Symfony 5. Быстрый старт

Fabien Potencier

https://fabien.potencier.org/

@fabpot

@fabpot

Symfony 5. Быстрый старт

ISBN-13: 978-2-918390-42-8

current — Generated on February 13, 2020

Symfony SAS
92-98, boulevard Victor Hugo
92 110 Clichy
France

If you find typos or errors, feel free to report them at support@symfony.com. This book is continuously updated based on user feedback.

Оглавление

Содержание

Благодарности

Я люблю книги. Книги, которые я могу держать в руках.

Последний раз я писал книгу про Symfony ровно 10 лет назад. Тогда в ней рассматривался Symfony 1.4. С тех пор я никогда не писал о Symfony!

Я был так рад снова написать о Symfony, что закончил первый черновик за неделю. На версию, которую вы сейчас читаете, ушло намного больше времени. Написание книги занимает много времени и сил. От дизайна обложки до макета страницы. От поправок в коде до рецензий коллег. Это может продолжаться практически бесконечно. Вы всегда можете дополнить раздел, улучшить определённый кусок кода, исправить опечатки или переписать объяснение, чтобы сделать его короче и лучше.

Написание книги — это путешествие, которое не стоит совершать в одиночку. Огромное число людей прямо или косвенно помогли в этом нелёгком деле. Спасибо каждому из них!

Я искренне хочу поблагодарить всех прекрасных людей, которые потратили много времени на рецензию текста, чтобы найти опечатки и улучшить содержание; некоторые даже помогли мне написать фрагменты кода:

Javier Eguiluz

Nicolas Grekas

Ryan Weaver

Kévin Dunglas

Titouan Galopin

Tugdual Saunier

Grégoire Pineau

Alexandre Salomé

Переводчики

Официальная документация Symfony доступна только на английском языке. В прошлом у нас были переводы документации, но мы решили отказаться от них, так как они всегда отставали от оригинала. А устаревшая документация — это ещё хуже, чем её отсутствие.

Главная проблема с переводами — это их поддержка. Документация Symfony обновляется каждый день десятками разработчиков. Практически невозможно иметь команду волонтёров, которая бы могла переводить все изменения в режиме реального времени.

Однако задача по переводу книги, которую вы сейчас читаете, более выполнима, так как я пытался написать о том, что не будет сильно меняться со временем. Поэтому содержание книги должно быть достаточно актуальным в течение долгого времени.

Но зачем нам вообще в техническом мире, где английский язык де-факто основной язык, документация на других языках? Symfony используется разработчиками по всему миру. Некоторым из них не так комфортно читать на английском. Перевод раздела документации Getting Started (для тех, кто только начинают работу с фреймворком) — это часть программы Symfony по этническому разнообразию (Symfony Diversity), в которой мы стремимся найти способы сделать Symfony как можно доступнее для всех.

Как вы можете себе представить, перевод более 300 страниц — это огромная работа, и я хочу поблагодарить всех, кто помог перевести эту книгу:

Sergey Panteleev	Stan Drozdov
Dmitry Naydonov	Egor Ushakov
Alexey Pyltsyn	Sergey Kamardin
VladimirAus	Andrey Bocharov

Спонсоры среди компаний

Эту книгу *финансово поддержали* люди со всего мира. Благодаря им содержимое книги доступно бесплатно в интернете, а также в бумажном варианте, распространяемом на конференциях Symfony.

https://packagist.com/

https://darkmira.io/

https://basecom.de/

https://dats.team/

https://sensiolabs.com/

https://les-tilleuls.coop/

https://redant.nl/

https://www.akeneo.com/

https://www.facile.it/

https://izi-by-edf.fr/

https://www.musement.com/

https://setono.com/

Спонсоры среди частных лиц

Javier Eguiluz	@javiereguiluz
Tugdual Saunier	@tucksaun
Alexandre Salomé	https://alexandre.salome.fr
Timo Bakx	@TimoBakx
Arkadius Stefanski	https://ar.kadi.us
Oskar Stark	@OskarStark
slaubi	
Jérémy Romey	@jeremyFreeAgent
Nicolas Scolari	
Guys & Gals at SymfonyCasts	https://symfonycasts.com
Roberto santana	@robertosanval
Ismael Ambrosi	@iambrosi
Mathias STRASSER	https://roukmoute.github.io/
Platform.sh team	http://www.platform.sh
ongoing	https://www.ongoing.ch
Magnus Nordlander	@magnusnordlander
Nicolas Séverin	@nico-incubiq
Centarro	https://www.centarro.io
Lior Chamla	https://learn.web-develop.me
Art Hundiak	@ahundiak
Manuel de Ruiter	https://www.optiwise.nl/
Vincent Huck	
Jérôme Nadaud	https://nadaud.io
Michael Piecko	@mpiecko
Tobias Schilling	https://tschilling.dev
ACSEO	https://www.acseo.fr
Omines Internetbureau	https://www.omines.nl/
Seamus Byrne	http://seamusbyrne.com
Pavel Dubinin	@geekdevs
Jean-Jacques PERUZZI	https://linkedin.com/in/jjperuzzi
Alexandre Jardin	@ajardin

