be null in the database (nullable) (yes/no) [no]:
 >

 updated: src/Entity/AisShipType.php

 Add another property? Enter the property name (or press <return> to stop adding field
 > libelle

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

 Field length [255]:
 > 60

 Can this field be null in the database (nullable) (yes/no) [no]:
 >

 updated: src/Entity/AisShipType.php

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


 Success!


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

PS T:\Wampsites\CoursSymfony\navire>
```

Vous apporterez ensuite les compléments au code généré afin de satisfaire les contraintes :

```php
#[ORM\Table (name :'aisshiptype') ]
#[ORM\Entity(repositoryClass: AisShipTypeRepository::class)]
class AisShipType
{
    #[Assert\Unique(fields:['aisShipType'])]

    #[ORM\Id]
    #[ORM\GeneratedValue]
    #[ORM\Column (name:'id')]
    private ?int $id = null;

    #[ORM\Column (name:'aisshiptype')]
    #[Assert\Range(
        min: 1,
        max: 9,
        notInRangeMessage: 'Le type AIS doit être compris entre {{ min }} et {{ max }}',
    )]

    private ?int $aisShipType = null;

    #[ORM\Column(name:'libelle' , length: 60)]
    private ?string $libelle = null;
```

Puis vous générerez le fichier de migration :

```
ıy\navire> symfony console make:migration
```

Ordre SQL généré :

```php
// this up() migration is auto-generated, please modify it to your needs
$this->addSql('CREATE TABLE aisshiptype (id INT AUTO_INCREMENT NOT NULL, aisshiptype INT NOT NULL, '
        . 'libelle VARCHAR(60) NOT NULL, PRIMARY KEY(id)) '
        . 'DEFAULT CHARACTER SET utf8mb4 COLLATE `utf8mb4_unicode_ci` ENGINE = InnoDB');
```

Et vous effectuerez la migration :

```
PS T:\Wampsites\CoursSymfony\navire> symfony console make:migration

 Success!

 Next: Review the new migration "migrations/Version20221220110120.php"
 Then: Run the migration with php bin/console doctrine:migrations:migrate
 See https://symfony.com/doc/current/bundles/DoctrineMigrationsBundle/index.html
PS T:\Wampsites\CoursSymfony\navire> Symfony console doctrine:migrations:migrate

 WARNING! You are about to execute a migration in database "naviresymfo" that could result in schema changes and data loss. Are you sure you wish to continue? (yes/no) [yes]:
 >

[notice] Migrating up to DoctrineMigrations\Version20221220110120
[notice] finished in 945.9ms, used 20M memory, 1 migrations executed, 1 sql queries

PS T:\Wampsites\CoursSymfony\navire>
```

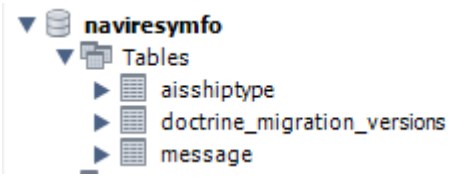
Une commande utile :

symfony console doctrine:migrations:status

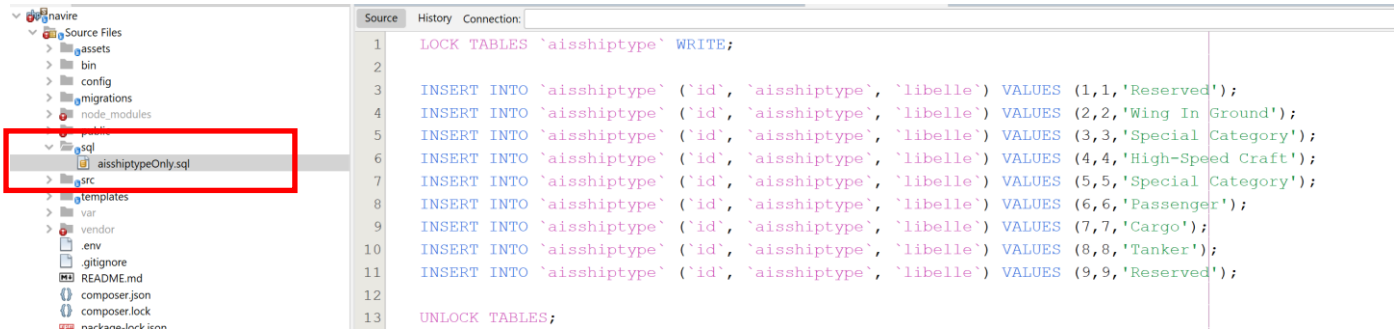
Elle vous permet de voir l'état des migrations :

