Выполнил: Даалы Анатолий

Composer require twig

Composer require annotations

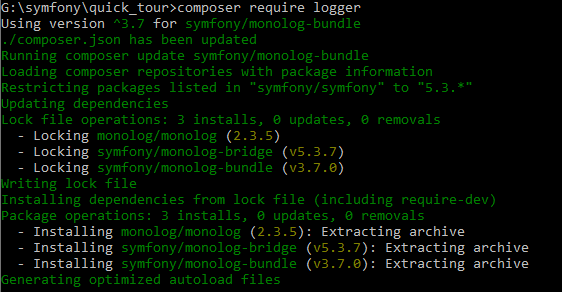
Composer require api

Добавление логирования[¶](https://symfony.com.ua/doc/current/quick_tour/the_architecture.html#id2)

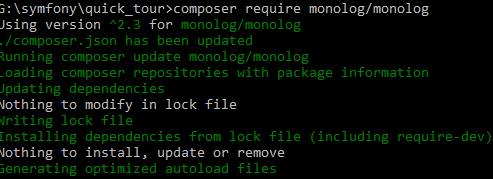
Новое приложение Symfony микроскопическое: оно по сути состоит просто из системы маршрутизации и контроллера. Но благодаря Flex, установка новых функций проста.

Хотите систему логирования? Не проблема:

Composer require logger



Composer require monolog/monolog



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



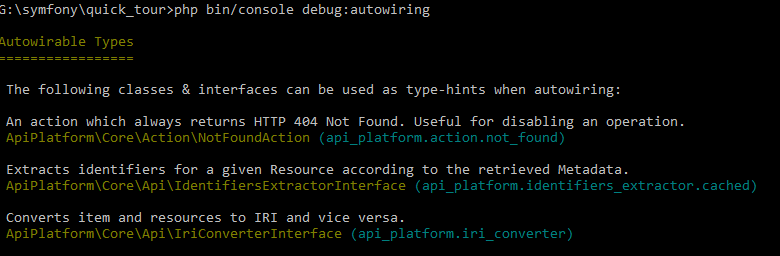
Новое сообщение лога будет написано в **var/log/dev.log**. Конечно же, это можно сконфигурировать, обновив один из файлов конфигурации, добавленный рецептом.



Сервисы и автомонтирование[¶](https://symfony.com.ua/doc/current/quick_tour/the_architecture.html" \l "id3" \o "Permalink to this headline)

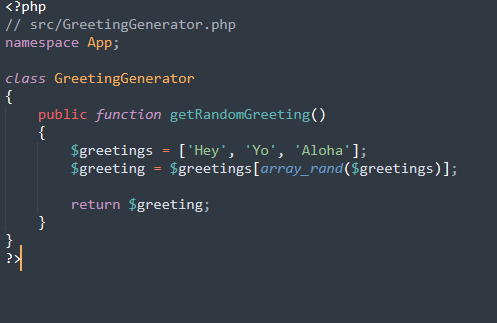
Чтобы сделать жизнь потрясающей, вы можете попросить Symfony передать вам сервис, используя типизирование. Какие другие возможные классы или интерфейсы вы можете использовать? Узнайте, выполнив:

php bin/console debug:autowiring



Создание сервисов[¶](https://symfony.com.ua/doc/current/quick_tour/the_architecture.html#id4)