Christian Ducrot	http://ducrot.de
Alexandre HUON	@Aleksanthaar
François Pluchino	@francoispluchino
We Are Builders	https://we.are.builders
Rector	@rectorphp
Ilyas Salikhov	@salikhov
Romaric Drigon	@romaricdrigon
Lukáš Moravec	@morki
Malik Meyer-Heder	@mehlichmeyer
Amrouche Hamza	@cDaed
Russell Flynn	https://custard.no
Shrihari Pandit	@shriharipandit
Salma NK.	@os_rescue
Nicolas Grekas	
Roman Ihoshyn	https://ihoshyn.com
Radu Topala	https://www.trisoft.ro
Andrey Reinwald	https://www.facebook.com/andreinwald
JoliCode	@JoliCode
Rokas Mikalkėnas	
Zeljko Mitic	@strictify
Wojciech Kania	@wkania
Andrea Cristaudo	https://andrea.cristaudo.eu/
Adrien BRAULT-LESAGE	@AdrienBrault
Cristoforo Stevio Cervino	http://www.steviostudio.it
Michele Sangalli	
Florian Reiner	http://florianreiner.com
Ion Bazan	@IonBazan
Marisa Clardy	@MarisaCodes
Donatas Lomsargis	http://donatas.dev
Johnny Lattouf	@johnnylattouf
Duilio Palacios	https://styde.net
Pierre Grimaud	@pgrimaud

Marcos Labad Díaz	@esmiz
Stephan Huber	https://www.factorial.io
Loïc Vernet	https://www.strangebuzz.com
Daniel Knoch	http://www.cariba.de
Emagma	http://www.emagma.fr
Gilles Doge	
Malte Wunsch	@MalteWunsch
Jose Maria Valera Reales	@Chemaclass
Cleverway	https://cleverway.eu/
Nathan	@nutama
Abdellah EL GHAILANI	https://connect.symfony.com/profile/aelghailani
Solucionex	https://www.solucionex.com
Elnéris Dang	https://linkedin.com/in/elneris-dang/
Class Central	https://www.classcentral.com/
Ike Borup	https://idaho.dev/
Christoph Lühr	https://www.christoph-luehr.com/
Zig Websoftware	http://www.zig.nl
Dénes Fakan	@DenesFakan
Danny van Kooten	http://dvk.co
Denis Azarov	http://azarov.de
Martin Poirier T.	https://linkedin.com/in/mpoiriert/
Dmytro Feshchenko	@dmytrof
Carl Casbolt	https://www.platinumtechsolutions.co.uk/
Irontec	https://www.irontec.com
Lukas Plümper	https://lukaspluemper.de/
Neil Nand	https://neilnand.co.uk
Andreas Möller	https://localheinz.com
Alexey Buldyk	https://buldyk.pw
Page Carbajal	https://pagecarbajal.com
Florian Voit	https://rootsh3ll.de
Webmozarts GmbH	https://webmozarts.com
Alexander M. Turek	@derrabus

Zan Baldwin	@ZanBaldwin
Ben Marks, Magento	http://bhmarks.com

Любовь семьи

Поддержка семьи — превыше всего. Большое спасибо моей жене **Элен** и двум моим замечательным детям **Томасу** и **Лукасу** за их постоянную поддержку.

Наслаждайтесь иллюстрацией Томаса… и книгой!

Шаг 0
О чём эта книга?

Symfony — один из наиболее успешных PHP-проектов. Это не только мощный фулстек-фреймворк, но и популярный набор переиспользуемых компонентов.

С выходом пятой версии проект, вероятно, достиг зрелости. Думаю, что всё, что мы сделали за последние пять лет, прекрасно сочетается между собой. Это и новые низкоуровневые компоненты, и высококачественные интеграции с другими программами, а также инструменты, повышающие продуктивность разработчика. Мы существенно улучшили удобство разработки без потери гибкости. Ещё никогда использование Symfony в новом проекте не было таким увлекательным.

Если вы только начинаете работать с Symfony, то выход **Symfony 5** — это самое время, чтобы научиться разрабатывать приложение шаг за шагом. Данная книга показывает разработчикам мощь фреймворка и как они с помощью него могут улучшить свою производительность.

Если вы уже разработчик на Symfony, то после прочтения книги вы заново откроете для себя этот фреймворк. За последние несколько лет фреймворк существенно развился и впечатление от процесса разработки значительно улучшилось. У меня такое ощущение, что многие Symfony-разработчики до сих цепляются за старые привычки,

из-за чего им трудно освоить новые способы разработки приложений с помощью Symfony. Я могу их понять. Темпы эволюции просто ошеломляют. Занятые полный рабочий день своими проектами, разработчики просто не успевают следить за всем происходящим в сообществе. Я это знаю из собственного опыта и поэтому не буду делать вид, что могу уследить за всем. Это далеко не так.

И я имею в виду не только новые способы работы. Речь идёт также о новых компонентах: клиент HTTP, Mailer, Workflow, Messenger. Они в корне меняют подходы к разработке. Вероятнее всего, перечисленные компоненты изменят ваше представление о приложениях на Symfony.

Помимо этого, я чувствую необходимость в новой книге, поскольку интернет претерпел значительные изменения. Сейчас нам нужно обсуждать *API-интерфейсы*, *SPA-приложения*, *контейнеризацию*, *непрерывное развёртывание* и т.п.

Время бесценно. Не ждите ни длинных абзацев, ни долгих пояснений основных понятий. Эта книга больше напоминает путешествие. С чего начать, когда и как писать код. Я постараюсь пробудить интерес к важным темам и дам вам самим решать, хотите ли вы подробнее в них разобраться.

Я также не хочу повторять то, что написано в документации. Она очень качественная, поэтому я буду часто ссылаться на неё в разделе «Двигаемся дальше» в конце каждого шага/главы. Рассматривайте эту книгу как список указателей на дополнительные материалы.

