**Symfony lab4**  **Stepanov Maxim IA-1902**

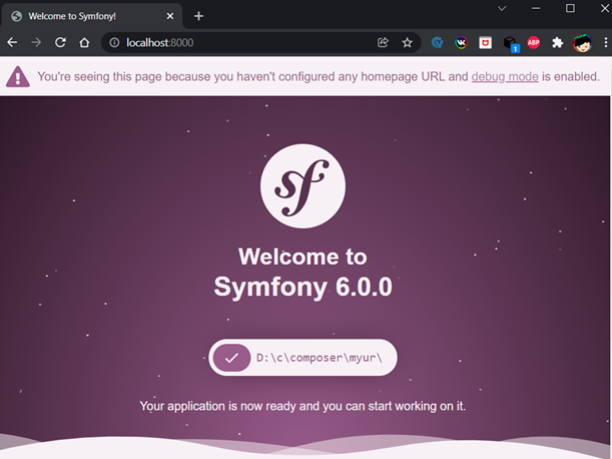
**Форма**

**1)composer create-project symfony/skeleton myus**

**2)composer require symfony/form**

**3)composer require annotations**

**4)composer require twig**



Пользователи создают и редактируют задачи используя формы Symfony. Каждая задача - это экземпляр следующего класса Task:

<?php

// src/Entity/Task.php

namespace App\Entity;

class Task

{

protected $task;

protected $dueDate;

public function getTask(): string

{

return $this->task;

}

public function setTask(string $task): void

{

$this->task = $task;

}

public function getDueDate(): ?\DateTime

{

return $this->dueDate;

}

public function setDueDate(?\DateTime $dueDate): void

{

$this->dueDate = $dueDate;

}

}

### **Создание формы в контроллере:**

Если ваш контроллер расширяется из [AbstractController](https://symfony.com.ua/doc/current/controller.html" \l "the-base-controller-class-services), используйте помощник createFormBuilder():

<?php

// src/Controller/TaskController.php

namespace App\Controller;

use App\Entity\Task;

use App\Form\Type\TaskType;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;

use Symfony\Component\Form\Extension\Core\Type\DateType;

use Symfony\Component\Form\Extension\Core\Type\SubmitType;

use Symfony\Component\Form\Extension\Core\Type\TextType;

use Symfony\Component\HttpFoundation\Request;

use Symfony\Component\HttpFoundation\Response;

use Symfony\Component\Routing\Annotation\Route;

class TaskController extends AbstractController

{

/\*\*

\* @Route("/form", name="form")

\*/

public function new(Request $request): Response

{

// создает объект задачи и инициализирует некоторые данные для этого примера

$task = new Task();

$task->setTask('Write a blog post');

$task->setDueDate(new \DateTime('tomorrow'));

$form = $this->createForm(TaskType::class, $task);

$form = $this->createFormBuilder($task)

->add('task', TextType::class)

->add('dueDate', DateType::class)

->add('save', SubmitType::class, ['label' => 'Create Task'])

->getForm();

// ...

}

}

### **Создание классов формы:**

Symfony рекомендует размещать минимально возможное количество логики в контроллерах. Вот почему лучше перемещать сложные формы в посвященные им классы, вместо того, чтобы определять их в действиях контроллера. Кроме того, формы, определенные в классах, могут быть использованы повторно в нескольких действиях и сервисах.

Классы формы - это [типы формы](https://symfony.com.ua/doc/current/forms.html" \l "form-types), которые реализуют [FormTypeInterface](https://github.com/symfony/symfony/blob/master/src/Symfony/Component/Form/FormTypeInterface.php" \o "Symfony\\Component\\Form\\FormTypeInterface). Однако, лучше расширять из [AbstractType](https://github.com/symfony/symfony/blob/master/src/Symfony/Component/Form/AbstractType.php" \o "Symfony\\Component\\Form\\AbstractType), который уже реализует интерфейс и предоставляет некоторые утилиты:

<?php

// src/Form/Type/TaskType.php

namespace App\Form\Type;

use Symfony\Component\Form\AbstractType;

use Symfony\Component\Form\Extension\Core\Type\DateType;

use Symfony\Component\Form\Extension\Core\Type\SubmitType;

use Symfony\Component\Form\Extension\Core\Type\TextType;

use Symfony\Component\Form\FormBuilderInterface;

class TaskType extends AbstractType

{

public function buildForm(FormBuilderInterface $builder, array $options): void

{

$builder

->add('task', TextType::class)

->add('dueDate', DateType::class)

->add('save', SubmitType::class)

