Formularze

v3.1

Plan

- Podstawy używania formularzy
- Zaawansowane formularze

Podstawy używania formularzy

Formularze

- Jedną z głównych części frameworku Symfony jest komponent **`Form`** służący do budowy i obsługi formularzy.
- Tak jak inne komponenty jest niezależny i może być wykorzystywany poza Symfony.

Komponent **`Form`** ma wbudowane następujące metody:

- obsługę requestów HTTP wraz z uploadem plików,
- ochronę formularzy przed atakami CSRF,
- integrację z komponentem Templating,
- integrację z komponentem Validator,
- integrację z komponentem Translation.

Formularze

- Obiekt formularza można zbudować bezpośrednio w kontrolerze lub zdefiniować jako klasę.
- Formularze powiązane są z modelami danych (np. encjami) lub tablicami.
- Dane z formularza mapowane są na obiekty lub na tablice asocjacyjne.

Budowa formularza składa się z trzech części:

- stworzenie obiektu klasy **`FormBuilder`** dla danej klasy encji,
- dodanie do obiektu **`FormBuilder`** pól, jakie mają być w formularzu (i połączenie ich z atrybutami encji),
- dodanie metody i akcji (opcjonalne),
- stworzenie formularza.

Budowa formularzy w kontrolerze

- Aby stworzyć obiekt klasy **FormBuilder** dla danej encji musimy użyć metody **createFormBuilder()**, którą dziedziczymy z podstawowej klasy kontrolera.
- Metodzie musimy przekazać obiekt naszej encji (nowy lub wczytany z bazy danych).

```
$post = new Post();
$form = $this->createFormBuilder($post);
/* ... */
```

Budowa formularzy w kontrolerze

- Aby stworzyć obiekt klasy **FormBuilder** dla danej encji musimy użyć metody **createFormBuilder()**, którą dziedziczymy z podstawowej klasy kontrolera.
- Metodzie musimy przekazać obiekt naszej encji (nowy lub wczytany z bazy danych).

```
$post = new Post();
$form = $this->createFormBuilder($post);
/* ... */
```

Tworzymy obiekt formularza, klasa **FormBuilder**

Budowa formularzy w kontrolerze

Opcjonalnie możemy dodać metodę i akcję naszego formularza. Jeżeli tego nie zrobimy, to domyślnie:

- metoda nastawiona jest na **POST**,
- akcja nastawiona jest na akcję, która stworzyła formularz.

```
$post = new Post();
$form = $this->createFormBuilder($post)
             ->setAction($this->generateUrl('target_route'))
             ->setMethod('GET');
/* ... */
```

Budowa formularzy w kontrolerze

Opcjonalnie możemy dodać metodę i akcję naszego formularza. Jeżeli tego nie zrobimy, to domyślnie:

- metoda nastawiona jest na **POST**,
- akcja nastawiona jest na akcję, która stworzyła formularz.

```
$post = new Post();
$form = $this->createFormBuilder($post)
             ->setAction($this->generateUrl('target_route'))
             ->setMethod('GET');
/* ... */
```

Dodajemy akcję naszego formularza, atrybut **action** w html

Budowa formularzy w kontrolerze

Opcjonalnie możemy dodać metodę i akcję naszego formularza. Jeżeli tego nie zrobimy, to domyślnie:

- metoda nastawiona jest na **POST**,
- akcja nastawiona jest na akcję, która stworzyła formularz.

```
$post = new Post();
$form = $this->createFormBuilder($post)
             ->setAction($this->generateUrl('target_route'))
             ->setMethod('GET');
/* ... */
```

Ustawiamy metodę HTTP jaką formularz zostanie przesłany, w przypadku braku tej linijki domyślnie **POST**

Budowa formularzy w kontrolerze

- Następnie musimy dodać poszczególne pola naszego formularza za pomocą metody **add()** wywoływanej na obiekcie **FormBuilder**.

```
$post = new Post();
$form = $this->createFormBuilder($post)
             ->setAction($this->generateUrl('target_route'))
             ->setMethod('GET');
             ->add('title', 'text')
             ->add('postText', 'text')
             ->add('save', 'submit', ['label' => 'Create post']);
/* ... */
```

Budowa formularzy w kontrolerze

- Następnie musimy dodać poszczególne pola naszego formularza za pomocą metody **add()** wywoływanej na obiekcie **FormBuilder**.

```
$post = new Post();
$form = $this->createFormBuilder($post)
             ->setAction($this->generateUrl('target_route'))
             ->setMethod('GET');
             ->add('title', 'text')
             ->add('postText', 'text')
             ->add('save', 'submit', ['label' => 'Create post']);
/* ... */
```

Nazwa pola musi być taka sama jak nazwa atrybutu w naszej encji (poza **submit**).