В книге описывается создание приложения с нуля, вплоть до развёртывания в продакшене. Тем не менее, мы не будем разрабатывать всё до полной готовности. Поэтому не стоит ожидать идеального результата. Мы пойдем по короткому пути, и не будем затрагивать особые случаи обработки, проверки или тестирования чего-либо. Мы не всегда будем следовать рекомендуемым практикам, но затронем практически все аспекты современного проекта на Symfony.

Перед тем, как приступить к работе над книгой, я сначала разработал итоговое приложение. Я был впечатлён результатом и скоростью, которую сумел сохранить, легко добавляя новую функциональность. И всё это благодаря хорошей документации и тому, что Symfony 5 знает, что вам нужно на вашем пути. Уверен, что Symfony ещё есть куда улучшать (я написал несколько заметок о возможных улучшениях на этот счёт), хотя если сравнивать с несколькими годами ранее,

разрабатывать на нём стало намного приятнее. Я хочу рассказать всем об этом.

Книга разделена на шаги. Каждый шаг в свою очередь состоит из ещё более мелких шагов, так что всё это должно читаться быстро. Но гораздо важнее, к чему я вас призываю — начните писать код сразу, по ходу чтения книги. Напишите код, протестируйте его, посмотрите в действии, и затем попробуйте его улучшить.

И последнее, но не менее важное: не стесняйтесь обращаться за помощью, если окажетесь в тупике. Вы можете попасть в непредвиденную ситуацию или столкнуться с опечаткой в вашем коде, которую возможно будет трудно найти и исправить. Задайте вопросы, у нас есть замечательное сообщество в *Slack* и *Stack Overflow*.

Готовы разрабатывать? Тогда наслаждайтесь!

Шаг 1
Проверка рабочего окружения

Прежде чем приступить к разработке проекта, нужно проверить, что у вас корректно настроено рабочее окружение. Это очень важно. Имеющиеся на сегодня инструменты разработки сильно отличаются от тех, которые были 10 лет назад. Они существенно изменились в лучшую сторону. Было бы глупо не использовать их: хорошие инструменты помогут добиться больших успехов.

Пожалуйста, не пропускайте этот шаг. Или, по крайней мере, прочитайте последний раздел про Symfony CLI.

1.1 Компьютер

Вам понадобится компьютер. К счастью, он может быть на любой популярной операционной системе: macOS, Windows или Linux. Symfony и все инструменты, которые мы будем использовать, совместимы с любой из них.

1.2 Используемые технологии

Будем разрабатывать быстро, используя лучшие инструменты на сегодняшний момент. Для этого я выбрал наиболее подходящие, на мой взгляд, технологии.

PostgreSQL для сервера базы данных.

RabbitMQ — фаворит для работы с очередями.

1.3 IDE

Если хотите, вы можете использовать Notepad. Хотя я бы не рекомендовал этого делать.

Раньше я работал в Textmate. Теперь же я пользуюсь «настоящей» IDE. Благодаря таким функциям IDE, как автодополнение, автоматическое добавление и сортировка операторов `use`, быстрое переключение с одного файла на другой, ваша производительность увеличится в разы.

Я бы рекомендовал использовать *Visual Studio Code* или *PhpStorm*. Первая программа бесплатна, вторая — нет, но лучше работает с Symfony (благодаря плагину *Symfony Support*). Решение за вами. Я знаю, что вы хотели бы узнать, какую из этих IDE я использую. Эту книгу я пишу в Visual Studio Code.

1.4 Терминал

Мы будем постоянно переключаться между IDE и командной строкой. Вы можете использовать встроенный терминал в IDE, хотя лично мне больше нравится использовать внешний терминал из-за того, что можно изменить размер окна.

В Linux есть встроенный терминал — программа `Terminal`. В macOS рекомендую вместо штатного терминала использовать *iTerm2*, а в Windows хорошо зарекомендовал себя *Hyper*.

1.5 Git

В своей последней книге я рекомендовал Subversion в качестве системы управления версиями. Сейчас времена изменились — все, как правило, используют *Git*.

В Windows установите *Git Bash*.

Убедитесь, что вы знаете, как выполнять основные команды, такие как `git clone`, `git log`, `git show`, `git diff`, `git checkout` и т. п.

1.6 PHP

Мы будем использовать Docker для разнообразных сервисов приложения, хотя для простоты, производительности и стабильности на моём компьютере установлен PHP. Можете считать меня консерватором, но сочетание локально установленного PHP и сервисов Docker — это идеальный для меня вариант.

По возможности используйте PHP 7.3 или 7.4, в зависимости от того, когда вы читаете эту книгу. Убедитесь, что следующие модули PHP установлены или будут установлены: `intl`, `pdo_pgsql`, `xsl`, `gd`, `openssl`, `sodium`. Можете также установить `redis` и `curl`.

С помощью команды `php -m` можно посмотреть включённые в данный момент модули.

Нам также понадобится `php-fpm`, если ваша платформа его поддерживает, хотя `php-cgi` тоже подойдёт.

1.7 Composer

Управление зависимостями в проекте на Symfony очень важно. Установите последнюю версию инструмента для управления зависимостями в PHP — *Composer*.

Если вы не работали с Composer, то рекомендую изучить его команды.

Можно использовать сокращённые имена команд: `composer req` вместо `composer require`, `composer rem` вместо `composer remove` и т. д.

1.8 Docker и Docker Compose

Docker и Docker Compose будут управлять сервисами приложения. *Установите их* и запустите Docker. Если вы впервые пользуетесь Docker, ознакомьтесь с этим инструментом. Не переживайте: сценарии использования Docker будут достаточно простыми, без сложных конфигураций и настроек.