;

}

}

Установите в своем проекте [MakerBundle](https://symfony.com/doc/current/bundles/SymfonyMakerBundle/index.html), чтобы генерировать классы формы используя команды make:form и make:registration-form.

composer require --dev symfony/maker-bundle

Поэтому, хотя это и не всегда необходимо, обычно хорошей идеей будет яdно указать опцию data\_class, добавив следующее к вашему классу типа формы:

<?php

// src/Form/Type/TaskType.php

namespace App\Form\Type;

use Symfony\Component\Form\AbstractType;

use Symfony\Component\Form\Extension\Core\Type\DateType;

use Symfony\Component\Form\Extension\Core\Type\SubmitType;

use Symfony\Component\Form\Extension\Core\Type\TextType;

use Symfony\Component\Form\FormBuilderInterface;

class TaskType extends AbstractType

{

public function buildForm(FormBuilderInterface $builder, array $options): void

{

$builder

->add('task', TextType::class)

->add('dueDate', DateType::class)

->add('save', SubmitType::class)

;

}

public function configureOptions(OptionsResolver $resolver): void

{

$resolver->setDefaults([

'data\_class' => Task::class,

]);

}

}

## Отображение формы:

Теперь, когда форма была создана, следующий шаг - отобразить ее. Вместо того, чтобы передавать весь объект формы шаблону, используйте метод createView(), чтобы построить другей объект с визуальным представлением формы:

<?php

// src/Controller/TaskController.php

namespace App\Controller;

use App\Entity\Task;

use App\Form\Type\TaskType;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;

use Symfony\Component\Form\Extension\Core\Type\DateType;

use Symfony\Component\Form\Extension\Core\Type\SubmitType;

use Symfony\Component\Form\Extension\Core\Type\TextType;

use Symfony\Component\HttpFoundation\Request;

use Symfony\Component\HttpFoundation\Response;

use Symfony\Component\Routing\Annotation\Route;

class TaskController extends AbstractController

{

/\*\*

\* @Route("/form", name="form")

\*/

public function new(Request $request): Response

{

// создает объект задачи и инициализирует некоторые данные для этого примера

$task = new Task();

$task->setTask('Write a blog post');

$task->setDueDate(new \DateTime('tomorrow'));

$form = $this->createForm(TaskType::class, $task);

$form = $this->createFormBuilder($task)

->add('task', TextType::class)

->add('dueDate', DateType::class)

->add('save', SubmitType::class, ['label' => 'Create Task'])

->getForm();

// ...

return $this->render('task/new.html.twig', [

'form' => $form->createView(),

]);

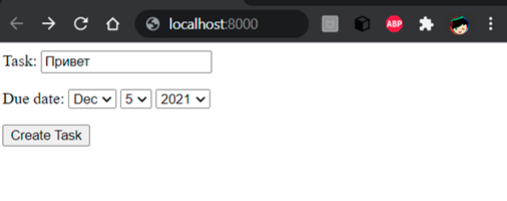
}

}

Затем, используйте какие-то [функции помощников формы](https://symfony.com.ua/doc/current/form/form_customization.html#reference-form-twig-functions), чтобы отобразить содержание формы:

{# templates/task/new.html.twig #}

{{ form(form) }}



## Валидация формы:

composer require symfony/validator

Валидация проводится путем добавления к классу набора правил, под названием (валидация) ограничения. Вы можете добавить их либо к классу сущности, либо к классу формы.

Чтобы увидеть первый подход - добавление ограничений к сущности - в действии, добавьте ограничения валидации, так, чтобы поля task и dueDate не могли быть пустыми, и последнее было валидным объектом DateTime.

<?php

// src/Entity/Task.php

namespace App\Entity;

use Symfony\Component\Validator\Constraints as Assert;

class Task

{

/\*\*

\* @Assert\NotBlank

\*/

protected $task;

/\*\*

\* @Assert\NotBlank

\* @Assert\Type("\DateTime")

\*/

protected $dueDate;

# config/validator/validation.yaml

App\Entity\Task:

properties:

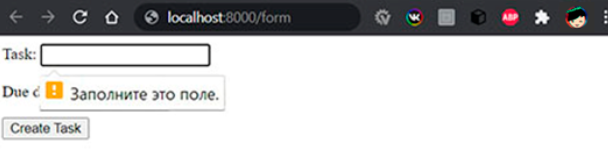
task:

- NotBlank: ~

dueDate:

