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

# **Arhitectură**[**¶**](https://translate.googleusercontent.com/translate_c?depth=1&rurl=translate.google.com&sl=auto&sp=nmt4&tl=ro&u=https://symfony.com.ua/doc/current/quick_tour/the_architecture.html&xid=17259,15700022,15700186,15700191,15700256,15700259,15700262,15700265,15700271,15700283&usg=ALkJrhjdpAcdKDR3BTLxCfjHO0cfWpir5g#id1)

Tu ești eroul meu! Cine ar fi crezut că veți mai fi aici după primele trei părți? Eforturile tale vor fi curând răsplătite. Primele trei părți nu au aprofundat în arhitectura cadrului. Deoarece diferențiază Symfony de mulțimea de cadre, să ne scufundăm acum în lumea arhitecturii.

## Adăugarea jurnalului [¶](https://translate.googleusercontent.com/translate_c?depth=1&rurl=translate.google.com&sl=auto&sp=nmt4&tl=ro&u=https://symfony.com.ua/doc/current/quick_tour/the_architecture.html&xid=17259,15700022,15700186,15700191,15700256,15700259,15700262,15700265,15700271,15700283&usg=ALkJrhjdpAcdKDR3BTLxCfjHO0cfWpir5g#id2)

Noua aplicație Symfony este microscopică: constă în esență doar dintr-un sistem de rutare și un controler. Dar, datorită lui Flex, instalarea de noi funcții este ușoară.

Vrei un sistem de logare? Nu este o problemă:

|  |
| --- |
| composer require logger |

Aceasta instalează și configurează (prin rețetă) puternica bibliotecă [Monolog](https://translate.googleusercontent.com/translate_c?depth=1&rurl=translate.google.com&sl=auto&sp=nmt4&tl=ro&u=https://github.com/Seldaek/monolog&xid=17259,15700022,15700186,15700191,15700256,15700259,15700262,15700265,15700271,15700283&usg=ALkJrhg2o0XMDM6qbUByvaSRUzCPRZ4xLw) . Pentru a utiliza loggerul în controler, adăugați un nou argument introdus de LoggerInterface :

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | // ...  public function index($name, LoggerInterface $logger)  {  $logger->info("Saying hello to $name!");  // ...  } |

Asta este totul! Un nou mesaj de jurnal va fi scris pe var/log/dev.log . Desigur, acest lucru poate fi configurat prin actualizarea unuia dintre fișierele de configurare adăugate de rețetă.

## Servicii și instalare de mașini [¶](https://translate.googleusercontent.com/translate_c?depth=1&rurl=translate.google.com&sl=auto&sp=nmt4&tl=ro&u=https://symfony.com.ua/doc/current/quick_tour/the_architecture.html&xid=17259,15700022,15700186,15700191,15700256,15700259,15700262,15700265,15700271,15700283&usg=ALkJrhjdpAcdKDR3BTLxCfjHO0cfWpir5g#id3)

Dar hei! S-a întâmplat ceva foarte fain. Symfony a citit tastarea LoggerInterface și a realizat automat că ar trebui să ne treacă obiectul Logger! Aceasta se numește montaj automat .

Fiecare parte a lucrării care se realizează în aplicația Symfony este realizată de obiect : jurnalul obiectului Logger, iar obiectul Twig afișează șabloanele. Aceste obiecte se numesc servicii și sunt instrumente care vă ajută să construiți funcții bogate.

Pentru a face viața uimitoare, puteți cere lui Symfony să vă transmită serviciul folosind tastarea. Ce alte clase sau interfețe posibile puteți utiliza? Aflați făcând:

|  |  |
| --- | --- |
| php bin / console debug: conectare automată | |
| **Clasa / tip de interfață** | **ID de serviciu aliat** | |
| Psr\Cache\CacheItemPoolInterface | aliat pentru „cache.app.recorder” | |
| Psr\Log\LoggerInterface | aliat pentru „monolog.logger” | |
| Symfony\Component\EventDispatcher\EventDispatcherInterface | aliat pentru „debug.event\_dispatcher” | |
| Symfony\Component\HttpFoundation\RequestStack | aliat pentru „request\_stack” | |
| Symfony\Component\HttpFoundation\Session\SessionInterface | aliat pentru „sesiune” | |
| Symfony\Component\Routing\RouterInterface | aliat pentru „router.default” | |

Acesta este doar un scurt rezumat al listei complete! Și pe măsură ce adăugați pachete, această listă va crește și ea!

