Symfony lab3

**Добавление логирования**

composer require logger

Это устанавливает и конфигурирует (через рецепт) мощную библиотеку [Monolog](https://github.com/Seldaek/monolog). Чтобы использовать логгер в контроллере, добавьте новый аргумент, типизрованный LoggerInterface:

<?php

namespace App\Controller;

use Psr\Log\LoggerInterface;

use Symfony\Component\Routing\Annotation\Route;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;

class DefaultController extends AbstractController

{

/\*\*

\* @Route("/hello/{name}")

\*/

public function index($name, LoggerInterface $logger)

{

$logger->info("Saying hello to $name!");

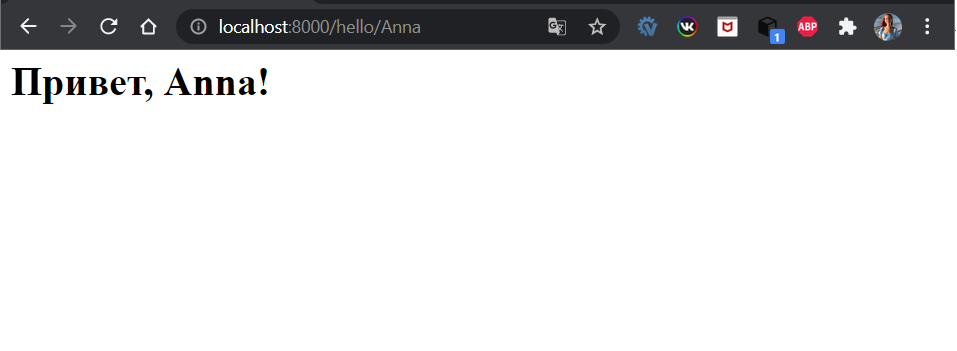
return $this->render('default/index.html.twig', [

'name' => $name,

]);

}

}

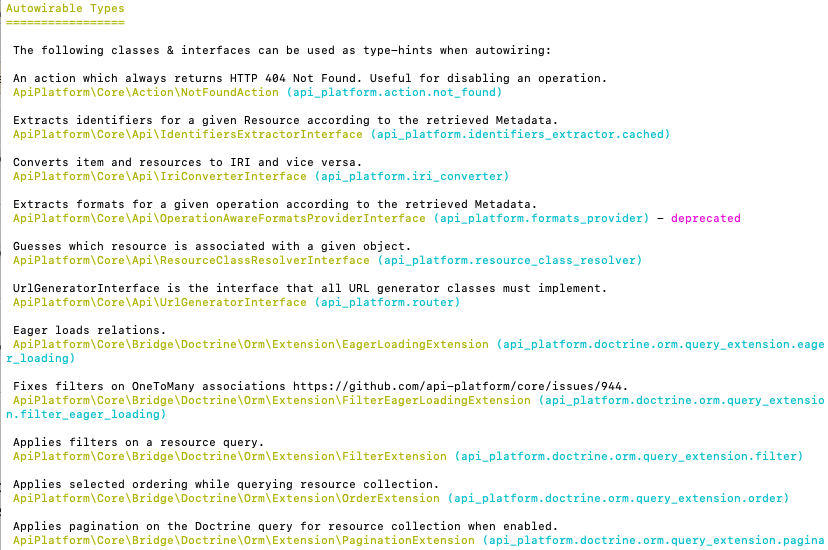


Сообщение в лог файле:

[2021-11-08T12:40:37.474547+00:00] app.INFO: Saying hello to Anna! [] []

**Сервисы и автомонтирование**

php bin/console debug:autowiring



**Создание сервисов**

Чтобы ваш код был упорядоченным, вы даже можете создать ваши собственные сервисы! Представьте, что вы хотите сгенерировать рандомное приветствие (например, "Привет", "Йо", и др.). Вместо того, чтобы помещать этот код напрямую в контроллер, создайте новый класс:

<?php

// src/GreetingGenerator.php

namespace App;

class GreetingGenerator

{

public function getRandomGreeting()

{

$greetings = ['Hey', 'Yo', 'Aloha'];

$greeting = $greetings[array\_rand($greetings)];

return $greeting;

