Mapping DTO et API côté Symfony

Pour mapper une entité sur Symfony il faut tout d'abord installer automapper et api

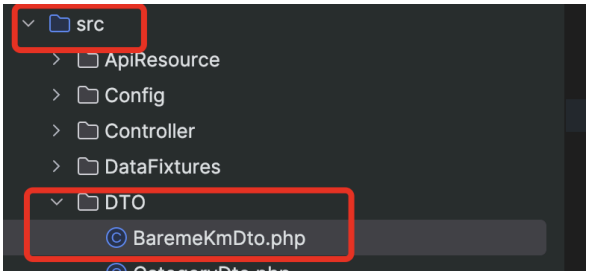
On installera d'abord api
composer require api

```
lea@mac notes-de-frais % composer require api
./composer.json has been updated
Running composer update api-platform/api-pack
Loading composer repositories with package information
Updating dependencies
Lock file operations: 3 installs, 0 updates, 0 removals
  - Locking api-platform/api-pack (v1.4.0)
  - Locking symfony/orm-pack (v2.7.0)
  - Locking symfony/serializer-pack (v1.4.0)
```

On utilisera dev-main pour une compatibilité avec la dernière version d'api ainsi que symfony 8
composer require jolicode/automapper dev-main

```
lea@mac notes-de-frais % composer require jolicode/automapper dev-main
./composer.json has been updated
Running composer update jolicode/automapper
Loading composer repositories with package information
Updating dependencies
Lock file operations: 0 installs, 1 update, 0 removals
  - Upgrading jolicode/automapper (dev-main 98069b1 => dev-main dd0a23d)
Writing lock file
```

Ensuite on prépare l'entité à mapper en DTO :
Pour s'y retrouver plus facilement, créer un fichier "DTO" dans le dossier src puis créer le fichier de l'entité souhaité, par exemple : "BaremeKmDTO.php" (le nom n'est pas obligé d'être identique à l'entité qu'on souhaite mapper)



Pour mapper l'entité, on récupère les champs qu'on souhaite retrouver dans l'application mobile avec le nom de la classe et "public" suivi du type de champ ainsi que le nom exact dans l'entité :

```php
namespace App\DTO;

class BaremeKmDto {
   public ?int $id;
   public ?string $bareme;
   public ?int $puissance_fiscale ;
   public ?int $annee;
   public ?\DateTimeImmutable $createdAt;
   public ?\DateTimeImmutable $updatedAt;
   public ?int $km_min;
   public ?int $km_max;
}
```

Dans l'entité :
On vérifie l'exactitude des champs

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

#[ORM\Column(type: 'decimal', precision: 6, scale: 3)]
private ?string $bareme = null;


#[ORM\Column]
private ?int $puissance_fiscale = null;

#[ORM\Column]
private ?int $annee = null;

#[ORM\Column]
private ?\DateTimeImmutable $createdAt = null;

#[ORM\Column(nullable: true)]
private ?\DateTimeImmutable $updatedAt = null;
```

```php
#[ORM\Column]
private ?int $km_min = null;

#[ORM\Column(nullable: true)]
private ?int $km_max = null;
```

Au début de l'entité, on prépare chaque route/endpoint nécessaire à l'utilisation du DTO (ici, pour un équivalent de crud sans suppression, on aura besoin de l'index, du show, du new et de l'edit) . Chaque entrée du tableau contient l'url utilisé, le controller, le nom de la route.

```php
#[ApiResource(
    operations: [
        new Get(
            uriTemplate: '/bareme',
            controller: ReferentielKmDtoController::class,
            name: 'api_liste_baremes',
        ),
        new Get(
            uriTemplate: '/bareme/{id}',
            controller: ReferentielKmDtoController::class,
            name: 'api_bareme_show',
            read: false
        ),
        new Post(
            uriTemplate: '/bareme/new',
            controller: ReferentielKmNewDtoController::class,
            name: 'api_bareme_new',
            read: false
        ),
        new Put(
            uriTemplate: '/bareme/edit/{id}',
            controller: ReferentielKmEditDtoController::class,
            name: 'api_bareme_edit',
            read: false
        ),

    ]
)]
```

On créera un Controller DTO spécifique pour chaque endpoint ajouté ici (l'index et le show étant rassemblés). Le chemin étant : src/Controller/Api