- NotBlank: ~

- Type: \DateTime



### **Передача опций форме:**

Если вы [создаете формы в классах](https://symfony.com.ua/doc/current/forms.html#creating-forms-in-classes), при создании формы в контроллере, вы можете передать ей пользовательсткие опции в качестве третьего необязательного аргумента createForm():

<?php

// src/Controller/TaskController.php

namespace App\Controller;

class TaskController extends AbstractController

{

/\*\*

\* @Route("/form", name="form")

\*/

public function new(Request $request): Response

{

// создает объект задачи и инициализирует некоторые данные для этого примера

$task = new Task();

$task->setTask('Write a blog post');

$task->setDueDate(new \DateTime('tomorrow'));

//$form = $this->createForm(TaskType::class, $task);

$dueDateIsRequired = true;

$form = $this->createForm(TaskType::class, $task, [

'require\_due\_date' => $dueDateIsRequired,

]);

Если вы попробуете использовать форму сейчас, вы увидите сообщение об ошибке: Опция “require\_due\_date” не существует. Это потому что формы должны заявлять обо всех опциях, которые они принимают, используя метод configureOptions():

class TaskType extends AbstractType

{

public function buildForm(FormBuilderInterface $builder, array $options): void

{

$builder

->add('task', TextType::class)

//->add('dueDate', DateType::class)

->add('dueDate', DateType::class, [

'required' => $options['require\_due\_date'],

])

->add('save', SubmitType::class)

;

}

public function configureOptions(OptionsResolver $resolver): void

{

$resolver->setDefaults([

// ...,

'require\_due\_date' => false,

]);

// вы также можете определить позволенные типы, значения и

// любые другие функции, поддерживающиеся компонентом OptionsResolver

$resolver->setAllowedTypes('require\_due\_date', 'bool');

}

### **Опции типа формы**

Каждый [тип формы](https://symfony.com.ua/doc/current/forms.html#form-types) имеет ряд опций для его конфигурации, как объясняется в [справочнике типов формы Symfony](https://symfony.com.ua/doc/current/reference/forms/types.html). Две наиболее ипользуемые опции - это required и label.

#### **Опция required**

Наиболее распространенной опцией является required, которая может прменяться к любому полю. По умолчанию, эта опция установлена как true, что означает, что готовые к HTML5 браузеры будут требовать заполнения всех полей перед отправкой формы.

Если вы не хотите такого поведения, то либо [отключите валидацию клиентской стороны](https://symfony.com.ua/doc/current/forms.html" \l "forms-html5-validation-disable) для всей формы, либо установите опцию required как false в одном или более полях:

class TaskType extends AbstractType

{

public function buildForm(FormBuilderInterface $builder, array $options): void

{

$builder

->add('task', TextType::class)

//->add('dueDate', DateType::class)

->add('dueDate', DateType::class, [

'required' => false,

])

Опция required не выполняет валидацию серверской стороны. Если пользователь отправляет пустое значение поля (либо в старом браузере, либо веб-сервисе, к примеру), оно будет принято как валидное значение, разве что вы также не используете ограничения валидации Symfony NotBlank или NotNull. Опция label

По умолчанию, ярлык полей формы - это очеловеченная версия имени свойства (user -> User; postalAddress -> Postal Address). Установите опцию label в полях, чтобы явно определить их ярлыки:

class TaskType extends AbstractType

{

public function buildForm(FormBuilderInterface $builder, array $options): void

{

$builder

->add('task', TextType::class)

//->add('dueDate', DateType::class)

->add('dueDate', DateType::class, [

'required' => false,

'label' => 'To Be Completed Before',

])

### **Изменение действия и HTTP-метода:**

По умолчанию, форма будет отправлена через запрос HTTP POST по тому же URL, под которым отображалась форма. При создании формы в контроллере, используйте методы setAction() и setMethod(), чтобы изменить это:

$form = $this->createFormBuilder($task)

->setAction($this->generateUrl('target\_route'))

->setMethod('GET')

->add('task', TextType::class)

->add('dueDate', DateType::class)

->add('save', SubmitType::class, ['label' => 'Create Task'])

->getForm();

При создании формы в классе, передайте действие и метод в виде опций формы:

$form = $this->createForm(TaskType::class, $task, [

'require\_due\_date' => $dueDateIsRequired,

'action' => $this->generateUrl('target\_route'),

'method' => 'GET',

]);

Наконец, вы можете пеоепределить действие в шаблоне, передав их функциям помощников form() или form\_start():

{# templates/task/new.html.twig #}

{{ form(form) }}

{{ form\_start(form, {'action': path('target\_route'), 'method': 'GET'}) }}

### **Изменение имени формы:**

Если вы исследуете HTML-содержание отображенной формы, вы увидите, что имя <form> и имена полей генерируются из имени класса типа (например, <form name="task" ...> и <select name="task[dueDate][date][month]" ...>).