}

}

<?php

namespace App\Controller;

use App\GreetingGenerator;

use Psr\Log\LoggerInterface;

use Symfony\Component\Routing\Annotation\Route;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;

class DefaultController extends AbstractController

{

/\*\*

\* @Route("/hello/{name}")

\*/

public function index($name, LoggerInterface $logger, GreetingGenerator $generator)

{

$greeting = $generator->getRandomGreeting();

$logger->info("Saying $greeting to $name!");

$logger->info("Saying hello to $name!");

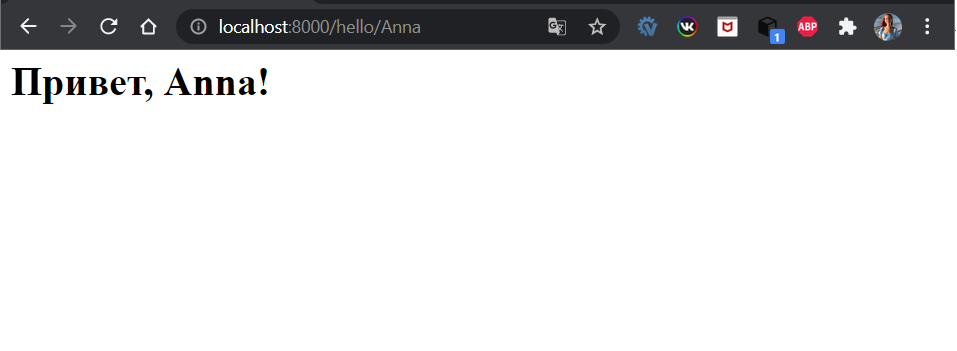
return $this->render('default/index.html.twig', [

'name' => $name,

]);

}

}

Сообщения в лог файле:

[2021-11-08T12:51:02.252852+00:00] app.INFO: Saying Hey to Anna! [] []

[2021-11-08T12:51:02.252960+00:00] app.INFO: Saying hello to Anna! [] []

Вот и всё! Symfony инстанциирует GreetingGenerator автоматически ипередаст его в качестве аргумента. Но, можем ли мы также переместить логику логгера в GreetingGenerator? Да! Вы можете использовать автомонтирование внутри сервиса, чтобы получить доступ к другимсервисам. Единственное отличие в том, что это делается в конструкторе:

<?php

// src/GreetingGenerator.php

namespace App;

use Psr\Log\LoggerInterface;

class GreetingGenerator

{

private $logger;

public function \_\_construct(LoggerInterface $logger)

{

$this->logger = $logger;

}

public function getRandomGreeting()

{

$greetings = ['Hey', 'Yo', 'Aloha'];

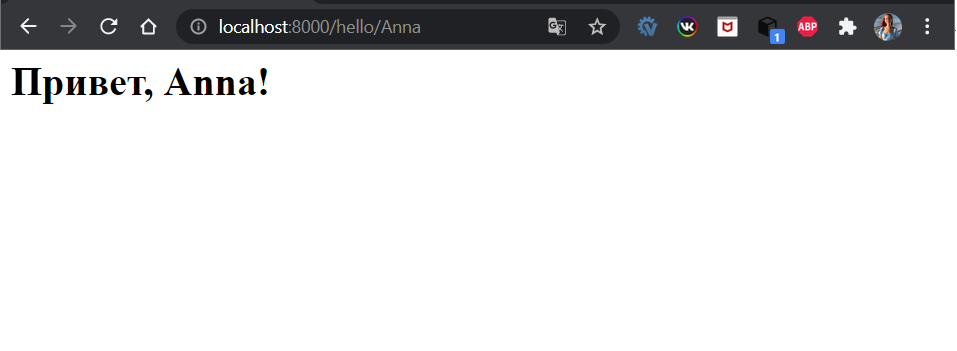
$greeting = $greetings[array\_rand($greetings)];

$this->logger->info('Using the greeting: '.$greeting);

return $greeting;

}

}



Сообщения в лог файле:

[2021-10-25T11:56:55.329076+00:00] app.INFO: Using the greeting: Yo [] []

[2021-10-25T11:56:55.329569+00:00] app.INFO: Saying Yo to Anna! [] []

[2021-10-25T11:56:55.329705+00:00] app.INFO: Saying hello to Anna! [] []

