<https://symfony.com/doc/current/forms.html>

Formulare

Versiunea 4.3

[editați această pagină](https://github.com/symfony/symfony-docs/edit/4.3/forms.rst)

* + [Instalare](https://symfony.com/doc/current/forms.html#installation)
  + [folosire](https://symfony.com/doc/current/forms.html#usage)
    - [Tipuri de forme](https://symfony.com/doc/current/forms.html#form-types)
  + [Formulare de construcție](https://symfony.com/doc/current/forms.html#building-forms)
    - [Crearea formularelor în controlere](https://symfony.com/doc/current/forms.html#creating-forms-in-controllers)
    - [Crearea claselor de formulare](https://symfony.com/doc/current/forms.html#creating-form-classes)
  + [Formulare de redare](https://symfony.com/doc/current/forms.html#rendering-forms)
  + [Procesarea formularelor](https://symfony.com/doc/current/forms.html#processing-forms)
  + [Formulare de validare](https://symfony.com/doc/current/forms.html#validating-forms)
  + [Alte caracteristici comune ale formei](https://symfony.com/doc/current/forms.html#other-common-form-features)
    - [Trecerea opțiunilor la formulare](https://symfony.com/doc/current/forms.html#passing-options-to-forms)
    - [Opțiuni de tip formular](https://symfony.com/doc/current/forms.html#form-type-options)
    - [Modificarea metodei de acțiune și HTTP](https://symfony.com/doc/current/forms.html#changing-the-action-and-http-method)
    - [Modificarea numelui formularului](https://symfony.com/doc/current/forms.html#changing-the-form-name)
    - [Validare HTML din partea clientului](https://symfony.com/doc/current/forms.html#client-side-html-validation)
    - [Formular Guessing](https://symfony.com/doc/current/forms.html#form-type-guessing)
    - [Câmpuri nemarcate](https://symfony.com/doc/current/forms.html#unmapped-fields)
  + [Aflați mai multe](https://symfony.com/doc/current/forms.html#learn-more)

**SCREENCAST**

Preferi tutorialele video? Consultați seria de [difuzare](https://symfonycasts.com/screencast/symfony-forms) a [ecranului Symfony Forms](https://symfonycasts.com/screencast/symfony-forms) .

Crearea și procesarea formularelor HTML este greu și repetitiv. Trebuie să vă ocupați cu redarea câmpurilor de formulare HTML, validarea datelor trimise, maparea datelor formularului în obiecte și multe altele. Symfony include o caracteristică puternică de formă care oferă toate aceste caracteristici și multe altele pentru scenarii cu adevărat complexe.

Instalare [¶](https://symfony.com/doc/current/forms.html#installation)

În aplicațiile care folosesc [Symfony Flex](https://symfony.com/doc/current/setup.html#symfony-flex) , executați această comandă pentru a instala funcția de formular înainte de a o utiliza:

|  |
| --- |
| composer require symfony/form |

Utilizare [¶](https://symfony.com/doc/current/forms.html#usage)

Fluxul de lucru recomandat atunci când lucrați cu formulare Symfony este următorul:

1. **Construiți formularul** într-un controler Symfony sau utilizând o clasă de formular dedicată;
2. **Redirecționați formularul** într-un șablon, astfel încât utilizatorul să îl poată edita și trimite;
3. **Procesați formularul** pentru validarea datelor transmise, transformarea acestora în date PHP și faceți ceva cu acestea (de exemplu, persistă într-o bază de date).

Fiecare dintre aceste etape este explicat în detaliu în secțiunile următoare. Pentru a face exemple mai ușor de urmărit, toți presupun că construiți o simplă aplicație de listă Todo care afișează „sarcini”.

Utilizatorii creează și editează activități utilizând formulare Symfony. Fiecare sarcină este o instanță a Taskclasei următoare :

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28 | *// src/Entity/Task.php*  namespace App\Entity;  class Task  {  protected $task;  protected $dueDate;  public function getTask()  {  return $this->task;  }  public function setTask($task)  {  $this->task = $task;  }  public function getDueDate()  {  return $this->dueDate;  }  public function setDueDate(\DateTime $dueDate = null)  {  $this->dueDate = $dueDate;  }  }  public function setDueDate(\DateTime $dueDate = null)  {  $this->dueDate = $dueDate;  }  } |

Această clasă este un „obiect simplu-vechi-PHP” deoarece, până în prezent, nu are nicio legătură cu Symfony sau cu orice altă bibliotecă. Este un obiect PHP normal care rezolvă direct o problemă din aplicația *dvs.* (adică nevoia de a reprezenta o sarcină în aplicația dvs.). Dar puteți edita și [*entitățile Doctrine*](https://symfony.com/doc/current/doctrine.html) în același mod.