Если вы хотите изменить это, используйте метод [createNamed()](https://github.com/symfony/symfony/blob/master/src/Symfony/Component/Form/FormFactoryInterface.php" \o "Symfony\\Component\\Form\\FormFactoryInterface::createNamed()):

$form = $this->get('form.factory')->createNamed('my\_name', TaskType::class, $task);

### **HTML-валидация клиентской стороны:**

Благодаря HTML5, многие браузеры могут нативно устанавливать определенные ограничения валидации клиентской стороны. Наиболее распространенная валидация активируется путем добавления атрибута required в обязательные поля. Для браузеров, поддерживающих HTML5, это приведет к отображению родного сообщения браузера, если пользователь попробует отправить форму с таким пустым полем.

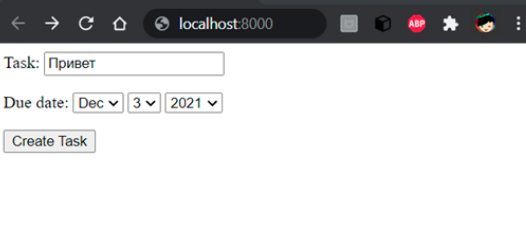
Сгенерированные формы вовсю пользуются этой новой фнукцией, добавляя правильные HTML-атрибуты, которые триггерят валидацию. Валидация клиентской стороны, однако, может быть отключена, путем добавления атрибута novalidate к тегу <form> или formnovalidate к тегу отправки. Это особенно полезно, если вы хотите протестировать ограничения валидации клиентское стороны, но ваш браузер не позволяет вам, к примеру, отправить пустые поля.

{# templates/task/new.html.twig #}

{{ form\_start(form, {'attr': {'novalidate': 'novalidate'}}) }}

{{ form\_widget(form) }}

{{ form\_end(form) }}



### **Угадывание типа формы:**[**¶**](https://symfony.com.ua/doc/current/forms.html#form-type-guessing-ru)

Если объект, обрабатываемый формой, имеет ограничения валидации, Symfony может исследовать мтеоды, чтобы угадать типа вашего поля и настроить его для вас. В примере выше, Symfony может угадать по правилам валидации, что поле task - это нормальное поле TextType, а поле dueDate - это поле DateType.

При создании формы, опустите второй аргумент метода add(), или передайте ему null, чтобы включить “механизм угадывания” Symfony:

class TaskType extends AbstractType

{

public function buildForm(FormBuilderInterface $builder, array $options): void

{

$builder

// если вы не определите опции поля, вы можете опустить второй аргумент

->add('task')

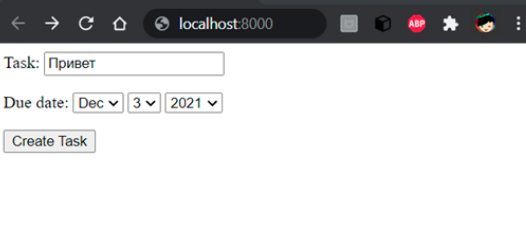
// если вы определите опции поля, передайте NULL в качестве второго аргумента

->add('dueDate', null, ['required' => false])

->add('save', SubmitType::class)

;

}



### **Неотображенные поля:**

При редактировании объекта через форму, все поля формы считаются свойствами объекта. Любые поля формы, не существующие в объекте, будут вызывать исключения.

Если вам нужны дополнительные поля в форме, которые не будут храниться в объекте (например, чтобы добавить флажок “Я согласен с этими условями” checkbox), установите опцию как false в этих полях:

class TaskType extends AbstractType

{

public function buildForm(FormBuilderInterface $builder, array $options): void

{

$builder

->add('task')

->add('dueDate')

->add('agreeTerms', CheckboxType::class, ['mapped' => false])

->add('save', SubmitType::class)

;

}

Эти “неотображенные поля” могут быть установлены и доступны в контроллере с помощью:

class TaskController extends AbstractController

{

/\*\*

\* @Route("/form", name="form")

\*/

public function new(Request $request): Response

{

// создает объект задачи и инициализирует некоторые данные для этого примера

$form->get('agreeTerms')->getData();

$form->get('agreeTerms')->setData(true);

Кроме того, если в форме есть поля, которые не добавлены в отправленные данные, эти поля будут явно установлены как null.