1.9 Symfony CLI

И последнее, но не менее важное: для ускорения разработки мы будем использовать инструмент командной строки `symfony`. Начиная с того, что он предоставляет функции локального веб-сервера, и заканчивая его полной интеграцией с Docker и поддержкой SymfonyCloud, этот инструмент сэкономит нам немало времени.

Установите *Symfony CLI* и не забудьте переместить эту утилиту в одну из директорий, заданных в переменной окружения `$PATH`. Создайте учётную запись *SymfonyConnect* если у вас её ещё нет, и авторизуйтесь с помощью команды `symfony login`.

Для использования HTTPS на локальном сервере нужно ещё добавить поддержку TLS, *установив центр сертификации (certificate authority, CA)* с помощью следующей команды:

```
$ symfony server:ca:install
```

Убедитесь, что на вашем компьютере всё готово для работы, выполнив следующую команду:

```
$ symfony book:check-requirements
```

Если вы хотите немного поэкспериментировать, запустите *прокси-сервер Symfony*. Хотя это необязательно, наш проект будет доступен по локальному доменному имени, оканчивающемуся на `.wip`.

При выполнении команды в терминале мы почти всегда будем использовать префикс `symfony`. Например, `symfony composer`, а не просто `composer`, либо `symfony console` вместо `./bin/console`.

Основная причина — Symfony CLI автоматически создаёт некоторые

переменные окружения, исходя из сервисов, запущенных на вашем компьютере с помощью Docker. Эти переменные окружения доступны для HTTP-запросов, поскольку локальный веб-сервер добавляет их автоматически. Таким образом, использование префикса `symfony` в CLI даст одинаковый результат на разных платформах.

Кроме того, Symfony CLI автоматически выбирает «самую подходящую» версию PHP из доступных на вашей локальной машине.

Шаг 2
Знакомство с проектом

Нам нужно придумать проект, над которым мы будем работать. Это довольно непростая задача. С одной стороны, проект должен быть достаточно большим, чтобы показать все возможности Symfony. С другой стороны, нужен небольшой проект, чтобы нам не наскучило реализовывать похожую функциональность по несколько раз.

2.1 Определение цели проекта

Было бы неплохо, если бы проект каким-то образом был связан с Symfony и конференциями, так как книга должна быть опубликована на конференции SymfonyCon 2019 в Амстердаме. Как насчёт *гостевой книги* или ливр д'ор, как мы называем её по-французски? Мне нравится это старое доброе ощущение разработки гостевой книги в 2019 году!

Наш проект предназначен для сбора отзывов о конференциях. На главной странице разместим список конференций. На странице

каждой конференции выведем полезные комментарии. Комментарий будет состоять из небольшого текста и фотографии (по желанию). Кажется, я расписал все требования, чтобы начать разработку.

В *проект* будет входить несколько *приложений*. Классическое веб-приложение с использованием HTML, API и SPA-приложение для мобильных устройств.

2.2 Обучение на практике

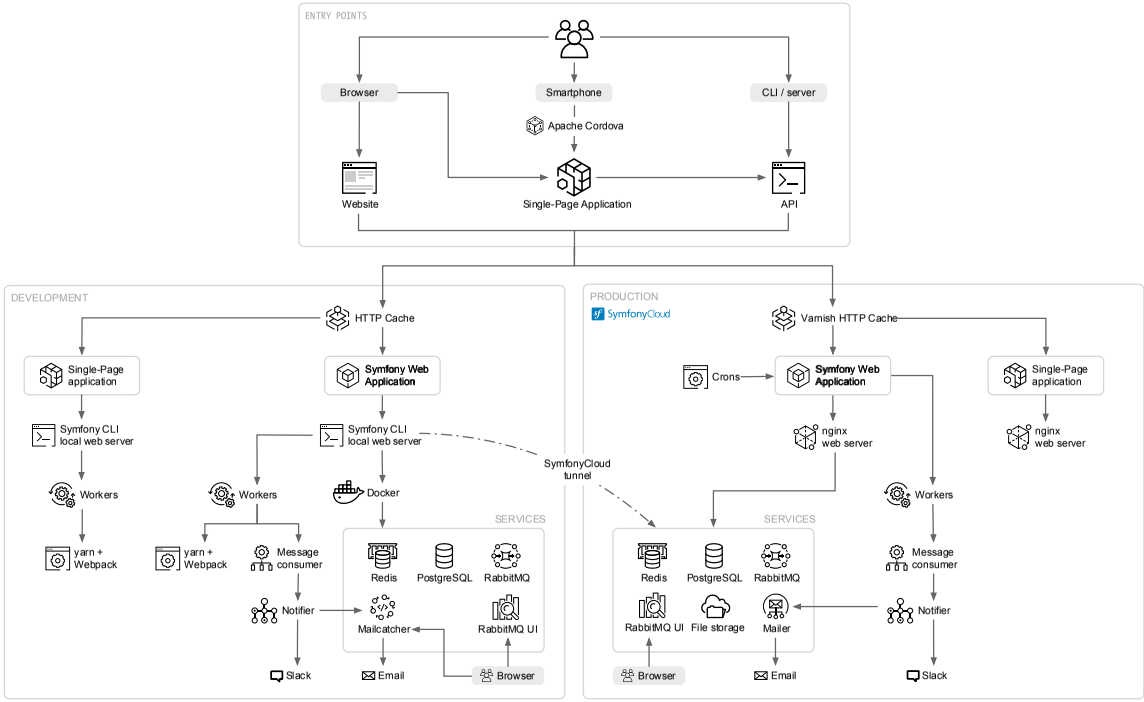
Мы учимся, когда мы что-то делаем. Только так, и никак иначе. Конечно, читать книгу о Symfony — это хорошо, но разрабатывать приложение на своём компьютере по ходу чтения книги — гораздо лучше. Эта книга — нечто особенное, потому что было сделано всё возможное, чтобы вы могли повторять то, что в ней описано: писать код и достигать тех же результатов, что и я, когда разрабатывал приложение у себя.