```
PS T:\Wampsites\CoursSymfony\navire> symfony console doctrine:migrations:status
+----------------------+---------------------+-----------------------------------------------------------------------------+
| Configuration                                                                                                            |
+----------------------+---------------------+-----------------------------------------------------------------------------+
| Storage              | Type                | Doctrine\Migrations\Metadata\Storage\TableMetadataStorageConfiguration |
|                      | Table Name          | doctrine_migration_versions                                                 |
|                      | Column Name         | version                                                                     |
|----------------------+---------------------+-----------------------------------------------------------------------------|
| Database             | Driver              | Symfony\Bridge\Doctrine\Middleware\Debug\Driver                             |
|                      | Name                | naviresymfo                                                                 |
|----------------------+---------------------+-----------------------------------------------------------------------------|
| Versions             | Previous            | DoctrineMigrations\Version20221221145628                                    |
|                      | Current             | DoctrineMigrations\Version20230101080743                                    |
|                      | Next                | Already at latest version                                                   |
|                      | Latest              | DoctrineMigrations\Version20230101080743                                    |
|----------------------+---------------------+-----------------------------------------------------------------------------|
| Migrations           | Executed            | 19                                                                          |
|                      | Executed Unavailable | 0                                                                          |
|                      | Available           | 19                                                                          |
|                      | New                 | 0                                                                           |
|----------------------+---------------------+-----------------------------------------------------------------------------|
| Migration Namespaces | DoctrineMigrations  | T:\Wampsites\CoursSymfony\navire/migrations                                 |
+----------------------+---------------------+-----------------------------------------------------------------------------+
PS T:\Wampsites\CoursSymfony\navire>
```

La table aisshiptype a été créée :



Vous pourrez la remplir avec le script fourni aisshiptypeOnly.sql que vous placerez dans le dossier sql à la racine du projet :



Vérification :

```
1 •    SELECT * FROM naviresymfo.aisshiptype;
```

Result Grid | Filter Rows: | Edit:

id	aisshiptype	libelle
1	1	Reserved
2	2	Wing In Ground
3	3	Special Category
4	4	High-Speed Craft
5	5	Special Category
6	6	Passenger
7	7	Cargo
8	8	Tanker
9	9	Reserved

3. Entity Navire

Vous créerez de la même manière l'entity Navire.

Il faudra prendre en compte les contraintes suivantes :

- ✓ Numéro IMO : Chaine de caractères 7 chiffres le premier ne pouvant pas être un 0. Unique
- ✓ Nom du navire : 3 caractères alphanumériques au minimum
- ✓ Numéro MMSI : Chaine de caractères 9 chiffres le premier ne pouvant pas être un 0. Unique.
- ✓ Indicatif d'appel : chaine de 10 caractères
- ✓ Eta : Estimated Time of arrival : type DateTime, peut être null

Extraits de la classe Navire :

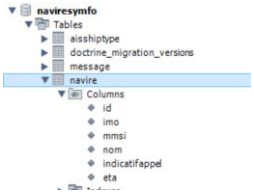
```php
class Navire
{
    #[Assert\Unique(fields:['imo','mmsi','indicatifAppel'])]

    #[ORM\Id]
    #[ORM\GeneratedValue]
    #[ORM\Column (name:'id')]
    private ?int $id = null;
```

```php
#[ORM\Column(name:'imo', length: 7)]
#[Assert\Regex('[1-9][0-9]{6}', message : 'le numéro IMO doit être unique et composé de 7 chiffres sans commencer par 0')]
private ?string $imo = null;
```

Créez la migration et observez le code généré dans le fichier de migration

