Молдавский Государственный Университет

Факультет Математики и Информатики

Специальность: Прикладная Информатика

Предмет: Framework. p/u Apl.WEB

Лабораторная работа №3(Symfony)

Выполнил: Грицюк Артем, IA1902

Руководитель: S.Bodrug

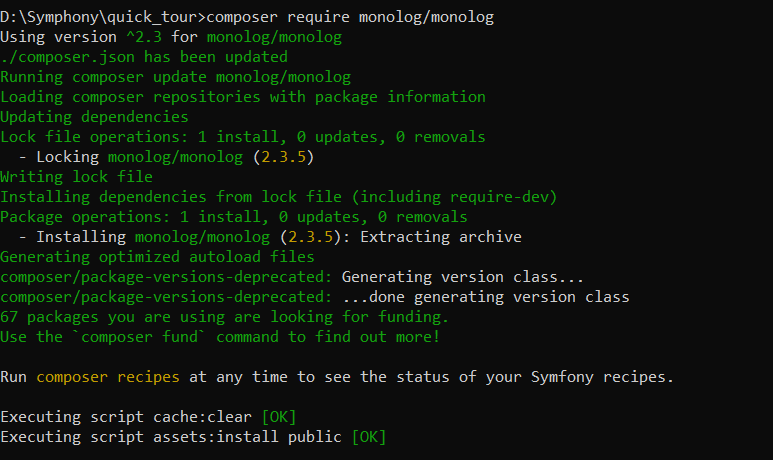
Кишинев, 2021

**Добавление логирования**

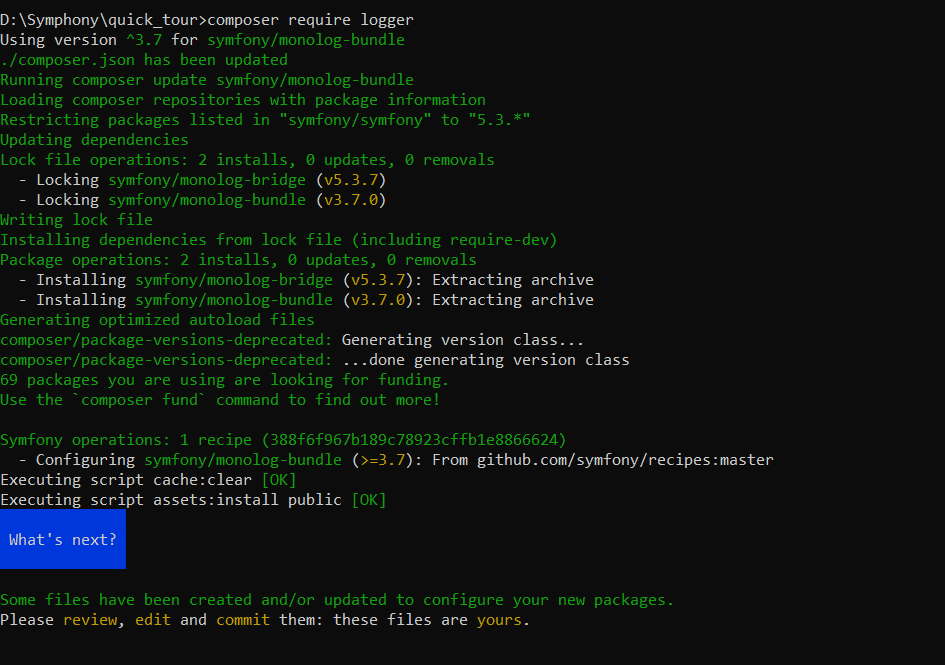
**Новое приложение Symfony микроскопическое: оно по сути состоит просто из системы маршрутизации и контроллера. Но благодаря Flex, установка новых функций проста.**

**Хотите систему логирования? Не проблема:**

**Composer require monolog/monolog**



**Composer require logger**



**Это устанавливает и конфигурирует (через рецепт) мощную библиотеку Monolog. Чтобы использовать логгер в контроллере, добавьте новый аргумент, типизрованный LoggerInterface:**

<?php

// src/Controller/DefaultController.php

namespace App\Controller;

use Psr\Log\LoggerInterface;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;

use Symfony\Component\Routing\Annotation\Route;

class DefaultController extends AbstractController

{

/\*\*

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

\*/

public function index($name, LoggerInterface $logger)

{

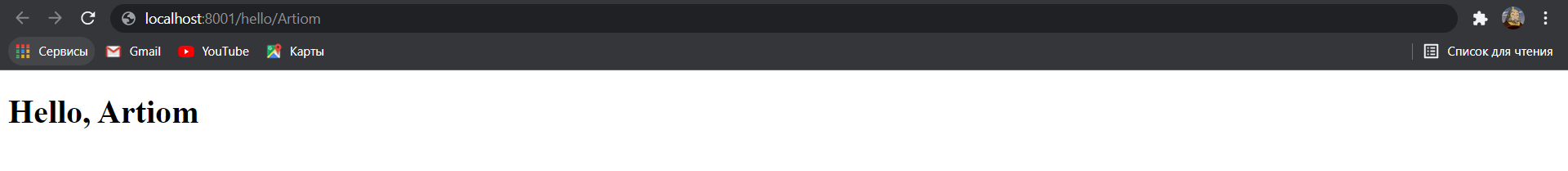
$logger->info("Saying hello to $name!");

return $this->render('default/index.html.twig', ['name' => $name,]);