**Расширение и автоконфигурация Twig**

Благодаря обработке сервисов Symfony, вы можете расширять Symfony множеством способов, вроде создания подписчика событий или избирателя безопасности для сложных правил авторизации. Давайте добавим в Twig новый фильтр под названием greet. Как? Просто создайте класс, расширяющий AbstractExtension:

<?php

// 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!";

}

}

После создания всего *одного* файла, вы можете сразу же это использовать:

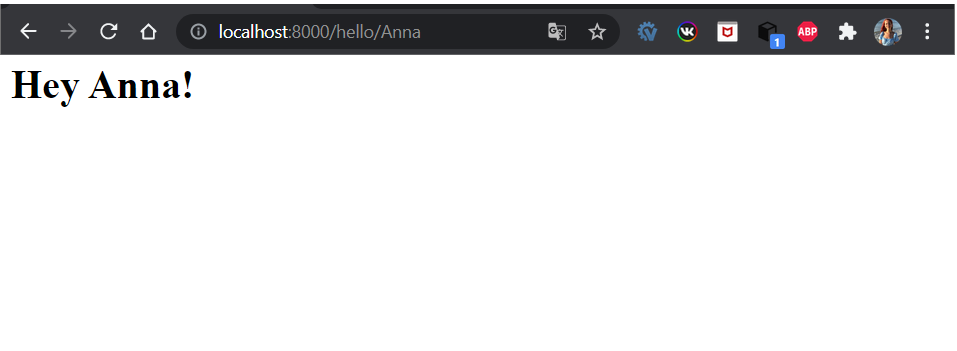
{# templates/default/index.html.twig #}

{% extends 'base.html.twig' %}

{% block body %}

<h1>{{ name | greet }}</h1>

{% endblock %}



## Разработка против производства: окружения

Это мощная идея: изменяя одну часть конфигурации (окружение), ваше приложение трансформируется из приятного опыта отладки в оптимизированное для скорости.

Ой, а как изменить окружение? Измените переменную окружения APP\_ENV с dev на prod:

#APP\_ENV=dev

APP\_ENV=prod

Но я хочу больше поговорить о переменных окружений дальше. Измените значение обратно на dev: инструменты отладки прекрасны, когда вы работаете локально.

## Переменные окружения

Каждое приложение содержит конфигурацию, которая отличается на каждом сервере - вроде информации о соединениях DB или паролей. Как их нужно хранить? В файлах? Или каким-то другим способом?

Symfony следует лучшей практике индустрии, храня конфигурацию, основанную на сервере, в виде переменных окружения. Это означает, что Symfony отлично работает с системами развёртывания Платформы, как Сервиса (PaaS), а также с Docker.

Но установка переменных окружения во время разработки может быть напряжной. Поэтому наше приложение автоматически загружает файл .env , если переменная окружения APP\_ENV не установлена в окружении. Ключи в этом файле потом становятся переменными окружения и считываются вашим приложением:

###> symfony/framework-bundle ###

APP\_ENV=dev

APP\_SECRET=6a58afcc365ed2b12372a747ab5395fe

###< symfony/framework-bundle ###

Вначале файл не содержит многого. Но с ростом вашего приложения, вы добавите больше конфигурации по мере необходимости. Но, на самом деле, становится намного интереснее! Представьте, что вашему приложение нужно DB ORM. Давайте установим Doctrine ORM:

composer require doctrine

###> doctrine/doctrine-bundle ###

# Format described at https://www.doctrine-project.org/projects/doctrine-dbal/en/latest/reference/configuration.html#connecting-using-a-url

# IMPORTANT: You MUST configure your server version, either here or in config/packages/doctrine.yaml

#

# DATABASE\_URL="sqlite:///%kernel.project\_dir%/var/data.db"

# DATABASE\_URL="mysql://db\_user:db\_password@127.0.0.1:3306/db\_name?serverVersion=5.7"

DATABASE\_URL="postgresql://db\_user:db\_password@127.0.0.1:5432/db\_name?serverVersion=13&charset=utf8"

###< doctrine/doctrine-bundle ###

Новая переменная окружения DATABASE\_URL была добавлена автоматически и на неё уже ссылается новый файл конфигурации doctrine.yaml. Объединив переменные окружения и Flex, вы используете лучшую практику индустрии без дополнительных усилий.