part2-0.md

Part 2: Multilingual blog

Sylius is fully internationalized at the core. But creating translatable entities is not trivial. In this exercise we will add translations to our blog post.

Create translation entity.

Create a new entity which will contain the translations for the existing `Post` entity and update the post entity to use it.

1. Create the translation entity (e.g. `PostTranslation`) extending `AbstractTranslation` and implementing `ResourceInterface`. It should have `$id`, `$title` and `$body` fields.
2. Modify the Post enity to make use of the translation entity.
    - It must use the `TranslatableTrait` and implement the `TranslatableInterface`.
    - It should get and set values using the traits `translate()` method.
    - It should initialize `$this->translations` (imported with the trait) with a new `ArrayCollection`. > Hints:

```
$this->translate()->getTitle();
```

```
$this->translate()->setTitle($title);
```

- `Sylius\Component\Resource\Model\TranslatableTrait`
- `Doctrine\Common\Collections\ArrayCollection`
- `Sylius\Component\Resource\Model\TranslatableInterface`
- `@ORM\Id()`
- `@ORM\GeneratedValue()`
- `@ORM\Column(type="string")`

Form types

In order to allow the user to translate fields we need to create a new form type for the `Post` entity (it will no longer be dynamically created) and create a form type for the `PostTranslation`.

1. Create the `PostTranslationType` class. It should extend `AbstractResourceType` and override the constructor to pass the full class name of your post translation class to the parent. It should add `title` and `body` fields.
2. Create a `PostType` class, also extending `AbstractResourceType` and passing the class name to the parent in the constructor. It should add the `publishedAt` field and a `translations` field.
3. Configure the translation class in the Post's resource configuration.
4. Configure the post's admin routing to use the new Post form type.
5. Using the admin interface, add locales to the shop.
6. Enable locales for the shop channel.
7. Translate your blog post!
8. View it on the frontend and change the locale!

Hints:

```
<?php

namespace AppBundle\Form;

use Symfony\Component\Form\AbstractType;
use Symfony\Component\Form\FormBuilderInterface;
```

```
use Symfony\Component\Form\FormBuilderInterface;
use Sylius\Bundle\ResourceBundle\Form\Type\AbstractResourceType;
use AppBundle\Entity\PostTranslation;

class PostTranslationType extends ...
{
    public function __construct()
    {
        parent::__construct(PostTranslation::class);
    }

    public function buildForm(FormBuilderInterface $builder, array $options)
    {
        // $builder->add(<field name>, <field type>);
    }
}
```

```
<?php

namespace AppBundle\Form;

use Symfony\Component\Form\AbstractType;
use Symfony\Component\Form\FormBuilderInterface;
use AppBundle\Entity\Post;
use Sylius\Bundle\ResourceBundle\Form\Type\AbstractResourceType;

class PostType extends ...
{
    public function __construct()
    {
        parent::__construct(Post::class);
    }

    public function buildForm(FormBuilderInterface $builder, array $options)
    {
        // $builder->add(<field name>, <field type>, <field options>);
    }
}
```

- Translation form type `sylius_translations` with option `type` and value `PostTranslationType::class`

```
# resources.yml
sylius_resource:
    resources:
        app.post:
            # ...
            translation:
                classes:
                    model: AppBundle\Entity\PostTranslation
```

```
# resource routing
    grid: app_post
    except: [ "show" ]
    templates: SyliusAdminBundle:Crud
    form: AppBundle\Form\PostType     # add this line
```

```
Configuration > Locales
```

```
Configuration > Channels
```