```
$this->addSql('CREATE TABLE navire (id INT AUTO_INCREMENT NOT NULL, imo VARCHAR(7) NOT NULL, mmsi VARCHAR(9) NOT NULL, '
        . 'nom VARCHAR(100) NOT NULL, indicatifappel VARCHAR(10) NOT NULL, eta DATETIME NOT NULL, '
        . 'PRIMARY KEY(id)) DEFAULT CHARACTER SET utf8mb4 COLLATE `utf8mb4_unicode_ci` ENGINE = InnoDB');
```

Effectuez la migration.

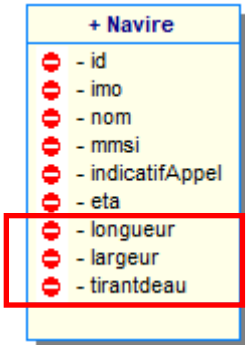


4. Entité navire corrections

Après le premier sprint où vous avez présenté la maquette de l'application au PO, celui-ci vous a demandé d'apporter des modifications à la classe Navire

Vous aurez à rajouter les attributs suivants:

- ✓ longueur entier positif
- ✓ largeur : entier positif
- ✓ tirantdeau : float !!!



Par ailleurs, il vous est demandé d'ajouter la relation bidirectionnelle entre navire et TypeNavire.
Il s'agit d'une relation ***ManyToOne***
Pas de suppression en cascade (orphanRemoval : no)

```
(orphanRemoval)? (yes/no) [no]
```

Vous modifierez l'entity Navire pour prendre en compte cette évolution du besoin
Vous n'oublierez pas l'attribut #[ORM\Column **(name:**'id')] pour forcer le nom de la colonne dans la BD

```
PS T:\Wampsites\CoursSymfony\navire> symfony console make:entity

 Class name of the entity to create or update (e.g. AgreeableElephant):
 > Navire
```

Extraits :
Classe Navire :

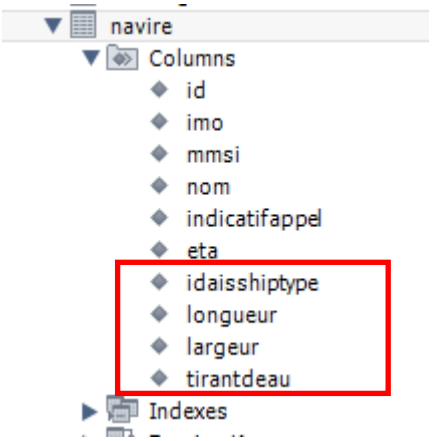
```
#[ORM\ManyToOne(inversedBy: 'navires')]
#[ORM\JoinColumn(name:'idaisshiptype', referencedColumnName:'id', nullable: false)]
private ?AisShipType $aisShipType = null;
```

Classe AisShipType :

```
#[ORM\OneToMany(mappedBy: 'aisShipType', targetEntity: Navire::class)]
private Collection $navires;
```

Créez la migration et observez le code généré dans le fichier de migration

Effectuez la migration.
Vérifiez dans la base :

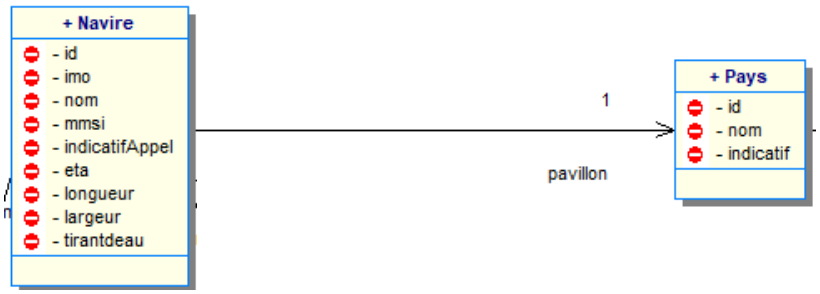


Bilan :

- ✓ *Vous venez de créer votre première relation bidirectionnelle ManyToOne / OneToMany*
- ✓ *vous avez vu les répercussions au niveau de la base de données avec la création d'une clé étrangère*
- ✓ *vous avez appris qu'il faut renommer tous les attributs et les objets*

Partie 4 : USER STORY 2

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



1. Diagramme de classes

Navire :

- ✓ Un Navire est caractérisé par un nom qui peut changer,
- ✓ un numéro IMO unique pour chaque navire,
- ✓ un identifiant MMSI unique pour chaque navire, il ne peut changer au cours de la vie du navire,
- ✓ un type de navire.
- ✓ un pavillon. Le pavillon correspond au ***pays*** dans lequel le navire est actuellement enregistré.

Pays :

- ✓ Nom : chaine de 70 caractères
- ✓ Indicatif : 3 lettres majuscules

Vous aurez ici à :

- ✓ Créer l'entity Pays,
- ✓ Apporter les modifications au code généré de la classe Pays pour satisfaire les contraintes imposées
- ✓ Rajouter la relation entre Navire et Pays. Relation unidirectionnelle ManyToOne dont l'entity Navire est prioritaire
- ✓ L'indicatif Pays doit être unique

2. Modifications apportées

Entity Pays :