В книге содержится весь код, который вам предстоит написать, а также все команды для выполнения — одним словом всё, чтобы получить готовое приложение. Все строки кода на своих местах. Все нужные команды прописаны. Это всё уместилось в этой книге благодаря тому, что в современных приложениях на Symfony очень мало шаблонного кода. В основном мы будем писать код, связанный с *бизнес-логикой* проекта. Всё остальное по большей части уже сделано за нас, либо автоматически генерируется.

2.3 Итоговая диаграмма структуры проекта

Несмотря на то, что наш проект кажется простым, мы не станем создавать приложение уровня «Hello World». Это значит, что мы не ограничимся только использованием PHP и взаимодействием с базой данных.

Речь идёт о создании проекта, с которым вы столкнетесь в реальности со всеми вытекающими сложностями. Не верите? Посмотрите на окончательный вариант инфраструктуры проекта:


ENTRY POINTS
Browser
Smartphone
CLI / server
Apache Cordova
Website
Single-Page Application
API
DEVELOPMENT
HTTP Cache
Single-Page application
Symfony Web Application
Symfony CLI local web server
Symfony CLI local web server
Workers
Workers
Docker
yarn + Webpack
yarn + Webpack
Message consumer
SERVICES
Redis
PostgreSQL
RabbitMQ
Notifier
Mailcatcher
RabbitMQ UI
Slack
Email
Browser
SymfonyCloud tunnel
PRODUCTION
SymfonyCloud
Varnish HTTP Cache
Crons
Symfony Web Application
Single-Page application
nginx web server
nginx web server
Workers
Message consumer
SERVICES
Redis
PostgreSQL
RabbitMQ
RabbitMQ UI
File storage
Mailer
Notifier
Browser
Email
Slack

Одно из главных преимуществ использования фреймворка — небольшое количество кода, необходимого для разработки подобного проекта:

- 20 PHP-классов в директории `src/` для работы сайта;
- 550 логических строк PHP-кода (LLOC) по данным *PHPLOC*;
- 40 строк конфигурационных данных в 3 файлах (с использованием аннотаций и YAML) преимущественно для настройки архитектуры бэкенд-части;
- 20 строк конфигурации инфраструктуры для разработки (Docker);
- 100 строк конфигурации для продакшен-инфраструктуры (SymfonyCloud);
- 5 установленных переменных окружения.

Готовы к такому испытанию?

2.4 Получение исходного кода проекта

Я бы мог по старинке приложить к книге компакт-диск с исходным кодом, так ведь? Но как насчёт использовать Git-репозиторий?

Клонируйте *репозиторий гостевой книги* на свой компьютер:

```
$ symfony new --version=5.0-1 --book guestbook
```

Этот репозиторий содержит весь код, который встречается в книге.

Обратите внимание, что мы используем команду `symfony new` вместо `git clone`, потому что она не только позволяет клонировать репозиторий, который находится в организации `the-fast-track` на GitHub по адресу `https://github.com/the-fast-track/book-5.0-1`. Также эта команда запускает веб-сервер и контейнеры, применяет миграции базы данных, загружает фикстуры и т.д. После выполнения команды будет запущен сайт, который можно сразу начать использовать.

Полученный код полностью соответствует тому, который присутствует в книге (используйте указанный выше URL-адрес

репозитория). Крайне сложно вручную поддерживать один и тот же код одновременно и в книге, и в репозитории. Я пытался ранее, но безуспешно. Я бы даже сказал, что это невозможно. Особенно в книгах, подобной этой, где поэтапно рассказывается, как разрабатывать сайт. Тем более учитывая, что каждая глава зависит от предыдущей, и очередное изменение может не самым лучшим образом отразиться во всех последующих главах.

К счастью, Git-репозиторий для книги *автоматически генерируется* из содержимого книги. Да, да, вы правильно поняли. Мне нравится всё автоматизировать, поэтому я написал скрипт, который считывает всю книгу и создаёт Git-репозиторий. У такого подхода есть приятный побочный эффект: при обновлении книги скрипт ничего не создаст, если будут противоречивые изменения, либо если я забуду обновить некоторые инструкции. Всё верно, это BDD (Book-Driven Development или разработка через написание книги)!

2.5 Навигация по коду

Более того, репозиторий — это не только окончательная версия кода в ветке `master`. Скрипт выполняет любое действие, описанное в книге, и в конце каждого раздела фиксирует изменения в репозитории. Вдобавок он помечает каждый шаг и подшаг, чтобы облегчить просмотр соответствующего кода. Разве это не здорово?

Если вам лень, то вы можете получить код нужного шага по его тегу. Например, если вы хотите посмотреть и попробовать в действии код в конце 10 шага, выполните следующую команду:

```
$ symfony book:checkout 10
```

Как и при клонировании репозитория, вместо команды `git checkout` мы используем `symfony book:checkout`. Благодаря этой команде, вне зависимости от текущего шага, у вас будет полноценно работающий сайт на указанном шаге. **Имейте в виду, все данные и код, а также контейнеры будут удалены после выполнения этой операции.**

Вы также можете перейти на любой подшаг:

```
$ symfony book:checkout 10.2
```

Повторюсь: очень рекомендую самостоятельно писать код. Но если вы столкнулись с затруднениями, попробуйте сравнить ваш код с тем, что в книге.