Tipuri de formular [¶](https://symfony.com/doc/current/forms.html#form-types)

Înainte de a crea prima dvs. formă Symfony, este important să înțelegeți conceptul de „tip de formular”. În alte proiecte, este comună diferențierea dintre „forme” și „câmpuri de forme”. În Symfony, toate sunt „tipuri de forme”:

* un <input type="text">câmp de formular unic este un „tip de formular” (de exemplu TextType);
* un grup de mai multe câmpuri HTML utilizate pentru introducerea unei adrese poștale este un „tip de formular” (de exemplu PostalAddressType);
* un întreg <form>cu mai multe câmpuri pentru a edita un profil de utilizator este un „tip de formular” (de exemplu UserProfileType).

Acest lucru poate fi confuz la început, dar vă va fi natural destul de curând. În plus, simplifică codul și face „compunerea” și „încorporarea” câmpurilor de forme mult mai ușor de implementat.

Există zeci de [*tipuri de formulare furnizate de Symfony*](https://symfony.com/doc/current/reference/forms/types.html) și puteți, de asemenea, [*să vă creați propriile tipuri de formulare*](https://symfony.com/doc/current/form/create_custom_field_type.html) .

Formulare de construcție [¶](https://symfony.com/doc/current/forms.html#building-forms)

Symfony oferă un obiect „constructor de forme” care vă permite să descrieți câmpurile de formular folosind o interfață fluentă. Mai târziu, acest constructor creează obiectul real de formă utilizat pentru redarea și procesarea conținutului.

Crearea formularelor în controlere [¶](https://symfony.com/doc/current/forms.html#creating-forms-in-controllers)

Dacă controlorul dvs. se extinde de la [AbstractController](https://symfony.com/doc/current/controller.html#the-base-controller-class-services) , utilizați createFormBuilder()asistentul:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28 | *// src/Controller/TaskController.php*  namespace App\Controller;  use App\Entity\Task;  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;  class TaskController extends AbstractController  {  public function new(Request $request)  {  *// creates a task object and initializes some data for this example*  $task = new Task();  $task->setTask('Write a blog post');  $task->setDueDate(new \DateTime('tomorrow'));  $form = $this->createFormBuilder($task)  ->add('task', TextType::class)  ->add('dueDate', DateType::class)  ->add('save', SubmitType::class, ['label' => 'Create Task'])  ->getForm();  *// ...*  }  } |

Dacă controlorul dvs. nu se extinde de la AbstractController, va trebui să [preiați servicii din controler](https://symfony.com/doc/current/controller.html#controller-accessing-services) și să utilizați createBuilder()metoda form.factoryserviciului.

În acest exemplu, ai adăugat două câmpuri la formularul tău - taskși dueDate - corespunzător proprietăților taskși dueDateproprietății Task clasei. De asemenea, vi s-au atribuit fiecărui [tip de formular](https://symfony.com/doc/current/forms.html#form-types) (de exemplu TextType și DateType), reprezentat de numele său de clasă complet calificat. În cele din urmă, ați adăugat un buton de trimitere cu o etichetă personalizată pentru trimiterea formularului către server.

Crearea claselor de formulare [¶](https://symfony.com/doc/current/forms.html#creating-form-classes)

Symfony recomandă să puneți cât mai puțină logică posibil în controlere. De aceea este mai bine să mutați formulare complexe la clase dedicate, în loc să le definiți în acțiunile controlorului. În plus, formularele definite în clase pot fi reutilizate în mai multe acțiuni și servicii.

Clasele de [formulare](https://symfony.com/doc/current/forms.html#form-types) sunt [tipuri de formulare](https://symfony.com/doc/current/forms.html#form-types) care se implementează [FormTypeInterface](https://github.com/symfony/symfony/blob/4.3/src/Symfony/Component/Form/FormTypeInterface.php). Cu toate acestea, este mai bine să se extindă de la [AbstractType](https://github.com/symfony/symfony/blob/4.3/src/Symfony/Component/Form/AbstractType.php), care implementează deja interfața și oferă câteva utilități:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | */ 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)  {  $builder  ->add('task', TextType::class)  ->add('dueDate', DateType::class)  ->add('save', SubmitType::class)  ;  }  } |

**BACSIS**

Instalați [MakerBundle](https://symfony.com/doc/current/bundles/SymfonyMakerBundle/index.html) în proiectul dvs. pentru a genera clase de formulare utilizând comenzile make:formși make:registration-form.