```
#[ORM\Table (name :'pays') ]
#[Assert\Unique(fiefds: ['indicatif'])]
#[ORM\Entity(repositoryClass: PaysRepository::class)]
class Pays
{
```

```
#[ORM\Column(name:'indicatif' , length: 3)]
#[ORM\Regex(pattern:"/[A-Z]{3}/", message:"L'indicatif Pays a strictement 3 caractères")]
private ?string $indicatif = null;
```

Entity Navire :

```
#[ORM\ManyToOne]
#[ORM\JoinColumn(name: 'idpays', referencedColumnName: 'id', nullable: false)]
private ?Pays $pavillon = null;
```

3. Les index

Si on observe la structure de la table *navire*, on s'perçoit que des index ont été posés sur les colonnes clés primaire et étrangères mais pas sur les colonnes imo, mmsi.

Vous allez donc poser les index suivants :

- ✓ Table Navire : colonnes imo, mmsi

✓ Table Pays : colonne indicatif

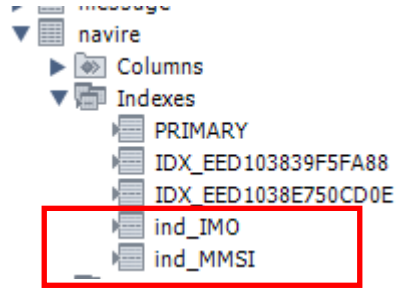
```
#[Assert\Unique(fiefds: ['imo', 'mmsi', 'indicatifAppel'])]
#[ORM\Entity(repositoryClass: NavireRepository::class)]
#[ORM\Index(name:'ind_IMO', columns: ['imo'])]
#[ORM\Index(name:'ind_MMSI', columns: ['mmsi'])]
class Navire {
```

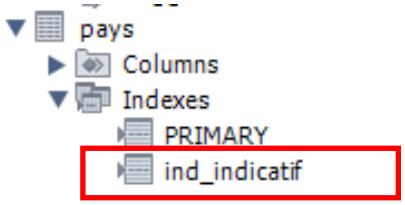
```
#[ORM\Table (name :'pays') ]
#[Assert\Unique(fiefds: ['indicatif'])]
#[ORM\Entity(repositoryClass: PaysRepository::class)]
#[ORM\Index(name:'ind_indicatif', columns: ['indicatif'])]
class Pays
```

Migration générée :