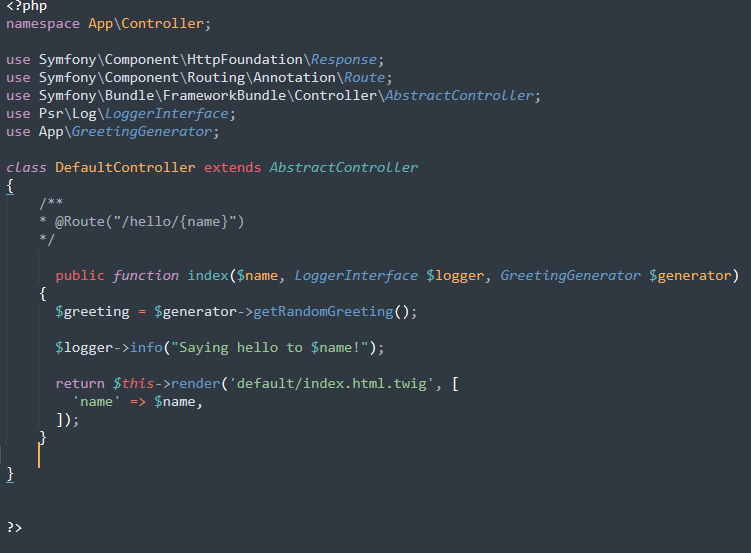
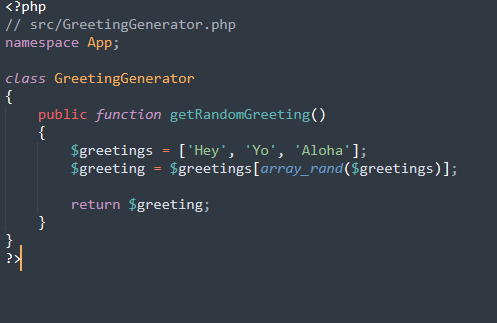
Чтобы ваш код был упорядоченым, вы даже можете создать ваши собственные сервисы! Представьте, что вы хотите сгенерировать рандомное приветствие (например, "Привет", "Йо", и др.). Вместо того, чтобы помещать этот код напрямую в контроллер, создайте новый класс:



Отлично! Вы можете использовать это в вашем контроллере незамедлительно:



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:

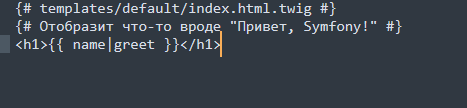


Расширение и автоконфигурация Twig

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:



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

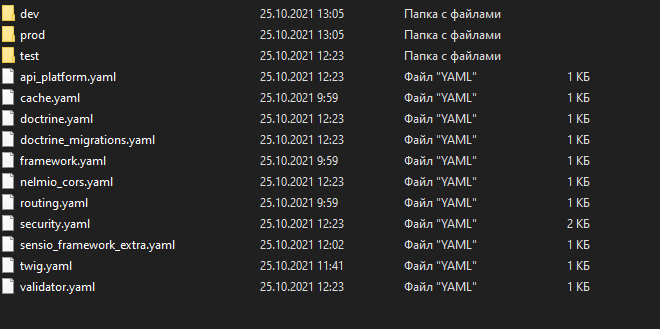


Разработка против производства: окружения

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.

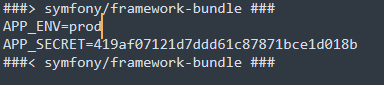
Но что если вы развёртываете в производстве? Нам нужно будет спрятать эти инструменты и оптимизироваться для скорости!

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:

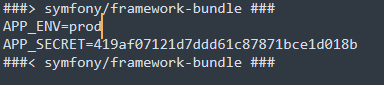


Переменные окружения[¶](https://symfony.com.ua/doc/current/quick_tour/the_architecture.html#id7)

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?

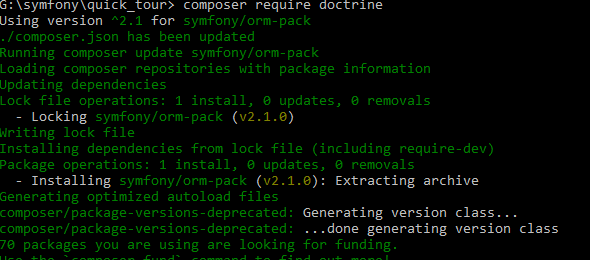
Symfony следует лучшей практике индустрии, храня конфигурацию, основанную на сервере, в виде переменных окружения. Это означает, что Symfony отлично работает с системами развёртывания Платформы, как Сервиса (PaaS), а также с Docker.

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:

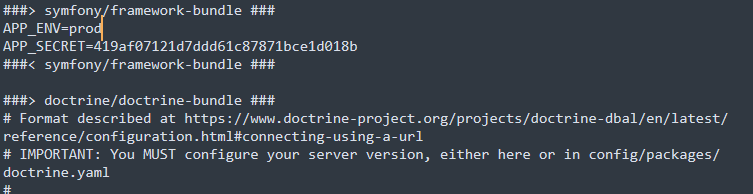


Вначале файл не содержит многого. Но с ростом вашего приложения, вы добавите больше конфигурации по мере необходимости. Но, на самом деле, становится намного интереснее! Представьте, что вашему приложение нужно БД ORM. Давайте установим Doctrine ORM

composer require doctrine



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



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.