Clasa de formular conține toate direcțiile necesare pentru crearea formularului de sarcini. În controlerele care se extind de la [AbstractController](https://symfony.com/doc/current/controller.html#the-base-controller-class-services) , utilizați createForm()asistentul (în caz contrar, utilizați create()metoda form.factoryserviciului):

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | *// src/Controller/TaskController.php*  namespace App\Controller;  use App\Form\Type\TaskType;  *// ...*  class TaskController extends AbstractController  {  public function new()  {  *// creates a task object and initializes some data for this example*  $task = new Task();  $task->setTask('Write a blog post');  $task->setDueDate(new \DateTime('tomorrow'));  $form = $this->createForm(TaskType::class, $task);  *// ...*  }  } |

Fiecare formular trebuie să știe numele clasei care deține datele de bază (de exemplu App\Entity\Task). De obicei, aceasta este doar ghicită pe baza obiectului trecut la al doilea argument la createForm()(adică $task). Mai târziu, când începeți să [*încorporați formularele*](https://symfony.com/doc/current/form/embedded.html) , acest lucru nu va mai fi suficient.

Așadar, deși nu este întotdeauna necesar, este în general o idee bună să specificați data\_classopțiunea adăugând următoarele la clasa dvs. de tip formular.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | *// src/Form/Type/TaskType.php*  namespace App\Form\Type;  use App\Entity\Task;  use Symfony\Component\OptionsResolver\OptionsResolver;  *// ...*  class TaskType extends AbstractType  {  *// ...*  public function configureOptions(OptionsResolver $resolver)  {  $resolver->setDefaults([  'data\_class' => Task::class,  ]);  }  } |

Formulare de redare [¶](https://symfony.com/doc/current/forms.html#rendering-forms)

Acum că formularul a fost creat, următorul pas este redarea acestuia. În loc să treceți întregul obiect al formularului la șablon, utilizați createView()metoda pentru a construi un alt obiect cu reprezentarea vizuală a formularului:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | *// src/Controller/TaskController.php*  namespace App\Controller;  use App\Entity\Task;  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;  class TaskController extends AbstractController  {  public function new(Request $request)  {  $task = new Task();  *// ...*  $form = $this->createForm(TaskType::class, $task);  return $this->render('task/new.html.twig', [  'form' => $form->createView(),  ]);  }  } |

Apoi, utilizați câteva [funcții](https://symfony.com/doc/current/form/form_customization.html#reference-form-twig-functions) de asistență de formular pentru a redă conținutul formularului:

|  |  |
| --- | --- |
| 1  2 | *{# templates/task/new.html.twig #}*  {{ form(form) }} |

Asta e! [Funcția de formă ()](https://symfony.com/doc/current/form/form_customization.html#reference-forms-twig-form) redă toate domeniile *și* de <form>început și de sfârșit tag - uri. În mod implicit, metoda formularului este POSTși URL-ul țintă este același care a afișat formularul, dar [puteți schimba ambele](https://symfony.com/doc/current/forms.html#forms-change-action-method) .

Observați cum taskcâmpul de introducere redat are valoarea task proprietății din $taskobiect (adică „Scrieți o postare pe blog”). Aceasta este prima lucrare a unui formular: să preiați date dintr-un obiect și să le traduceți într-un format potrivit pentru a fi redate într-un formular HTML.

**BACSIS**

Sistemul de formulare este suficient de inteligent pentru a accesa valoarea taskproprietății protejate prin metode getTask()și setTask()metode din Taskclasă. Cu excepția cazului în care o proprietate este publică, aceasta *trebuie să* aibă o metodă "getter" și "setter", astfel încât Symfony să poată obține și pune date pe proprietate. Pentru o proprietate booleană, puteți utiliza o metodă "isser" sau "hasser" (de exemplu isPublished()sau hasReminder()) în loc de un getter (de exemplu getPublished()sau getReminder()).