## Crearea serviciilor [¶](https://translate.googleusercontent.com/translate_c?depth=1&rurl=translate.google.com&sl=auto&sp=nmt4&tl=ro&u=https://symfony.com.ua/doc/current/quick_tour/the_architecture.html&xid=17259,15700022,15700186,15700191,15700256,15700259,15700262,15700265,15700271,15700283&usg=ALkJrhjdpAcdKDR3BTLxCfjHO0cfWpir5g#id4)

Pentru a vă menține codul eficient, puteți chiar să vă creați propriile servicii! Imaginați-vă că doriți să generați un salut aleatoriu (de exemplu, „Hello”, „Yo”, etc.). În loc să introduceți acest cod direct în controler, creați o clasă nouă:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | // src/GreetingGenerator.php  namespace App;  class GreetingGenerator  {  public function getRandomGreeting()  {  $greetings = ['Hey', 'Yo', 'Aloha'];  $greeting = $greetings[array\_rand($greetings)];  return $greeting;  }  } |

Excelent! Puteți utiliza acest lucru în controlerul dvs. imediat:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | use App\GreetingGenerator;  // ...  public function index($name, LoggerInterface $logger, GreetingGenerator $generator)  {  $greeting = $generator->getRandomGreeting();  $logger->info("Saying $greeting to $name!");  // ...  } |

Asta este totul! Symfony va iniția automat GreetingGenerator și o va transmite ca argument. Dar, putem muta și logica loggerului în GreetingGenerator ? Da! Puteți utiliza montajul automat în interiorul serviciului pentru a avea acces la alte servicii. Singura diferență este că acest lucru este realizat în constructor:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | + use Psr\Log\LoggerInterface;  class GreetingGenerator  {  + private $logger;  +  + public function \_\_construct(LoggerInterface $logger)  + {  + $this->logger = $logger;  + }  public function getRandomGreeting()  {  // ...  + $this->logger->info('Using the greeting: '.$greeting);  return $greeting;  }  } |

Da! Acest lucru funcționează și: fără configurare, fără pierdere de timp. Continuați să scrieți codul!

## Extensie și configurare automată Ramură [¶](https://translate.googleusercontent.com/translate_c?depth=1&rurl=translate.google.com&sl=auto&sp=nmt4&tl=ro&u=https://symfony.com.ua/doc/current/quick_tour/the_architecture.html&xid=17259,15700022,15700186,15700191,15700256,15700259,15700262,15700265,15700271,15700283&usg=ALkJrhjdpAcdKDR3BTLxCfjHO0cfWpir5g#twig)

Datorită procesării serviciilor Symfony, puteți extinde Symfony în multe feluri, cum ar fi crearea unui abonat de evenimente sau un selector de securitate pentru reguli de autorizare complexe. Să adăugăm un nou filtru numit greet în Twig. Cum? Creează doar o clasă care extinde AbstractExtension :

|  |  |
| --- | --- |
| 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 | // src/Twig/GreetExtension.php  namespace App\Twig;  use App\GreetingGenerator;  use Twig\Extension\AbstractExtension;  use Twig\TwigFilter;  class GreetExtension extends AbstractExtension  {  private $greetingGenerator;  public function \_\_construct(GreetingGenerator $greetingGenerator)  {  $this->greetingGenerator = $greetingGenerator;  }  public function getFilters()  {  return [  new TwigFilter('greet', [$this, 'greetUser']),  ];  }  public function greetUser($name)  {  $greeting = $this->greetingGenerator->getRandomGreeting();  return "$greeting $name!";  }  } |

După crearea unui singur fișier, puteți utiliza acest lucru imediat:

|  |  |
| --- | --- |
| 1  2 | {# Отобразит что-то вроде "Привет, Symfony!" #}  <h1>{{ name|greet }}</h1> |

Cum funcționează? Symfony observă că clasa dvs. extinde AbstractExtension și, prin urmare, o înregistrează automat ca o extensie Twig. Aceasta se numește autoconfigurare și funcționează pentru atâtea lucruri. Creează doar o clasă, apoi extinde clasa de bază (sau implementează interfața). Symfony are grijă de restul.

