Invoke response : Index, show, new et edit pour mobile

Construction d'un controller :
Un controller DTO qui a été configuré avec API ressource dans une entité se présente de cette manière :

```
namespace App\Controller\Api;

use App\DTO\BaremeKmDto;
use App\Entity\ReferentielKm;
use App\Entity\User;
use App\Repository\ReferentielKmRepository;
use AutoMapper\AutoMapperInterface;
use Doctrine\ORM\EntityManagerInterface;
use
Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\JsonResponse;
use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\HttpKernel\Attribute\AsController;

#[AsController]
class ReferentielKmDtoController extends AbstractController
{

   public function __construct(
       protected AutoMapperInterface    $mapper,
       protected EntityManagerInterface $entityManager,
   )
   {

   }

public function __invoke(): JsonResponse
{
```

La mention #AsController enregistre automatiquement la classe comme un service controller et permet à API Platform de l'utiliser comme endpoint personnalisé

Index

Un index d'un crud permet de récupérer toutes les informations d'une entité, c'est ce que va faire le controller DTO :

```
public function __invoke(): JsonResponse
{

   $bareme = $this->entityManager
       ->getRepository(ReferentielKm::class)
       ->findBy(
           [],
           ['annee' => 'DESC']
       );

$dto = $this->mapper->mapCollection($bareme,
BaremeKmDto::class);

return $this->json($dto);

}
```

On récupère tous les barèmes en indiquant qu'on les trie en fonction de l'année et
$dto = $this->mapper->mapCollection($bareme, BaremeKmDto::class);
sert à transformer une collection d'objets en collection de DTO en fonction du fichier construit précédemment (BaremeKmDto).

Show

On utilisera le même endpoint que l'index puisque l'application mobile filtrera ou sélectionnera celui dont elle a besoin pour le show.

New

Le new se construira comme la création d'un formulaire Symfony avec un new Entité() ainsi que des set pour chaque champ, ainsi qu'un persist et un flush de données. Cependant à la différence l'index, on ne fera pas un mapcollection mais un map uniquement car il n'y a qu'une seule création à la fois

```
ppublic function __invoke(Request $request): JsonResponse
{
   $data = json_decode($request->getContent(), true);

   $bareme = new ReferentielKm();
   $bareme->setBareme($data['bareme']);
   $bareme->setPuissanceFiscale($data['puissance_fiscale']);
```

```
    $bareme->setAnnee($data['annee']);
    $bareme->setKmMin($data['km_min']);
    $bareme->setKmMax($data['km_max'] ?? null);

    $this->entityManager->persist($bareme);
    $this->entityManager->flush();

    $dto = $this->mapper->map($bareme, BaremeKmDto::class);

    return new JsonResponse($dto, 201);
}
```

Edit

L'edit fonctionne comme un formulaire Symfony. On récupère le barème à modifier suivant son id, on récupère les nouvelles données en les comparant aux anciennes puis on flush. Comme pour le new on map les nouvelles données à la fin

```
public function __invoke(Request $request, int $id):
JsonResponse
{
    $bareme = $this->entityManager
        ->getRepository(ReferentielKm::class)
        ->find($id);

    if (!$bareme) {
        return new JsonResponse(['message' => 'Barème non
trouvé'], 404);
    }

    $data = json_decode($request->getContent(), true);

    $bareme->setBareme($data['bareme'] ??
$bareme->getBareme());
    $bareme->setPuissanceFiscale($data['puissance_fiscale'] ??
$bareme->getPuissanceFiscale());
    $bareme->setAnnee($data['annee'] ?? $bareme->getAnnee());
    $bareme->setKmMin($data['km_min'] ?? $bareme->getKmMin());
    $bareme->setKmMax($data['km_max'] ?? $bareme->getKmMax());
```

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

    $dto = $this->mapper->map($bareme, BaremeKmDto::class);

    return new JsonResponse($dto);
}
```

Symfony a préparé toutes ses données à envoyer.