Oricât de scurtă este această redare, nu este foarte flexibil. De obicei, veți avea nevoie de mai mult control despre aspectul întregii forme sau a unora dintre câmpurile sale. De exemplu, datorită [*integrării Bootstrap 4 cu formulare Symfony,*](https://symfony.com/doc/current/form/bootstrap4.html) puteți seta această opțiune pentru a genera forme compatibile cu cadrul Bootstrap 4 CSS:

* [YAML](https://symfony.com/doc/current/forms.html)

|  |  |
| --- | --- |
| 1  2  3 | *# config/packages/twig.yaml*  twig:  form\_themes: ['bootstrap\_4\_layout.html.twig'] |

* [XML](https://symfony.com/doc/current/forms.html)

* [PHP](https://symfony.com/doc/current/forms.html)

[Încorporat teme formă Symfony](https://symfony.com/doc/current/form/form_themes.html#symfony-builtin-forms) includ Bootstrap 3 și 4 și Fundația 5. De asemenea , puteți [crea propria temă Symfony formular](https://symfony.com/doc/current/form/form_themes.html#create-your-own-form-theme) .

În plus față de teme de formă, Symfony vă permite să [*personalizați modul în care câmpurile sunt redate*](https://symfony.com/doc/current/form/form_customization.html) cu mai multe funcții pentru a face fiecare parte a câmpului separat (widgeturi, etichete, erori, mesaje de ajutor etc.)

Procesarea formularelor [¶](https://symfony.com/doc/current/forms.html#processing-forms)

Modul [recomandat de procesare a formularelor](https://symfony.com/doc/current/best_practices.html#best-practice-handle-form) este de a utiliza o singură acțiune atât pentru redarea formularului, cât și pentru manipularea formularului trimis. Puteți utiliza acțiuni separate, dar utilizarea unei singure acțiuni simplifică totul, păstrând codul concis și mentenabil.

Prelucrarea unui formular înseamnă traducerea datelor trimise de utilizator în proprietățile unui obiect. Pentru a face acest lucru, datele transmise de la utilizator trebuie să fie scrise în obiectul formularului:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29 | *// ...*  use Symfony\Component\HttpFoundation\Request;  public function new(Request $request)  {  *// just setup a fresh $task object (remove the example data)*  $task = new Task();  $form = $this->createForm(TaskType::class, $task);  $form->handleRequest($request);  if ($form->isSubmitted() && $form->isValid()) {  *// $form->getData() holds the submitted values*  *// but, the original `$task` variable has also been updated*  $task = $form->getData();  *// ... perform some action, such as saving the task to the database*  *// for example, if Task is a Doctrine entity, save it!*  *// $entityManager = $this->getDoctrine()->getManager();*  *// $entityManager->persist($task);*  *// $entityManager->flush();*  return $this->redirectToRoute('task\_success');  }  return $this->render('task/new.html.twig', [  'form' => $form->createView(),  ]);  } |

Acest controller urmează un model comun pentru manipularea formularelor și are trei căi posibile:

1. Când încărcați inițial pagina într-un browser, formularul nu a fost încă trimis și $form->isSubmitted()se returnează false. Deci, formularul este creat și redat;
2. Când utilizatorul trimite formularul, [handleRequest()](https://github.com/symfony/symfony/blob/4.3/src/Symfony/Component/Form/FormInterface.php) recunoaște acest lucru și scrie imediat datele transmise în proprietățile taskși dueDateproprietățile $taskobiectului. Apoi acest obiect este validat (validarea este explicată în secțiunea următoare). Dacă nu este valid, [isValid()](https://github.com/symfony/symfony/blob/4.3/src/Symfony/Component/Form/FormInterface.php)se returnează falseși formularul este redat din nou, dar acum cu erori de validare;
3. Când utilizatorul trimite formularul cu date valide, datele trimise sunt scrise din nou în formular, dar de data aceasta [isValid()](https://github.com/symfony/symfony/blob/4.3/src/Symfony/Component/Form/FormInterface.php) se întoarce true. Acum aveți ocazia să efectuați unele acțiuni folosind $taskobiectul (de exemplu, persistă-l către baza de date) înainte de a redirecționa utilizatorul către o altă pagină (de exemplu, o pagină „mulțumesc” sau „succes”);

**NOTĂ**

Redirecționarea unui utilizator după trimiterea unui formular de succes este o practică optimă care împiedică utilizatorul să apeleze pe butonul „Actualizează” al browserului său și să re-posteze datele.

**PRUDENȚĂ**

createView()Metoda ar trebui să fie numit *după* handleRequest() este numit. În caz contrar, atunci când utilizați [*evenimente de formă*](https://symfony.com/doc/current/form/events.html) , modificările efectuate în \*\_SUBMITevenimente nu vor fi aplicate vizualizării (cum ar fi erorile de validare).