Budowa formularzy w kontrolerze

- Na koniec musimy wywołać **getForm()** na obiekcie **FormBuilder**.
- Sam **FormBuilder** nie będzie nam już potrzebny.

```
$post = new Post();
$form = $this->createFormBuilder($post)
             ->setAction($this->generateUrl('target_route'))
             ->setMethod('GET');
             ->add('title', 'text')
             ->add('postText', 'text')
             ->add('save', 'submit', ['label' => 'Create post'])
             ->getForm();
```

Budowa formularzy w kontrolerze

- Na koniec musimy wywołać **getForm()** na obiekcie **FormBuilder**.
- Sam **FormBuilder** nie będzie nam już potrzebny.

```
$post = new Post();
$form = $this->createFormBuilder($post)
             ->setAction($this->generateUrl('target_route'))
             ->setMethod('GET');
             ->add('title', 'text')
             ->add('postText', 'text')
             ->add('save', 'submit', ['label' => 'Create post'])
             ->getForm();
```

Wywołujemy metodę **getForm()**

Budowa formularzy w kontrolerze

- Na koniec musimy wywołać **getForm()** na obiekcie **FormBuilder**.
- Sam **FormBuilder** nie będzie nam już potrzebny.

```php
$post = new Post();
$form = $this->createFormBuilder($post)
             ->setAction($this->generateUrl('target_route'))
             ->setMethod('GET');
             ->add('title', 'text')
             ->add('postText', 'text')
             ->add('save', 'submit', ['label' => 'Create post'])
             ->getForm();
```

Wszystkie metody możemy wywoływać w chainingu

Przekazanie formularza do widoku

- Następnie musimy przekazać formularz do widoku.
- Do widoku nie przekazujemy obiektu **Form**, tylko jego widok, który otrzymujemy dzięki metodzie **createView()**.

```
$post = new Post();
$form = $this->createFormBuilder($post)
             ->setAction($this->generateUrl('target_route'))
             ->setMethod('GET');
             ->add('title', 'text')
             ->add('postText', 'text')
             ->add('save', 'submit', ['label' => 'Create post'])
             ->getForm();
```

```
return $this->render('default/new.html.twig', ['form' => $form->createView()]);
```

Wyświetlanie formularza w widoku

- Przekazaliśmy do **Twiga** nasz obiekt widoku formularza, możemy go tam zatem wyświetlić.
- Najprostszy sposób, to użycie trzech tagów **Twiga:**

```
{{ form_start(form) }}
{{ form_widget(form) }}
{{ form_end(form) }}
```

Wyświetlanie formularza w widoku

- Przekazaliśmy do **Twiga** nasz obiekt widoku formularza, możemy go tam zatem wyświetlić.
- Najprostszy sposób, to użycie trzech tagów **Twiga:**

```
{{ form_start(form) }}
{{ form_widget(form) }}
{{ form_end(form) }}
```

Gdzieś w widoku

Przechwytywanie danych z formularza

- Następnie musimy napisać kod, który przechwyci dane z formularza wypełnionego przez użytkownika.
- Musimy stworzyć taki sam formularz, jakiego przedtem użyliśmy (dlatego dobrze jest przenieść tworzenie formularza do osobnej metody).

Przechwytywanie danych z formularza

Po stworzeniu formularza musimy przekazać mu obiekt **Request**.

Formularz sam wyjmie z niego dane, a następnie uzupełni o nie obiekt encji.

Częstym podejściem jest tworzenie jednej akcji do wyświetlenia formularza i przetwarzania go.

Żeby dowiedzieć się, czy formularz został wysłany, używamy metody **isSubmitted().**

```
$form = $this->createFormBuilder()
/* ... */
->getForm();

$form->handleRequest($request);

if ($form->isSubmitted()) {
    $post = $form->getData();
    $em = $this->getDoctrine()->getManager();
    $em->persist($post);
    $em->flush();

    return $this->redirectToRoute('task_success');
}
```

Przechwytywanie danych z formularza

Po stworzeniu formularza musimy przekazać mu obiekt **Request**.

