<https://symfony.com.ua/doc/current/quick_tour/flex_recipes.html>

# **Flex: creează-ți aplicația**[**¶**](https://symfony.com.ua/doc/current/quick_tour/flex_recipes.html#flex)

După ce ai citit prima parte a acestui tutorial, ai decis că Symfony merită încă 10 minute. O alegere grozavă! În această a doua parte, veți afla despre Symfony Flex: un instrument grozav care face ca adăugarea de noi funcții să fie atât de ușoară precum rularea unei comenzi. De asemenea, este și motivul pentru care Symfony este ideal pentru un mic serviciu mic sau o aplicație uriașă. Interesat? Excelent!

## **Symfony: Începeți cu Micro!**[**¶**](https://symfony.com.ua/doc/current/quick_tour/flex_recipes.html#symfony)

Dacă nu construiți o API curată (mai multe despre asta mai târziu!), Cel mai probabil veți dori să afișați HTML. Pentru a face acest lucru, veți folosi [Twig](https://twig.symfony.com/) . Twig este un motor de șabloane flexibil, rapid și sigur pentru PHP. Îți face șabloanele mai lizibile și mai compacte; de asemenea, le face mai prietenoase pentru designeri web.

Twig este deja instalat în aplicația noastră? De fapt, nu încă! Și acest lucru este minunat! Când începeți un nou proiect Symfony, este mic : doar cele mai critice dependențe sunt incluse în fișierul dvs composer.json.:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | "require": {  "...",  "symfony/console": "^4.1",  "symfony/flex": "^1.0",  "symfony/framework-bundle": "^4.1",  "symfony/yaml": "^4.1"  } |

Acest lucru diferențiază Symfony de orice alt cadru PHP! În loc să începi cu o aplicație greoaie cu fiecare caracteristică posibilă de care ai putea avea nevoie, aplicația Symfony este mică, simplă și rapidă . Și aveți control deplin asupra a ceea ce adăugați.

## **Rețete și nume suplimentare Flex**[**¶**](https://symfony.com.ua/doc/current/quick_tour/flex_recipes.html#id1)

Deci, cum instalăm și configurăm Twig? Rulați doar o comandă:

|  |
| --- |
| composer require twig |

Două lucruri foarte interesante se întâmplă în culise datorită Symfony Flex: plugin-ul Composer instalat deja în proiectul nostru.

În primul rând, twigacesta nu este numele pachetului Composer: este numele opțional Flex care indică symfony/twig-bundle. Flex rezolvă acest nume alternativ pentru Composer.

Și în al doilea rând, Flex stabilește rețeta pentru symfony/twig-bundle. Ce este o rețetă? Acesta este un mod prin care biblioteca se configurează automat prin adăugarea și modificarea fișierelor. Datorită rețetelor, adăugarea funcțiilor este perfectă și automatizată: instalați pachetul și ați terminat!

Puteți găsi o listă completă de rețete și nume suplimentare accesând [https://symfony.sh](https://symfony.sh/) .

Ce a făcut această rețetă? Pe lângă conectarea automată a funcției config/bundles.php, el a adăugat 3 lucruri:

config/packages/twig.yaml

Un fișier de configurare care instalează Twig cu setări implicite rezonabile.

config/routes/dev/twig.yaml

Un traseu care vă ajută să depanați paginile de eroare.

templates/

Directorul în care vor trăi fișierele șablon. Rețeta a adăugat și un fișier format base.html.twig.

In proiectul de la lectia trecuta

## **TWIG(in ro Crenguță): șablon afișare**[**¶**](https://symfony.com.ua/doc/current/quick_tour/flex_recipes.html#twig)

Datorită Flex, după o singură comandă, puteți începe imediat să folosiți Twig:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | // src/Controller/DefaultController.php  // ...  + use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;  -class DefaultController  +class DefaultController extends AbstractController  {  /\*\*  \* @Route("/hello/{name}")  \*/  public function index($name)  {  - return new Response("Hello $name!");  + return $this->render('default/index.html.twig', [  + 'name' => $name,  + ]);  } |

După extindere AbstractController, acum aveți acces la o listă de metode și instrumente de reducere, cum ar fi render(). Creați un nou șablon:

|  |  |
| --- | --- |
| 1  2 | {# templates/default/index.html.twig #}  <h1>Привет, {{ name }}</h1> |

Asta este totul! Sintaxa va {{ name }}afișa variabila namecare este transmisă de la controler. Dacă sunteți nou în Twig, atunci bineveniți! Veți afla mai multe despre sintaxa și puterea acesteia mai târziu.

Dar chiar acum. pagina conține doar eticheta h1. Pentru a-i oferi un aspect HTML, extindeți base.html.twig:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | {# templates/default/index.html.twig #}  {% extends 'base.html.twig' %}  {% block body %}  <h1>Привет, {{ name }}</h1>  {% endblock %} |

Aceasta se numește moștenire șablon: pagina noastră moștenește acum structura HTML de la base.html.twig.