Dacă aveți nevoie de mai mult control cu ​​privire la momentul în care formularul dvs. este trimis sau ce date i-au fost transmise, puteți [*utiliza metoda send () pentru a gestiona trimiterile de formular*](https://symfony.com/doc/current/form/direct_submit.html) .

Validarea formularelor [¶](https://symfony.com/doc/current/forms.html#validating-forms)

În secțiunea anterioară, ați aflat cum poate fi trimis un formular cu date valide sau nevalide. În Symfony, întrebarea nu este dacă „formularul” este valid, ci dacă obiectul care stă la baza lor ( $taskîn acest exemplu) este valabil sau nu după ce formularul i-a aplicat datele trimise. Apelarea $form->isValid()este o comandă rapidă care întreabă $taskobiectul dacă are sau nu date valide.

Înainte de a utiliza validarea, adăugați suport pentru aceasta în aplicația dvs.:

|  |
| --- |
| composer require symfony/validator |

Validarea se face prin adăugarea unui set de reguli (numite restricții) la o clasă. Pentru a vedea acest lucru în acțiune, adăugați restricții de validare, astfel încât taskcâmpul nu poate fi gol și dueDatecâmpul nu poate fi gol și trebuie să fie un obiect DateTime valabil.

* [Adnotările](https://symfony.com/doc/current/forms.html)

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | *// src/Entity/Task.php*  namespace App\Entity;  use Symfony\Component\Validator\Constraints as Assert;  class Task  {  */\*\**  *\* @Assert\NotBlank*  *\*/*  public $task;  */\*\**  *\* @Assert\NotBlank*  *\* @Assert\Type("\DateTime")*  *\*/*  protected $dueDate;  } |

* [YAML](https://symfony.com/doc/current/forms.html)

* [XML](https://symfony.com/doc/current/forms.html)

* [PHP](https://symfony.com/doc/current/forms.html)

Asta e! Dacă trimiteți din nou formularul cu date nevalide, veți vedea erorile corespunzătoare tipărite cu formularul. Citiți [*documentația de validare Symfony*](https://symfony.com/doc/current/validation.html) pentru a afla mai multe despre această caracteristică puternică.

Alte caracteristici comune ale formei [¶](https://symfony.com/doc/current/forms.html#other-common-form-features)

Trecerea opțiunilor la formulare [¶](https://symfony.com/doc/current/forms.html#passing-options-to-forms)

Dacă [creați formulare în clase](https://symfony.com/doc/current/forms.html#creating-forms-in-classes) , atunci când [creați formularul în](https://symfony.com/doc/current/forms.html#creating-forms-in-classes) controler, puteți să-i treceți opțiuni personalizate ca al treilea argument opțional din createForm():

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | *// src/Controller/TaskController.php*  namespace App\Controller;  use App\Form\Type\TaskType;  *// ...*  class TaskController extends AbstractController  {  public function new()  {  $task = new Task();  *// use some PHP logic to decide if this form field is required or not*  $dueDateIsRequired = ...  $form = $this->createForm(TaskType::class, $task, [  'require\_due\_date' => $dueDateIsRequired,  ]);  *// ...*  }  } |

Dacă încercați să utilizați formularul acum, veți vedea un mesaj de eroare: *Opțiunea „require\_due\_date” nu există.*Asta pentru că formularele trebuie să declare toate opțiunile pe care le acceptă folosind configureOptions()metoda:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | *// src/Form/Type/TaskType.php*  namespace App\Form\Type;  use Symfony\Component\OptionsResolver\OptionsResolver;  *// ...*  class TaskType extends AbstractType  {  *// ...*  public function configureOptions(OptionsResolver $resolver)  {  $resolver->setDefaults([  *// ...,*  'require\_due\_date' => false,  ]);  *// you can also define the allowed types, allowed values and*  *// any other feature supported by the OptionsResolver component*  $resolver->setAllowedTypes('require\_due\_date', 'bool');  }  } |

Acum puteți utiliza această nouă opțiune de formulare în cadrul buildForm()metodei:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | *// 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\FormBuilderInterface;  class TaskType extends AbstractType  {  public function buildForm(FormBuilderInterface $builder, array $options)  {  $builder  *// ...*  ->add('dueDate', DateType::class, [  'required' => $options['require\_due\_date'],  ])  ;  }  *// ...*  } |

Opțiunile tipului formular [¶](https://symfony.com/doc/current/forms.html#form-type-options)

Fiecare [tip de formular](https://symfony.com/doc/current/forms.html#form-types) are o serie de opțiuni de configurare, așa cum este explicat în [*referința tipurilor de formulare Symfony*](https://symfony.com/doc/current/reference/forms/types.html) . Două opțiuni utilizate frecvent sunt requiredși label.