```
// this up() migration is auto-generated, please modify it to your needs
$this->addSql('CREATE INDEX ind_IMO ON navire (imo)');
$this->addSql('CREATE INDEX ind_MMSI ON navire (mmsi)');
$this->addSql('CREATE INDEX ind_indicatif ON pays (indicatif)');
```

Après l'exécution des migrations :





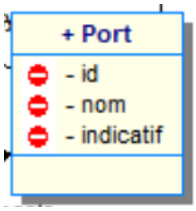
Vous pouvez dès maintenant exécuter le script sql/paysOnly.sql pour charger la table des pays :

id	nom	indicatif
1	Aruba (Ile d')	ABW
2	Angola	AGO
3	Anguilla (Ile d')	AIA
4	Albanie	ALB
5	Aland (Iles d')	ALD
6	Antilles Néerlandaises	ANT
7	Emirats Arabes Unis	ARE

Partie 5 : USER STORY 3 : LES PORTS

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

1. Diagramme de classes



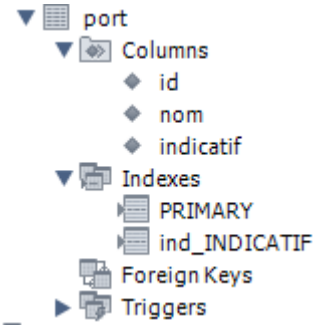
Les contraintes :

- ✓ Nom : chaine de caractères de 70 caractères maximum. Not null
- ✓ Indicatf : chaine de 5 caractères de A à Z. Les valeurs sont uniques et on veut un index appelé ind_indicatif sur la colonne. Not null

2. Entity créée

```php
#[ORM\Entity(repositoryClass: PortRepository::class)]
#[Assert\Unique(fiefds: ['indicatif'])]
#[ORM\Index(name:'ind_INDICATIF', columns: ['indicatif'])]
class Port
{
    #[ORM\Id]
    #[ORM\GeneratedValue]
    #[ORM\Column (name:'id')]
    private ?int $id = null;

    #[ORM\Column(name: 'nom',length: 70)]
    private ?string $nom = null;

    #[ORM\Column(name:'indicatif' , length: 5)]
    #[ORM\Regex(pattern:"/[A-Z]{5}/", message:"L'indicatif Port a strictement 5 caractères")]
    private ?string $indicatif = null;
```

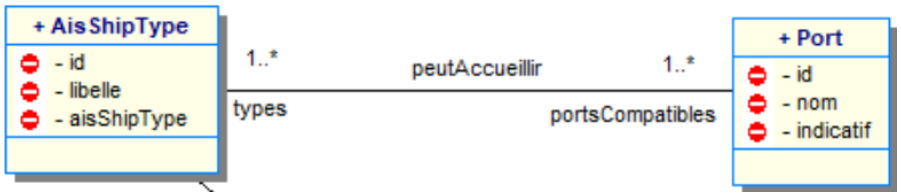
Après migration :



Partie 6 : 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. Diagramme de classes



2. La relation

Il s'agit pour vous de créer une relation bidirectionnelle entre les 2 entities

- ✓ Attention aux cardinalités 1 minimum de chaque côté de l'association
- ✓ L'entity Port est propriétaire de la relation, c'est donc elle que vous modifierez pour créer la relation.
- ✓ Nom de la table de jointure qui sera créée : porttypecompatible
- ✓ Nom des colonnes de la table de jointure : idport, idaisshiptype

3. Code des relations :

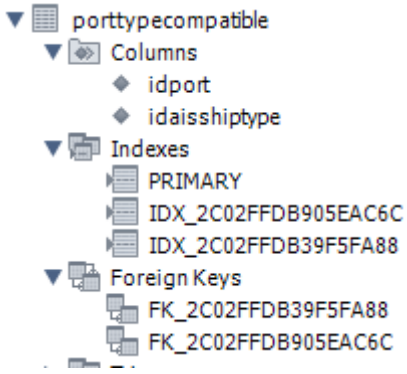
Entity Port :

```
#[ORM\ManyToMany(targetEntity: AisShipType::class, inversedBy: 'portsCompatibles')]
#[ORM\JoinTable(name:'porttypecompatible')]
#[ORM\JoinColumn(name:'idport', referencedColumnName:'id')]
#[ORM\InverseJoinColumn(name:'idaisshiptype', referencedColumnName:'id')]
private Collection $types;
```

Entity AisShipType :