}

}

?>

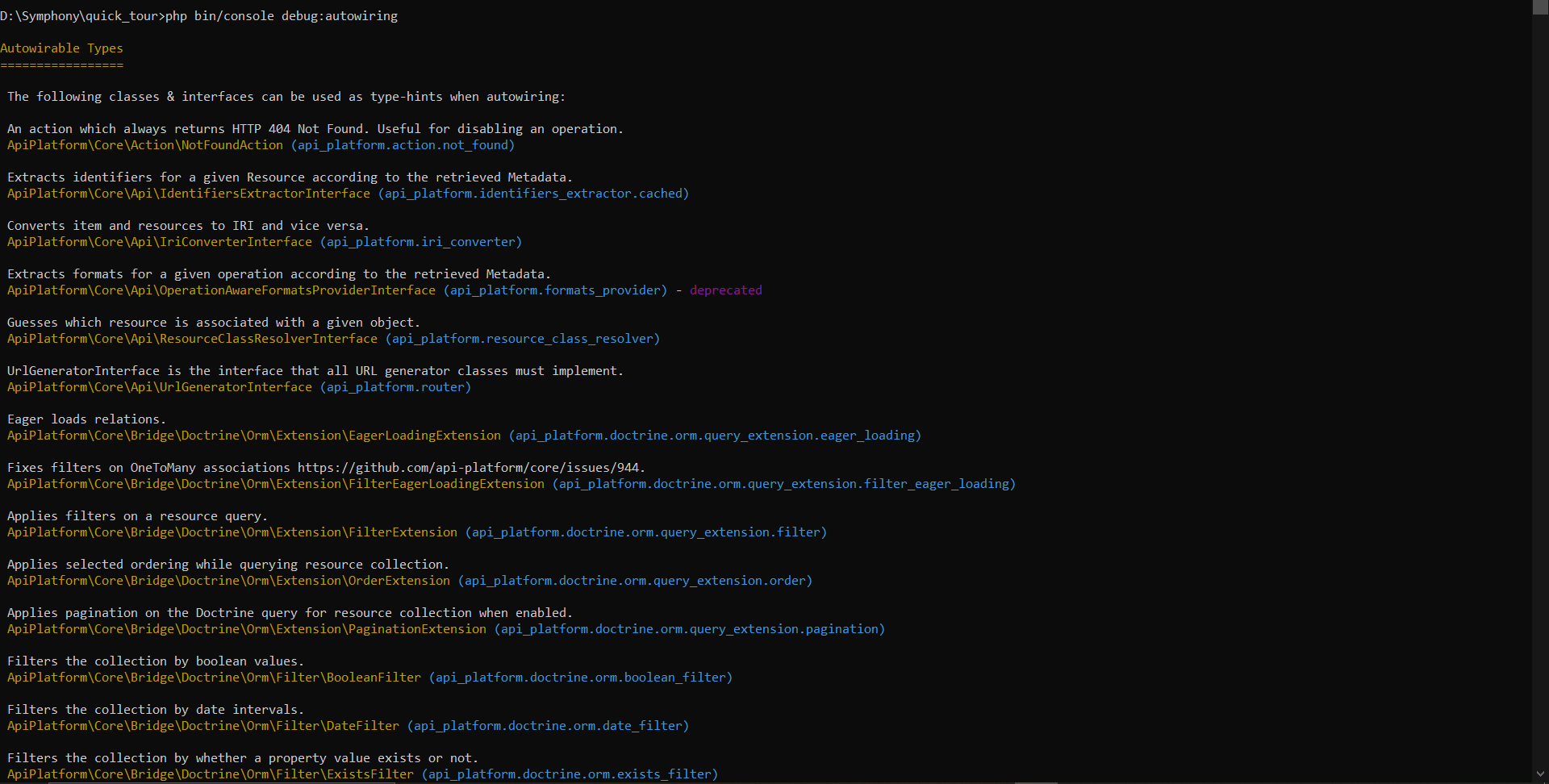


**Новое сообщение лога будет написано в var/log/dev.log. Конечно же, это можно сконфигурировать, обновив один из файлов конфигурации, добавленный рецептом.**



**Сервисы и автомонтирование**

**Чтобы сделать жизнь потрясающей, вы можете попросить Symfony передать вам сервис, используя типизирование. Какие другие возможные классы или интерфейсы вы можете использовать? Узнайте, выполнив:**



**Создание сервисов**

**Чтобы ваш код был упорядоченым, 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;

}

}

?>

**Отлично! Вы можете использовать это в вашем контроллере незамедлительно:**

**<?php**

// src/Controller/DefaultController.php

namespace App\Controller;

use App\GreetingGenerator;

use Psr\Log\LoggerInterface;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;

use Symfony\Component\HttpFoundation\Response;

use Symfony\Component\Routing\Annotation\Route;

class DefaultController extends AbstractController

{

/\*\*

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

\*/

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

{

$greeting = $generator->getRandomGreeting();

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

return $this->render('default/index.html.twig',['name' => $name,]);

// ...