Opțiunea necesară [¶](https://symfony.com/doc/current/forms.html#the-required-option)

Cea mai comună opțiune este required opțiunea, care poate fi aplicată oricărui câmp. În mod implicit, această opțiune este setată la true, ceea ce înseamnă că browserele gata HTML5 vor trebui să completeze toate câmpurile înainte de a trimite formularul.

Dacă nu doriți acest comportament, [dezactivați validarea din partea clientului](https://symfony.com/doc/current/forms.html#forms-html5-validation-disable) pentru întregul formular sau setați requiredopțiunea falsepe unul sau mai multe câmpuri:

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

'required' => false,

])

requiredOpțiunea nu efectuează nici o validare de server-side. Dacă un utilizator trimite o valoare necompletată pentru câmp (fie cu un browser vechi sau cu un serviciu web, de exemplu), acesta va fi acceptat ca valoare valabilă, cu excepția cazului în care utilizați NotBlankși NotNullrestricțiile de validare ale Symfony .

Opțiunea label [¶](https://symfony.com/doc/current/forms.html#the-label-option)

În mod implicit, eticheta câmpurilor de formular este versiunea *umanizată* a numelui proprietății ( user-> User; postalAddress-> Postal Address). Setați labelopțiunea pe câmpuri pentru a defini etichetele lor în mod explicit:

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

*// set it to FALSE to not display the label for this field*

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

])

**BACSIS**

În mod implicit, <label>etichetele câmpurilor obligatorii sunt redate cu o requiredclasă CSS, astfel încât să puteți afișa un asterisc pentru câmpurile necesare aplicând aceste stiluri CSS:

|  |  |
| --- | --- |
| 1  2  3 | label.required:before {  content: "\*";  } |

Modificarea metodei de acțiune și HTTP [¶](https://symfony.com/doc/current/forms.html#changing-the-action-and-http-method)

În mod implicit, un formular va fi trimis prin intermediul unei solicitări POST HTTP către aceeași adresă URL sub care a fost redat formularul. Când construiți formularul în controler, utilizați setAction()și setMethod()metode pentru a schimba acest lucru:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | *// src/Controller/TaskController.php*  namespace App\Controller;  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;  class TaskController extends AbstractController  {  public function new()  {  *// ...*  $form = $this->createFormBuilder($task)  ->setAction($this->generateUrl('target\_route'))  ->setMethod('GET')  *// ...*  ->getForm();  *// ...*  }  } |

Când construiți formularul într-o clasă, treceți acțiunea și metoda ca opțiuni de formulare:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | *// src/Controller/TaskController.php*  namespace App\Controller;  use App\Form\TaskType;  use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;  class TaskController extends AbstractController  {  public function new()  {  *// ...*  $form = $this->createForm(TaskType::class, $task, [  'action' => $this->generateUrl('target\_route'),  'method' => 'GET',  ]);  *// ...*  }  } |

În cele din urmă, puteți trece peste acțiunea și metoda din șablon trecându-le la funcțiile form()sau la form\_start()asistent:

|  |  |
| --- | --- |
| 1  2 | *{# templates/task/new.html.twig #}*  {{ form\_start(form, {'action': path('target\_route'), 'method': 'GET'}) }} |

**NOTĂ**

Dacă metoda formularului nu este GETsau POST, dar PUT, PATCHsau DELETE, Symfony va insera un câmp ascuns cu numele \_method care stochează această metodă. Formularul va fi prezentat într - o normală POST cerere, dar de [*rutare Symfony lui*](https://symfony.com/doc/current/routing.html) este capabil să detecteze \_methodparametrul și îl va interpreta ca PUT, PATCHsau DELETEcerere. Consultați opțiunea [http\_method\_override](https://symfony.com/doc/current/reference/configuration/framework.html#configuration-framework-http-method-override) .

Modificarea numelui de formular [¶](https://symfony.com/doc/current/forms.html#changing-the-form-name)

Dacă inspectați conținutul HTML al formularului redat, veți vedea că <form>numele și numele de câmp sunt generate din numele clasei de tip (de exemplu <form name="task" ...>și <select name="task[dueDate][date][month]" ...>).