```
#[ORM\ManyToMany(targetEntity: Port::class, mappedBy: 'types')]
private Collection $portsCompatibles;
```

Et dans la base de données :



Partie 7 : USER STORY 5

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

Chaque port est situé dans un pays qui est caractérisé par un nom et un indicatif international.

1. Diagramme de classes :

2. La relation

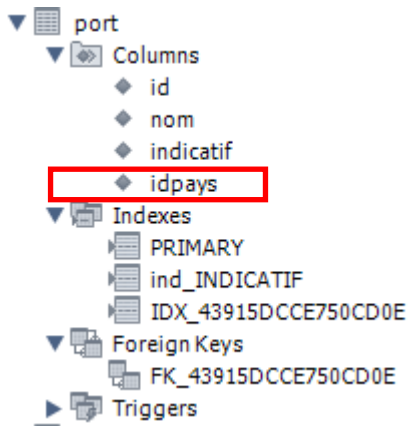
Ici il s'agit de créer une relation unidirectionnelle dont :

- ✓ L'entity Port est propriétaire de la relation, c'est donc elle que vous modifierez pour créer la relation.
- ✓ Le nom de la colonne de jointure dans la table port sera idpays

3. Code des relations

```
#[ORM\ManyToOne]
#[ORM\JoinColumn(name:'idpays', nullable: false)]
private ?Pays $pays = null;
```

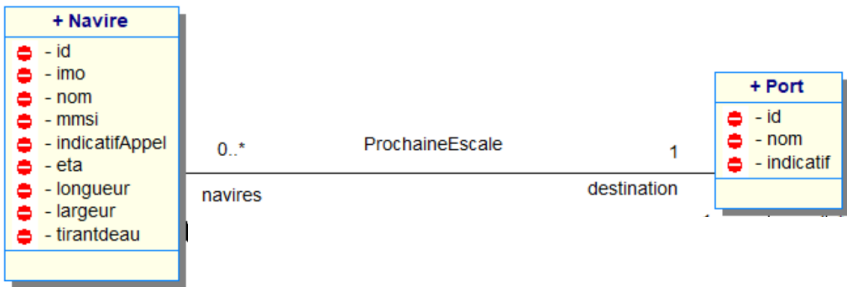
Et dans la base de données :



Partie 8 : USER STORY 6

En tant que gestionnaire Je voudrais connaitre la prochaine escale d'un navire Afin d'informer le port de destination de sa prochaine arrivée.

1. Diagramme de classes



2. La relation

Ici il s'agit de créer une relation bidirectionnelle dont :

- ✓ L'entity Navire est propriétaire de la relation, c'est donc elle que vous modifierez pour créer la relation.
- ✓ L'entity navire aura donc un nouvel attribut *destination* de type Port
- ✓ L'entity Port aura donc un nouvel attribut *navires* de type Collection de la classe Navire
- ✓ Le nom de la colonne de jointure dans la table navire sera idport

Il pourra arriver qu'un navire ait pour prochaine destination un Port qui n'est pas encore enregistré dans la base de données. Dans ce cas, le port de destination du Navire devra être créé en même temps que l'ajout ou la mise à jour d'un navire.

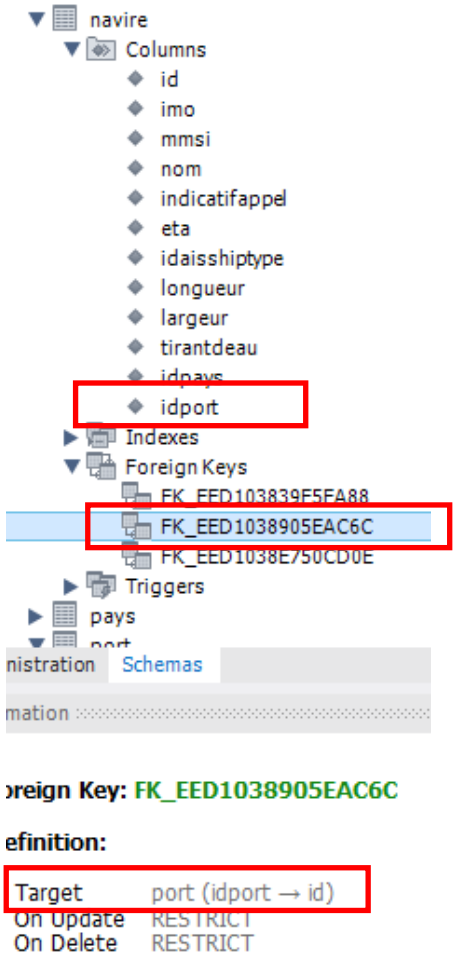
Entity Navire :