## Viteza vântului: container în cache [¶](https://translate.googleusercontent.com/translate_c?depth=1&rurl=translate.google.com&sl=auto&sp=nmt4&tl=ro&u=https://symfony.com.ua/doc/current/quick_tour/the_architecture.html&xid=17259,15700022,15700186,15700191,15700256,15700259,15700262,15700265,15700271,15700283&usg=ALkJrhjdpAcdKDR3BTLxCfjHO0cfWpir5g#id5)

Când vedeți cât de mult face Symfony în mod automat, ați putea gândi „Nu afectează performanța?” De fapt, nu! Symfony rulează cu viteza vântului.

Cum este posibil acest lucru? Un sistem de servicii este gestionat de un obiect foarte important numit container. Majoritatea cadrelor au un container, dar în Symfony este unic, deoarece este în cache . Când încărcați prima pagină, toate informațiile despre servicii au fost compilate și salvate. Aceasta înseamnă că funcțiile de montare automată și configurare automată nu se adaugă la sarcină! Înseamnă, de asemenea, că primiți erori excelente : Symfony examinează și validează totul când este construit containerul.

Acum s-ar putea să vă întrebați ce se întâmplă când actualizați fișierul și cache-ul trebuie să fie reconstruit? Îmi place gândirea ta! El este suficient de inteligent pentru a construi pe pagina următoare de încărcare. Dar de fapt acesta este subiectul secțiunii următoare.

## Dezvoltare versus producție: medii [¶](https://translate.googleusercontent.com/translate_c?depth=1&rurl=translate.google.com&sl=auto&sp=nmt4&tl=ro&u=https://symfony.com.ua/doc/current/quick_tour/the_architecture.html&xid=17259,15700022,15700186,15700191,15700256,15700259,15700262,15700265,15700271,15700283&usg=ALkJrhjdpAcdKDR3BTLxCfjHO0cfWpir5g#id6)

Una dintre principalele sarcini ale cadrului este facilitarea depanării! Și aplicația noastră este plină de instrumente excelente pentru asta: bara de instrumente de depanare web se afișează în partea de jos a paginii, erorile sunt mari, frumoase și complete și orice configurare a cache-ului este reconstruită automat dacă este necesar.

Dar dacă vă desfășurați în producție? Va trebui să ascundem aceste instrumente și să optimizăm viteza!

Acest lucru este rezolvat de sistemul de mediu Symfony și există trei dintre ele: dev , prod și test . În funcție de mediu, Symfony încarcă diferite fișiere în directorul config/ :

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | config/  ├─ services.yaml  ├─ ...  └─ packages/  ├─ framework.yaml  ├─ ...  ├─ \*\*dev/\*\*  ├─ monolog.yaml  └─ ...  ├─ \*\*prod/\*\*  └─ monolog.yaml  └─ \*\*test/\*\*  ├─ framework.yaml  └─ ...  └─ routes/  ├─ annotations.yaml  └─ \*\*dev/\*\*  ├─ twig.yaml  └─ web\_profiler.yaml |

Aceasta este o idee puternică : schimbând o parte a configurației (mediu), aplicația dvs. se transformă dintr-o experiență plăcută de depanare într-una optimizată pentru viteză.

Oh, și cum să schimbi mediul? Modificați variabila de mediu APP\_ENV la dev la prod :

|  |  |
| --- | --- |
| 1  2  3 | # .env  - APP\_ENV=dev  + APP\_ENV=prod |

Dar vreau să vorbesc mai multe despre variabilele de mediu. Modificați valoarea înapoi pentru dev : instrumentele de depanare sunt grozave atunci când lucrați local.