Не уверены, что всё сделали правильно в подшаге 10.2? Сравните с вашим кодом:

```
$ git diff step-10-1...step-10-2

# And for the very first substep of a step:
$ git diff step-9...step-10-1
```

Хотите узнать, когда файл был создан или изменён?

```
$ git log -- src/Controller/ConferenceController.php
```

Вы также можете просматривать изменения, теги и коммиты прямо на GitHub. Это отличный способ скопировать код, особенно если вы читаете бумажное издание книги!

Шаг 3
С нуля до развёртывания

Мне нравится работать быстро. Я хочу, чтобы наш маленький проект был доступен в продакшене как можно скорее. Вот прямо сейчас. Поскольку мы ещё ничего не сделали, начнём с развёртывания простой и понятной страницы-заглушки типа «Under construction». Вам это понравится!

Потратьте немного времени, чтобы найти в интернете наиболее подходящую, старомодную анимированную картинку с надписью «Under construction». Вот *эту GIF-картинку* я собираюсь использовать:

Я же вам говорил, что будет очень весело.

3.1 Инициализация проекта

Создайте новый Symfony-проект с помощью CLI-утилиты `symfony`, которую мы ранее установили:

```
$ symfony new guestbook --version=5.0
$ cd guestbook
```

Эта команда представляет собой обертку над `Composer`, которая облегчает создание проектов на Symfony. Она использует *заготовку проекта*, которая включает в себя минимальный набор зависимостей; компоненты Symfony, необходимые практически для любого проекта: консольная утилита и HTTP-абстракция, используемая для создания веб-приложений.

Если вы посмотрите на скелет проекта в репозитории на GitHub, то заметите, что он почти пуст. Там только лишь файл `composer.json`. Однако в директории `guestbook` полно файлов. Как это вообще возможно? Ответ кроется в пакете `symfony/flex`. Symfony Flex — это плагин для Composer, который внедряется в процесс установки. Когда он обнаруживает пакет, который содержит так называемый *рецепт*, Composer выполняет его.

Основной точкой входа рецептов Symfony является файл манифеста, в котором описаны операции, которые необходимо выполнить, чтобы автоматически зарегистрировать пакет в Symfony-приложении. Вам никогда не придётся заглядывать в файл README для установки пакета Symfony. Автоматизация — ключевая особенность Symfony.

Учитывая, что Git установлен на вашем компьютере, команда `symfony new` также создала Git-репозиторий и сделала в нём первый коммит.

Посмотрите на структуру директорий:

```
├── bin/
├── composer.json
├── composer.lock
├── config/
├── public/
├── src/
├── symfony.lock
├── var/
└── vendor/
```

Директория `bin/` содержит основной скрипт командной строки `console`. Вы будете использовать его постоянно.

Директория `config/` состоит из набора готовых конфигурационных файлов. По одному файлу на каждый пакет. Вам редко предстоит их редактировать, поскольку значения по умолчанию хорошо подходят для приложения.

Директория `public/` — это корневая директория сайта, а скрипт `index.php` в ней — основная точка входа для всех динамических HTTP-ресурсов.

Директория `src/` содержит весь код, который вы напишете — в ней вы будете проводить большую часть времени. По умолчанию все классы в этой директории используют пространство имён `App`. Это ваша рабочая директория, ваш код, ваша бизнес-логика. Symfony имеет мало общего с этим.

Директория `var/` содержит кеш-файлы, логи и прочие файлы, сгенерированные приложением во время выполнения; не обращайте внимания на неё. Это единственная директория, которая должна быть доступна для записи в продакшене.

Директория `vendor/` содержит все пакеты, которые установил Composer, включая и сам Symfony. Это наше секретное оружие для максимальной продуктивности. Давайте не будем изобретать велосипед, а вместо этого воспользуемся существующими библиотеками для решения сложных задач. В этой директории всем заведует Composer, поэтому ничего в ней не изменяйте.

Это всё, что вам нужно знать на данный момент.

3.2 Создание публичных ресурсов

К всему, что находится в директории `public/`, можно обратиться из браузера. Например, если вы переместите анимированный GIF-файл (назовите его `under-construction.gif`) в новую директорию `public/images/`, он будет доступен по URL-адресу, такому как `https://localhost/images/under-construction.gif`.

Вы можете скачать мой GIF-файл отсюда:

```
$ mkdir public/images/
$ php -r "copy('http://clipartmag.com/images/website-under-construction-
image-6.gif', 'public/images/under-construction.gif');"
```

3.3 Запуск локального веб-сервера

Консольная утилита `symfony` включает в себя веб-сервер, оптимизированный для разработки. Как вы можете себе представить, он очень хорошо работает с Symfony. Однако никогда не используйте его в продакшене.