```
#[ORM\ManyToOne(inversedBy: 'navires' , cascade:['persist'])]
#[ORM\JoinColumn(name: 'idport', referencedColumnName: 'id', nullable: true)]
private ?Port $destination = null;
```

Entity Port :

```
#[ORM\OneToMany(mappedBy: 'destination', targetEntity: Navire::class)]
private Collection $navires;
```

```
// this up() migration is auto-generated, please modify it to your needs
$this->addSql('ALTER TABLE navire ADD idport INT NULL');
$this->addSql('ALTER TABLE navire ADD CONSTRAINT FK_EED1038905EAC6C FOREIGN KEY (idport) REFERENCES port (id)');
$this->addSql('CREATE INDEX IDX_EED1038905EAC6C ON navire (idport)');
```



Et dans la base de données :

La colonne idport a été créée
elle accepte Null (on peut le voir dans l'ordre de migration généré) .
Et c'est bien une clé étrangère

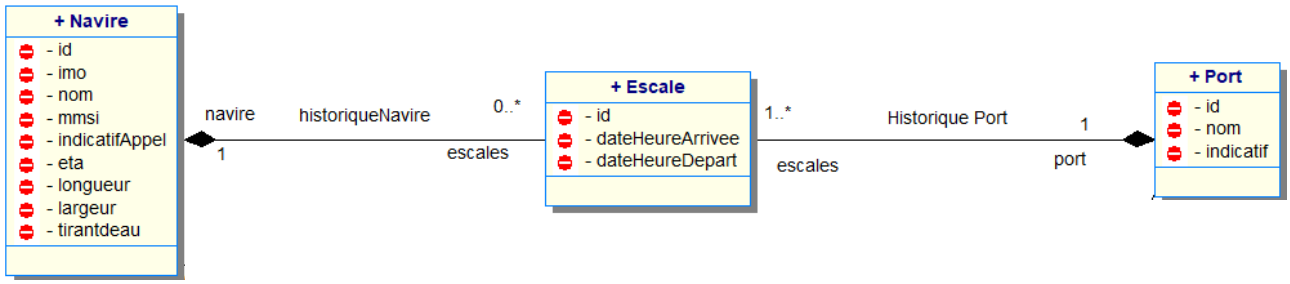
Partie 9 : USER STORY 7

En tant que gestionnaire Je voudrais connaitre la prochaine escale d'un navire Afin d'informer le port de destination de sa prochaine arrivée.

Pour une escale, on désire connaitre :

- ✓ Le navire concerné,
- ✓ Le port concerné,
- ✓ La date et l'heure d'arrivée
- ✓ La date et l'heure de départ

1. Diagramme de classes :



2. Modifications à faire

Il s'agit ici :

- ✓ De créer l'entity Escale
  - ➢ L'attribut dateHeureArrive est de tite DateTime Not null
  - ➢ L'attribut dateHeureDepart est de type DateTime ***Nullable***

- ✓ De créer la relation historiqueNavire dont l'entity Escale sera propriétaire.
  - ➢ Il s'agit d'une relation bidirectionnelle.
  - ➢ La colonne de jointure dans la table escale se nommera idnavire

- ✓ De créer la relation historiquePort dont l'entity Escale sera propriétaire.
  - ➢ Il s'agit d'une relation bidirectionnelle.
  - ➢ La colonne de jointure dans la table escale se nommera idport

Vous remarquerez l'ajout d'une composition sur les Classes Navire et Port. Cela signifie que la classe Escale est n'existe que par rapport aux deux classes reliées. Pas d'escale sans Navire et pas d'escale sans Port

Vous répondrez par yes à ces deux questions lors de la création des relations :

```
Do you want to activate orphanRemoval on your relationship?
A Escale is "orphaned" when it is removed from its related Navire.
e.g. $navire->removeEscale($escale)