Dacă doriți să modificați acest lucru, utilizați [createNamed()](https://github.com/symfony/symfony/blob/4.3/src/Symfony/Component/Form/FormFactoryInterface.php) metoda:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | *// src/Controller/TaskController.php*  namespace App\Controller;  use App\Form\TaskType;  use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;  class TaskController extends AbstractController  {  public function new()  {  $task = ...;  $form = $this->get('form.factory')->createNamed('my\_name', TaskType::class, $task);  *// ...*  }  } |

Puteți chiar suprima numele complet setându-l pe un șir gol.

Validare HTML din partea clientului [¶](https://symfony.com/doc/current/forms.html#client-side-html-validation)

Datorită HTML5, multe browsere pot aplica în mod nativ anumite constrângeri de validare din partea clientului. Cea mai comună validare este activată prin adăugarea unui requiredatribut pe câmpurile care sunt obligatorii. Pentru browserele care acceptă HTML5, acest lucru va duce la afișarea unui mesaj de browser nativ dacă utilizatorul încearcă să trimită formularul cu acel câmp necompletat.

Formele generate profită din plin de această nouă caracteristică adăugând atribute HTML sensibile care declanșează validarea. Validarea din partea clientului, însă, poate fi dezactivată adăugând novalidateatributul la <form>etichetă sau formnovalidatela eticheta de trimitere. Acest lucru este util mai ales când doriți să testați restricțiile de validare din partea serverului, dar sunt împiedicate de browserul dvs., de exemplu, să trimită câmpuri necompletate.

|  |  |
| --- | --- |
| 1  2  3  4 | *{# templates/task/new.html.twig #}*  {{ form\_start(form, {'attr': {'novalidate': 'novalidate'}}) }}  {{ form\_widget(form) }}  {{ form\_end(form) }} |

Ghicirea tipului de formular [¶](https://symfony.com/doc/current/forms.html#form-type-guessing)

Dacă obiectul gestionat de formular include constrângeri de validare, Symfony poate introspecta acele metadate pentru a ghici tipul câmpului dvs. și să îl setați pentru dvs. În exemplul de mai sus, Symfony poate ghici din regulile de validare că atât taskcâmpul este un TextTypecâmp normal, cât și dueDatecâmpul este un DateTypecâmp.

Când construiți formularul, omiteți al doilea argument al add()metodei sau treceți nullla acesta, pentru a activa „mecanismul de ghicire” al Symfony:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | *// 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)  {  $builder  *// if you don't define field options, you can omit the second argument*  ->add('task')  *// if you define field options, pass NULL as second argument*  ->add('dueDate', null, ['required' => false])  ->add('save', SubmitType::class)  ;  }  } |

**PRUDENȚĂ**

Când utilizați un [*grup de validare*](https://symfony.com/doc/current/form/validation_groups.html) specific de [*formular*](https://symfony.com/doc/current/form/validation_groups.html) , ghicitorul de tip de câmp va lua în considerare *toate* restricțiile de validare atunci când ghiciți tipurile de câmp (inclusiv constrângerile care nu fac parte din grupul (grupurile) de validare).

Opțiuni de tip formular de ghicire [¶](https://symfony.com/doc/current/forms.html#form-type-options-guessing)

Când mecanismul de ghicire este activat pentru un anumit câmp (adică omiteți sau treceți nullca al doilea argument la add()), pe lângă tipul său de formă, pot fi ghicite și următoarele opțiuni:

required

requiredOpțiunea poate fi ghicit pe baza regulilor de validare ( de exemplu , este câmpul NotBlanksau NotNull) sau metadatele Doctrina (adică este câmpul nullable). Acest lucru este foarte util, deoarece validarea dvs. din partea clientului se va potrivi automat regulilor de validare.

maxlength

Dacă câmpul este un fel de câmp text, atunci maxlengthatributul opțiune poate fi ghicit din constrângerile de validare (dacă este Lengthsau Rangeeste utilizat) sau din metadatele [*Doctrinei*](https://symfony.com/doc/current/doctrine.html) (prin lungimea câmpului).

Dacă doriți să schimbați una dintre valorile ghicite, treceți-o peste trecerea opțiunii din câmpul de opțiuni:

->add('task', null, ['attr' => ['maxlength' => 4]])

**Nouă în versiunea 4.3:**Pornind de la Symfony 4.3, metadatele [*Doctrinei*](https://symfony.com/doc/current/doctrine.html) sunt introvăzute pentru a adăuga [restricții de validare automată](https://symfony.com/doc/current/doctrine.html#automatic-object-validation) .

Câmpuri nemarcate [¶](https://symfony.com/doc/current/forms.html#unmapped-fields)

La editarea unui obiect prin intermediul unui formular, toate câmpurile de formular sunt considerate proprietăți ale obiectului. Orice câmpuri de pe formularul care nu există pe obiect va determina să fie aruncată o excepție.