## Variabile de mediu [¶](https://translate.googleusercontent.com/translate_c?depth=1&rurl=translate.google.com&sl=auto&sp=nmt4&tl=ro&u=https://symfony.com.ua/doc/current/quick_tour/the_architecture.html&xid=17259,15700022,15700186,15700191,15700256,15700259,15700262,15700265,15700271,15700283&usg=ALkJrhjdpAcdKDR3BTLxCfjHO0cfWpir5g#id7)

Fiecare aplicație conține o configurație diferită pe fiecare server - cum ar fi informații despre conexiunile bazelor de date sau parolele. Cum trebuie depozitate? În fișiere? Sau în alt fel?

Symfony urmărește cele mai bune practici ale industriei prin stocarea configurației bazate pe server sub formă de variabile de mediu . Aceasta înseamnă că Symfony funcționează excelent cu sistemele de implementare a Platformei ca serviciu (PaaS), precum și cu Docker.

Dar setarea variabilelor de mediu în timpul dezvoltării poate fi stresantă. Prin urmare, aplicația noastră descarcă automat fișierul .env dacă variabila de mediu APP\_ENV nu APP\_ENV setată în mediu. Tastele din acest fișier devin apoi variabile de mediu și sunt citite de aplicația dvs.:

|  |
| --- |
| # .env  ###> symfony / framework-bundle ###  APP\_ENV = dev  APP\_SECRET = cc86c7ca937636d5ddf1b754beb22a10  ### <symfony / framework-bundle ### |

La început, fișierul nu conține multe. Dar odată cu creșterea aplicației dvs., veți adăuga mai multe configurații, după cum este necesar. Dar, de fapt, devine mult mai interesant! Imaginează-ți că aplicația ta are nevoie de o bază de date ORM. Să instalăm ORD Doctrine:

|  |
| --- |
| compozitor necesită doctrină |

Datorită noii rețete instalate de Flex, uitați- .env din nou la fișierul .env :

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | ###> symfony/framework-bundle ###  APP\_ENV=dev  APP\_SECRET=cc86c7ca937636d5ddf1b754beb22a10  ###< symfony/framework-bundle ###  + ###> doctrine/doctrine-bundle ###  + # ...  + DATABASE\_URL=mysql://db\_user:db\_password@127.0.0.1:3306/db\_name  + ###< doctrine/doctrine-bundle ### |

O nouă variabilă de mediu, DATABASE\_URL fost adăugată automat și noul fișier de configurare doctrine.yaml face deja referință. Combinând variabilele de mediu și Flex, utilizați cele mai bune practici din industrie fără eforturi suplimentare.

## Continuă! [¶](https://translate.googleusercontent.com/translate_c?depth=1&rurl=translate.google.com&sl=auto&sp=nmt4&tl=ro&u=https://symfony.com.ua/doc/current/quick_tour/the_architecture.html&xid=17259,15700022,15700186,15700191,15700256,15700259,15700262,15700265,15700271,15700283&usg=ALkJrhjdpAcdKDR3BTLxCfjHO0cfWpir5g#id8)

Sună-mă nebun, dar după ce ai citit această parte, ar trebui să fii confortabil cu cele mai importante părți ale Symfony. Totul în Symfony este conceput pentru a nu vă împiedica să scrieți cod și să adăugați funcții, toate cu viteza și calitatea dorită.

Totul este pentru un tur rapid. De la autentificare, la forme, la memorie cache, sunt mult mai multe de explorat. Sunteți gata să vă scufundați în aceste subiecte? Apoi nu căutați, ci accesați documentația oficială [Symfony](https://translate.googleusercontent.com/translate_c?depth=1&rurl=translate.google.com&sl=auto&sp=nmt4&tl=ro&u=https://symfony.com.ua/doc/current/index.html&xid=17259,15700022,15700186,15700191,15700256,15700259,15700262,15700265,15700271,15700283&usg=ALkJrhgJmy2AjNat_BJJf41kIoyjbAF88w) și selectați orice director doriți.