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

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

Relation historiqueNavire

```
Do you want to activate orphanRemoval on your relationship?
A Escale is "orphaned" when it is removed from its related Port.
e.g. $port->removeEscale($escale)

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

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

Relation historiqueEscales

L'entity Escale :

```php
class Escale
{
    #[ORM\Id]
    #[ORM\GeneratedValue]
    #[ORM\Column]
    private ?int $id = null;

    #[ORM\Column(name:'dateHeureArrivee' , type: Types::DATETIME_MUTABLE)]
    private ?\DateTimeInterface $dateHeureArrivee = null;

    #[ORM\Column(name:'dateHeureDepart' , type: Types::DATETIME_MUTABLE, nullable: true)]
    private ?\DateTimeInterface $dateHeureDepart = null;

    #[ORM\ManyToOne(inversedBy: 'escales')]
    #[ORM\JoinColumn(name:'idnavire' , referencedColumnName: 'id', nullable: false)]
    private ?Navire $navire = null;

    #[ORM\ManyToOne(inversedBy: 'escales')]
    #[ORM\JoinColumn(name:'idport', referencedColumnName: 'id', nullable: false)]
    private ?Port $port = null;
```

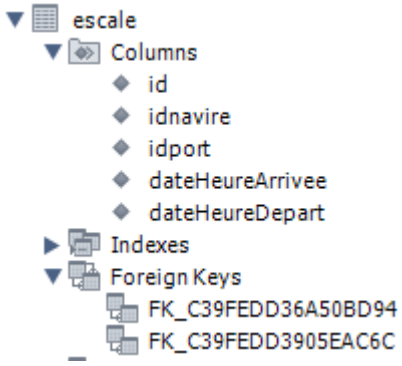
L'entity Navire :

```php
    #[ORM\OneToMany(mappedBy: 'navire', targetEntity: Escale::class, orphanRemoval: true)]
    private Collection $escales;
```

L'entity Port :

```php
    #[ORM\OneToMany(mappedBy: 'port', targetEntity: Escale::class, orphanRemoval: true)]
    private Collection $escales;
```

3. La base de données :



Partie 10 : CHARGEMENT DE LA BASE DE DONNEES

1. chargement

Si vous avez bien travaillé, bien respecté les noms et formats de colonnes ainsi que les relations, vous devriez pouvoir exécuter le script fourni par votre professeur pour charger les données dans la base de données.

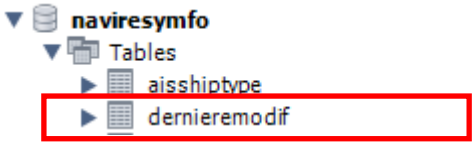
Le script est à effectuer sous une connexion usernavire

Nom du script : *chargenavire2022.sql*

2. Modifications dans la base

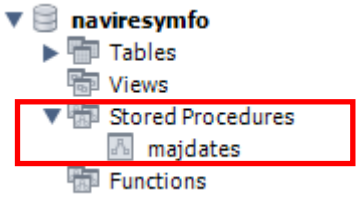
Une fois votre script entièrement bien exécuté, vous remarquerez :

- ✓ Une nouvelle table : dernièremodif



Cette table va servir à conserver la date de la dernière mise à jour de données effectuée. Elle servira à calculer le décalage de dates à effectuer

- ✓ La procédure stockée majdates



Cette procédure permet d'effectuer une translation de dates depuis la dernière mise à jour afin d'avoir toujours :

- ✓ Des bateaux au port
- ✓ Des bateaux en mer avec leur ETA

3. Mise à jour des données de navigation

Une fois que le script est correctement exécuté, il n'y a plus qu'à exécuter la procédure stockée :

```
call naviresymfo.majdates();
```

Vous pouvez exécuter ce script chaque jour

En dernier recours, vous pouvez aussi exécuter le script complet à tout moment !!!

Symfony étant toujours inachevé, Il ne vous reste plus maintenant qu'à mettre tout ceci en musique !
… Dans le prochain TD

Bravo pour votre travail,