Formularz sam wyjmie z niego dane, a następnie uzupełni o nie obiekt encji.

Częstym podejściem jest tworzenie jednej akcji do wyświetlenia formularza i przetwarzania go.

Żeby dowiedzieć się, czy formularz został wysłany, używamy metody **isSubmitted().**

```php
$form = $this->createFormBuilder()
/* ... */
->getForm();

$form->handleRequest($request);

if ($form->isSubmitted()) {
    $post = $form->getData();
    $em = $this->getDoctrine()->getManager();
    $em->persist($post);
    $em->flush();

    return $this->redirectToRoute('task_success');
}
```

Uzupełnienie obiektu encji

Zadania

Zaawansowane formularze

Jak formularz łączy się z encją?

Całą potęgą formularzy w Symfony jest automatyczne połączenie formularza z naszą encją.

Dzieje się to dzięki temu, że formularz szuka z naszej encji:

- publicznego atrybutu z taką sama nazwą jak jego pole,
- publicznej metody **set** odpowiadającej nazwie pola.

Formularze

Metoda add.

Metoda **add()** tworzy nam pole w formularzu.

Musimy jej przekazać następujące dane:

- **nazwę pola** (taka sama jak atrybut, do którego odnosi się to pole),
- **typ pola,**
- **tablicę z dodatkowymi opcjami.**

Typy pól

W Symfony możemy skorzystać m.in. z następujących typów pól:

- **pola tekstowe,**
- **pola wyboru,**
- **przycisk,**
- **pola ukryte.**

Pełen spis typów pól:
http://symfony.com/doc/2.8/reference/forms/types.html

Opcje przekazywania do add()

- Do metody **add()** możemy przekazywać różne opcje (w formie tablicy podawanej jako trzeci parametr).
- Opcje te różnią się w zależności od typu pola.
- Opcje przyjmowane przez dany typ pola można znaleźć w dokumentacji.

Opcje przekazywania do add()

Jednymi z najczęściej używanych opcji są:

- **required** – określa, czy pole formularza jest wymagane (przyjmuje wartości **true** albo **false**),
- **label** – dodaje opis do pola formularza.

```
->add('dueDate', 'date', [
    'required' => true,
    'label'  => 'Due Date',
]);
```

Typ pola – entity

- Jednym z najważniejszych typów pól jest pole wyboru encji.
- Pozwala nam ono na wybranie jednej z encji danej klasy, która jest zapisana w bazie danych.
- Na stronie wyświetla się jako pole wyboru.

Typ pola – entity

Jeżeli wybraliśmy typ pola **entity**, musimy przekazać do formularza co najmniej dwie opcje:

- **class** – nazwa klasy encji budowana na zasadzie **BundleName:EntityName**
- **choice_label** – nazwa atrybutu, który będzie używany do wyświetlenia w formularzu.

```
->add('users', 'entity', [
    'class' => 'myBundle:Tag',
    'choice_label' => 'tagName',
]);
```

Typ pola – entity

- Jeżeli chcemy, żeby nasz formularz wczytywał tylko wybrane encje danej klasy, możemy przekazać mu odpowiednie zapytanie DQL.
- Robimy to przez użycie opcji **`query_builder`**.

Więcej o tej opcji możecie przeczytać tutaj:
http://symfony.com/doc/2.8/reference/forms/types/entity.html#using-a-custom-query-for-the-entities

Zaawansowane renderowanie

Możemy też wpłynąć na to, jak nasz formularz jest renderowany. Możemy wyświetlać po kolei każde jego pole.

Służą do tego odpowiednie metody **Twig:**

- **form_start()**,
- **form_label()**,
- **form_widget()**,
- **form_end()**.

```
{{ form_start(form) }}
    <div>
        {{ form_label(form.task) }}
        {{ form_widget(form.task) }}
    </div>

    <div>
        {{ form_label(form.dueDate) }}
        {{ form_widget(form.dueDate) }}
    </div>

    <div>
        {{ form_widget(form.save) }}
    </div>
{{ fsorm_end(form) }}
```

Dodatkowe informacje

W dokumentacji znajdziesz więcej informacji o formularzach. Zwłaszcza o następujących zaawansowanych zagadnieniach:

- nadpisywaniu szablonów formularzy,
- tworzeniu specjalnych klas formularzy,
- tworzeniu formularzy pod wiele encji,
- używaniu formularzy bez klasy encji.

http://symfony.com/doc/2.8/book/forms.html

Zadania