## **Profiler: Debugging Paradise**[**¶**](https://symfony.com.ua/doc/current/quick_tour/flex_recipes.html#id2)

Una dintre cele mai cool caracteristici ale Symfony nu a fost încă instalată! Să o rezolvăm:

|  |
| --- |
| composer require profiler  nu lucreaza!!!!  Cere  composer req symfony / twig-bundle |

Da! Acesta este un alt aliat! De asemenea, Flex instalează o altă rețetă care automatizează configurația Symfony Profiler. Care este rezultatul? Actualizează!

Vezi mai jos acest panou negru? Aceasta este o bară de instrumente de depanare web și este noul tău cel mai bun prieten. Afișând fiecare pictogramă, puteți obține informații despre ce controler a fost completat, informații despre performanță, noroc și memorie în cache, și multe altele. Faceți clic pe orice pictogramă pentru a intra în profiler , de unde puteți obține date și mai detaliate despre depanare și performanță!

Și, în timpul instalării de noi biblioteci, veți primi mai multe instrumente (de exemplu, pictograma barei de instrumente de depanare web care afișează interogări ale bazei de date).

Utilizarea profilatorului este simplă, deoarece s-a configurat ea însăși datorită rețetei. Ce altceva putem instala la fel de ușor?

## **suport bogat pentru API**[**¶**](https://symfony.com.ua/doc/current/quick_tour/flex_recipes.html#api)

Construiți o API? Puteți returna deja cu ușurință JSON de la orice controler:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | /\*\*  \* @Route("/api/hello/{name}")  \*/  public function apiExample($name)  {  return $this->json([  'name' => $name,  'symfony' => 'rocks',  ]);  } |

Dar pentru o API cu adevărat bogată, încercați să instalați [platforma Api](https://api-platform.com/) :

|  |
| --- |
| composer require api |

Acesta este un nume suplimentar api-platform/api-packcare are dependențe în mai multe alte pachete, cum ar fi Symfony Validator și componente de securitate, precum și Doctrine ORM. De fapt, Flex a stabilit 5 rețete!

Dar, ca de obicei, putem începe imediat utilizarea noii biblioteci. Vrei să creezi o API bogată pentru un tabel product? Creați o entitate Productși dați adnotarea @ApiResource():

|  |  |
| --- | --- |
| 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  30  31 | // src/Entity/Product.php  // ...  use ApiPlatform\Core\Annotation\ApiResource;  use Doctrine\ORM\Mapping as ORM;  /\*\*  \* @ORM\Entity()  \* @ApiResource()  \*/  class Product  {  /\*\*  \* @ORM\Id  \* @ORM\GeneratedValue(strategy="AUTO")  \* @ORM\Column(type="integer")  \*/  private $id;  /\*\*  \* @ORM\Column(type="string")  \*/  private $name;  /\*\*  \* @ORM\Column(type="string")  \*/  private $price;  // ...  } |

Gata! Acum aveți puncte finale pentru listarea, adăugarea, actualizarea și ștergerea produselor! Nu mă credeți? Enumerați-vă rutele făcând:

php bin/console debug:router

|  |
| --- |
|  |
| 1  2  3  4  5  6  7  8  9  10 | ------------------------------ -------- -------------------------------------  Имя Метод Путь  ------------------------------ -------- -------------------------------------  api\_products\_get\_collection GET /api/products.{\_format}  api\_products\_post\_collection POST /api/products.{\_format}  api\_products\_get\_item GET /api/products/{id}.{\_format}  api\_products\_put\_item PUT /api/products/{id}.{\_format}  api\_products\_delete\_item DELETE /api/products/{id}.{\_format}  ...  ------------------------------ -------- ------------------------------------- |

## **Rețete ușor de șters**[**¶**](https://symfony.com.ua/doc/current/quick_tour/flex_recipes.html#id3)

Încă nu ești convins? Nu este o problemă: eliminați biblioteca:

|  |
| --- |
| composer remove api |

Flex va șterge rețetele: ștergerea fișierelor și eliminarea modificărilor pentru a returna aplicația la starea inițială. Experimentați fără teamă.

## **Mai multe caracteristici, arhitectură și viteză**[**¶**](https://symfony.com.ua/doc/current/quick_tour/flex_recipes.html#id4)

Sper că sunteți la fel de fericiți de Flex ca mine! Dar mai există un singur capitol și va fi cel mai important dintre toate. Vreau să vă arăt cum Symfony vă oferă posibilitatea de a construi rapid funcții, fără a sacrifica calitatea sau performanța codului. Totul se referă la containerul de servicii, iar acesta este superputerea Symfony. Citiți mai multe despre [Arhitectura](https://symfony.com.ua/doc/current/quick_tour/the_architecture.html) .