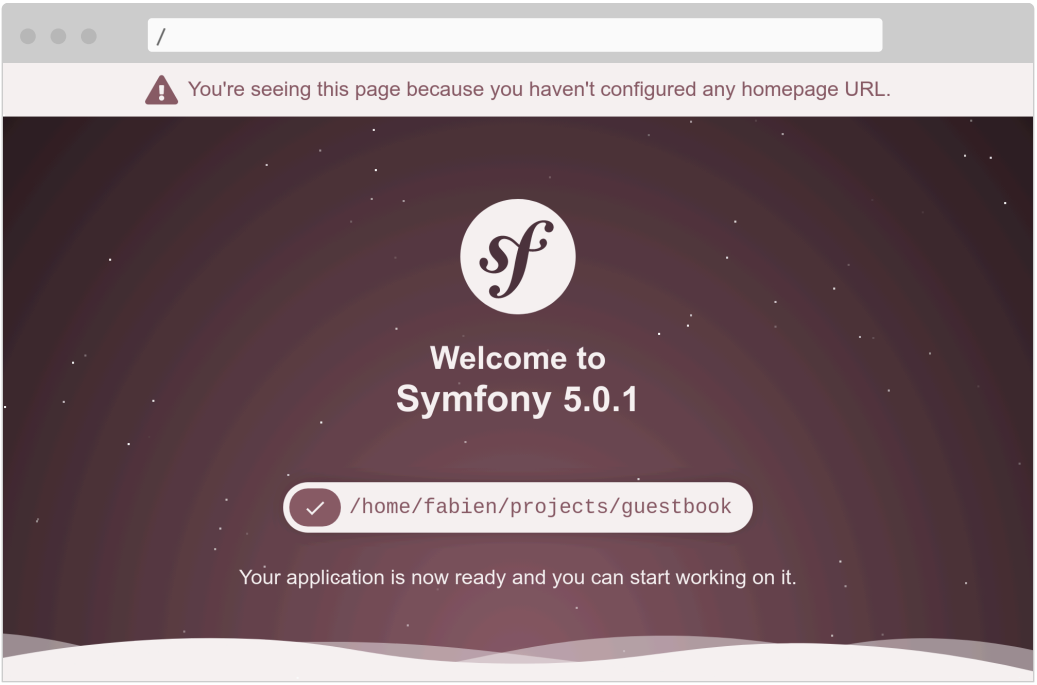
Запустите веб-сервер в фоновом режиме прямо из директории проекта, используя флаг `-d`:

```
$ symfony server:start -d
```

Сервер запустится на первом доступном порту, начиная с 8000. Вы можете открыть сайт по ссылке из CLI:

```
$ symfony open:local
```

В вашем браузере по умолчанию откроется новая вкладка, на которой будет примерно следующее:



Для поиска причин неполадок используйте команду `symfony server:log`; эта команда в режиме реального времени выводит последние строки из логов веб-сервера, PHP и самого приложения.

Перейдите к `/images/under-construction.gif`. Выглядит так же?

Довольны результатом? Теперь давайте закоммитим нашу работу:

```
$ git add public/images
$ git commit -m'Add the under construction image'
```

3.4 Добавление фавиконки

Чтобы убрать из логов большое количество HTTP-ошибок 404 из-за отсутствующей фавиконки, которую автоматически запрашивают

браузеры, давайте добавим её:

```
$ php -r "copy('https://symfony.com/favicon.ico', 'public/favicon.ico');"
$ git add public/
$ git commit -m'Add a favicon'
```

3.5 Подготовка к развёртыванию в продакшене

Как насчёт развёртывания нашей работы в продакшене? Я в курсе, что у нас пока ещё нет HTML-страницы, чтобы поприветствовать пользователей. Но даже если мы просто посмотрим на маленькую картинку с надписью «under construction» в продакшене — это будет большим шагом вперёд. Ну, вы наверняка знаете этот девиз: *разворачивай приложение как можно раньше и чаще*.

Вы можете разместить приложение у любого провайдера с поддержкой PHP… то есть почти на всех хостингах. Но учтите несколько требований: поддержка последней версии PHP, наличие баз данных, сервера очереди и т.п.

Я сделал свой выбор: это будет *SymfonyCloud*. У него есть всё необходимое, а кроме этого он помогает финансировать разработку Symfony.

В консольной команде `symfony` есть встроенная поддержка SymfonyCloud. Давайте создадим проект в SymfonyCloud:

```
$ symfony project:init
```

Эта команда создаёт несколько файлов, которые требуются для работы с SymfonyCloud: `.symfony/services.yaml`, `.symfony/routes.yaml` и `.symfony.cloud.yaml`.

Добавьте их в Git и закоммитьте:

```
$ git add .
$ git commit -m"Add SymfonyCloud configuration"
```

Использование универсальной команды `git add .` может быть рискованно. Но не в нашем случае, так как все файлы, которые нам не нужно коммитить, будут исключены автоматически благодаря сгенерированному во время создания проекта файлу `.gitignore`.

3.6 Развёртывание в продакшене

Не пора ли нам развернуть приложение?

Создайте новый проект SymfonyCloud:

```
$ symfony project:create --title="Guestbook" --plan=development
```

Эта команда выполняет ряд операций:

- При первом запуске этой команды нужно пройти аутентификацию с помощью учётной записи SymfonyConnect, если вы ранее этого не сделали.
- Команда создаст новый проект на SymfonyCloud (в течение 7 дней вы можете *бесплатно* разместить любой новый проект).

Наконец, разворачиваем приложение:

```
$ symfony deploy
```

Приложение разворачивается автоматически каждый раз при отправке новых изменений в Git-репозиторий. После того как команда отработает, проект будет доступен по уникальному доменному имени.

Убедитесь, что всё работает хорошо:

```
$ symfony open:remote
```

Вы должны получить ошибку 404, но перейдя на `/images/under-construction.gif`, вы увидите GIF-изображение.

Обратите внимание, что вы не увидите стандартную красивую страницу Symfony на SymfonyCloud. Почему? Скоро вы узнаете, что Symfony поддерживает работу с несколькими окружениями, а SymfonyCloud автоматически развёрнул код в окружении продакшена.