Dacă aveți nevoie de câmpuri suplimentare în formularul care nu va fi stocat în obiect (de exemplu, pentru a adăuga o casetă de selectare *„Sunt de acord cu acești termeni* ), setați mapped opțiunea false în aceste câmpuri:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | use Symfony\Component\Form\FormBuilderInterface;  public function buildForm(FormBuilderInterface $builder, array $options)  {  $builder  ->add('task')  ->add('dueDate')  ->add('agreeTerms', CheckboxType::class, ['mapped' => false])  ->add('save', SubmitType::class)  ;  } |

Aceste „câmpuri nemarcate” pot fi setate și accesate într-un controler cu:

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

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

În plus, dacă există câmpuri din formular care nu sunt incluse în datele transmise, aceste câmpuri vor fi setate în mod explicit null.

Aflați mai multe [¶](https://symfony.com/doc/current/forms.html#learn-more)

Când creați formulare, rețineți că primul obiectiv al unui formular este de a traduce date de la un obiect ( Task) la un formular HTML, astfel încât utilizatorul să poată modifica aceste date. Al doilea obiectiv al unui formular este să preiați datele transmise de utilizator și să le reaplicați la obiect.

Există multe mai multe de învățat și o mulțime de trucuri *puternice* în formele Symfony:

Referinţă:

* [Tipuri de formulare Referință](https://symfony.com/doc/current/reference/forms/types.html)

Caracteristici avansate:

* [Cum se încarcă fișiere](https://symfony.com/doc/current/controller/upload_file.html)
* [Cum să implementăm protecția CSRF](https://symfony.com/doc/current/security/csrf.html)
* [Cum să accesați servicii sau să configurați din interiorul unui formular](https://symfony.com/doc/current/form/form_dependencies.html)
* [Cum se creează un tip de câmp de formular personalizat](https://symfony.com/doc/current/form/create_custom_field_type.html)
* [Mod de utilizare a transformatoarelor de date](https://symfony.com/doc/current/form/data_transformers.html)
* [Când și cum se utilizează mapele de date](https://symfony.com/doc/current/form/data_mappers.html)
* [Cum se creează o extensie de tip formular](https://symfony.com/doc/current/form/create_form_type_extension.html)
* [Crearea unui Guesser tip personalizat](https://symfony.com/doc/current/form/type_guesser.html)

Formulare Teme și personalizare:

* [Bootstrap 4 Forma Tema](https://symfony.com/doc/current/form/bootstrap4.html)
* [Cum să personalizați redarea formularelor](https://symfony.com/doc/current/form/form_customization.html)
* [Cum să lucrezi cu teme de formă](https://symfony.com/doc/current/form/form_themes.html)

Evenimente:

* [Formați evenimente](https://symfony.com/doc/current/form/events.html)
* [Cum să modificați dinamic formularele folosind evenimente de formulare](https://symfony.com/doc/current/form/dynamic_form_modification.html)

Validare:

* [Cum să definiți grupurile de validare de utilizat](https://symfony.com/doc/current/form/validation_groups.html)
* [Cum să configurați dinamic grupurile de validare a formularelor](https://symfony.com/doc/current/form/validation_group_service_resolver.html)
* [Cum să alegeți grupuri de validare pe baza butonului clic](https://symfony.com/doc/current/form/button_based_validation.html)
* [Cum dezactivați validarea datelor trimise](https://symfony.com/doc/current/form/disabling_validation.html)

Diverse .:

* [Cum să utilizați funcția de trimitere () pentru a gestiona trimiterile de formular](https://symfony.com/doc/current/form/direct_submit.html)
* [Cum să încorporați formularele](https://symfony.com/doc/current/form/embedded.html)
* [Cum să încorporați o colecție de formulare](https://symfony.com/doc/current/form/form_collections.html)
* [Cum se reduce duplicarea codului cu „ereditate\_data”](https://symfony.com/doc/current/form/inherit_data_option.html)
* [Cum să trimiteți un formular cu mai multe butoane](https://symfony.com/doc/current/form/multiple_buttons.html)
* [Cum unitatea Testează-ți formularele](https://symfony.com/doc/current/form/unit_testing.html)
* [Cum să configurați datele goale pentru o clasă de formulare](https://symfony.com/doc/current/form/use_empty_data.html)
* [Cum să utilizați un formular fără o clasă de date](https://symfony.com/doc/current/form/without_class.html)