}

}

?>

**Вот и всё! Symfony инстанциирует GreetingGenerator автоматически и передаст 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 другим сервисам. Единственное отличие в том, что это делается в конструкторе:**



**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!";

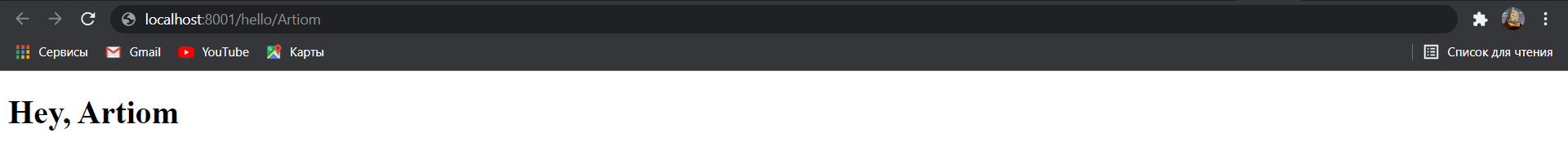
}

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

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

{# Отобразит что-то вроде "Привет, Symfony!" #}

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

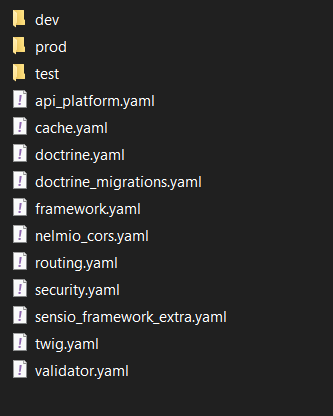


**Разработка против производства: окружения**

**Одной из главных задач фреймворка является облегчение отладки! И наше приложение полно отличных инструментов для этого: панель инструментов веб-отладки отображает внизу страницы, ошибки большие, прекрасные и полные, и любая конфигурация кеша автоматически строится заново при необходимости.**

**Но что если вы развёртываете в производстве? Нам нужно будет спрятать эти инструменты и оптимизироваться для скорости!**

**Это решается системой окружений Symfony и их есть три: dev, prod и test. В зависимости от окружения, Symfony загружает разные файлы в каталог config/:**



**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:**

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

APP\_ENV=dev

APP\_SECRET=de45623af946c0f61ea7da58e480b872

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

**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 ###

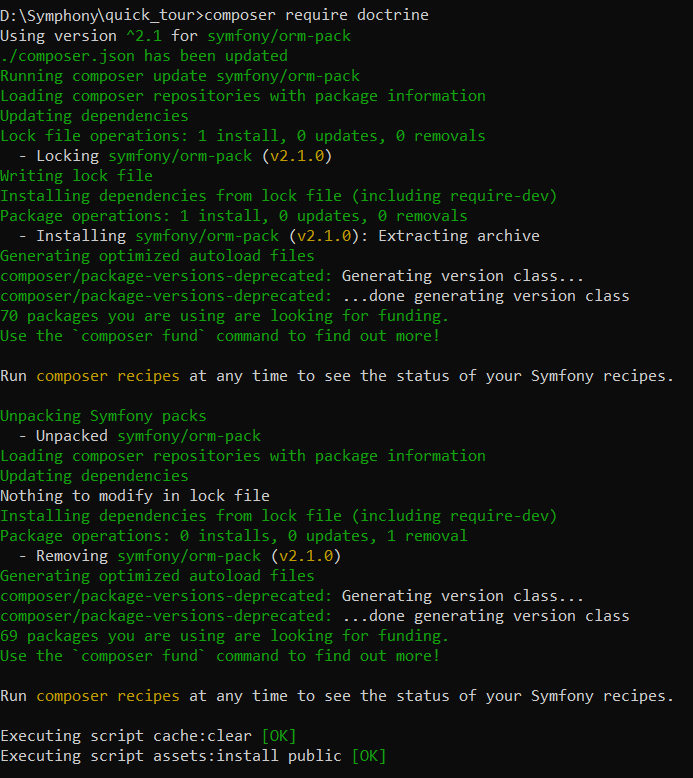
APP\_ENV=dev

APP\_SECRET=de45623af946c0f61ea7da58e480b872

###< 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:**

**Прописываем composer require doctrine**



**Благодаря новому рецепту, установленному Flex, посмотрите на файл .env ещё раз:**

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

APP\_ENV=dev

APP\_SECRET=de45623af946c0f61ea7da58e480b872

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

###> 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://symfony:ChangeMe@127.0.0.1:5432/app?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.**