Если вы хотите удалить проект на SymfonyCloud, воспользуйтесь командой `project:delete`.

Двигаемся дальше

- *Сервер рецептов Symfony*, где вы можете найти все рецепты, которые можете использовать в собственных приложениях на Symfony;
- Репозитории *официальных рецептов Symfony* и *рецептов сообщества*, где вы можете разместить свои рецепты;
- *Локальный веб-сервер Symfony*;
- *Документация SymfonyCloud*.

Шаг 4

Выбор методологии разработки

Учить — значит повторять что-то одно и то же снова и снова. Я не буду применять такой подход, обещаю. В конце каждого шага исполните победный танец и сохраните результат проделанной работы. Это подобно нажатию `Ctrl+S`, только для всего сайта целиком.

4.1 Стратегия использования Git

В конце каждого шага не забудьте зафиксировать изменения в git:

```
$ git add .
$ git commit -m'Add some new feature'
```

Вы можете смело добавлять в коммит все файлы, потому что при создании Symfony-проекта уже был подготовлен файл `.gitignore`. Кроме того, каждый пакет может добавлять свои собственные шаблоны игнорируемых файлов в `.gitignore`. Взгляните на текущее содержимое:

.gitignore

```
###> symfony/framework-bundle ###
/.env.local
/.env.local.php
/.env.*.local
/public/bundles/
/var/
/vendor/
###< symfony/framework-bundle ###
```

Эти странные строки — маркеры, добавленные Symfony Flex, чтобы определить, что нужно удалить, если вы решите удалить зависимость. Как я упоминал ранее, вся эта утомительная работа выполняется автоматически Symfony.

Теперь самое время разместить репозиторий на удалённый сервер. Для этого прекрасно подойдёт GitHub, GitLab или Bitbucket.

Если вы развёртываете проект на SymfonyCloud, у вас уже есть копия Git-репозитория, но вы не должны её использовать. Поскольку она не является резервной копией и предназначена только для развёртывания.

4.2 Развёртывание в продакшене с помощью непрерывной интеграции

Ещё одной хорошей практикой является частое развёртывание. Развёртывание в конце каждого шага — хорошо и полезно:

```
$ symfony deploy
```

Шаг 5

Поиск и устранение неисправностей

Настройка проекта также подразумевает наличие правильных инструментов для отладки.

5.1 Установка дополнительных зависимостей

Помните, наш проект создан с очень небольшим количеством зависимостей: без шаблонизатора, без инструментов для отладки, без логера. Такой подход позволяет добавить другие зависимости только тогда, когда они вам нужны. Зачем вам шаблонизатор, если вы разрабатываете HTTP API или CLI-инструмент?

Как добавить больше зависимостей? Через Composer. Помимо «обычных» Composer-пакетов, мы будем работать с двумя «специальными» видами пакетов:

- *Компоненты Symfony*: пакеты с базовой функциональностью и низкоуровневыми абстракциями, необходимые большинству приложений (маршрутизация, консоль, HTTP-клиент, почтовый клиент, кеш и т.д.);
- *Бандлы Symfony:* пакеты, которые добавляют высокоуровневую функциональность или предлагают интеграцию со сторонними библиотеками (эти бандлы в основном разрабатываются сообществом).

Для начала давайте добавим Symfony Profiler, который поможет сэкономить много времени в поиске источника проблемы:

```
$ symfony composer req profiler --dev
```

`profiler` — это псевдоним для пакета `symfony/profiler-pack`.

Псевдонимы — это не функция самого Composer, а концепция Symfony, чтобы облегчить нам жизнь. Псевдонимы — это ярлыки для популярных Composer-пакетов. Вашему приложению нужен ORM? Установите `orm`. Хотите разработать API? Установите `api`. Псевдонимы автоматически преобразуются в один или несколько обычных Composer-пакетов. Псевдонимы назначаются основной командой Symfony.

Ещё одна интересная особенность — можно не указывать вендора `symfony` в имени пакета. Поэтому, например, вместо `symfony/cache` набирайте `cache`.

Мы уже упоминали Composer-плагин `symfony/flex`. Псевдонимы — лишь одна из его функциональных возможностей.

5.2 Понимание окружений Symfony

Вы заметили флаг `--dev` при использовании команды `composer req`? Поскольку Symfony Profiler имеет смысл использовать во время разработки, то не стоит устанавливать его в продакшене.

Symfony поддерживает создание *окружений*. По умолчанию есть три окружения с возможностью добавить дополнительные: `dev`, `prod` и `test`. Все окружения используют один и тот же код, но имеют разные

конфигурации.

Например, в окружении `dev` все инструменты отладки включены. Когда как в окружении `prod` такого нет, потому что приложение должно быть оптимизировано для повышения производительности.

Изменяя значение переменной окружения `APP_ENV` можно переключаться с одного окружения на другое.

Во время развёртывания на SymfonyCloud, окружение (сохранённое в `APP_ENV`) автоматически переключится на `prod`.

5.3 Управление конфигурациями окружений

Переменная `APP_ENV` может быть задана с помощью «настоящих» переменных окружения в терминале:

```
$ export APP_ENV=dev
```

Переменные окружения, такие как `APP_ENV`, рекомендуется явно определять на продакшен-серверах. Однако в процессе разработки установка таким образом множество переменных окружения может стать утомительной. Поэтому вместо этого определите их в файле `.env`.

При создании проекта был автоматически сгенерирован файл `.env`:

.env

```
###> symfony/framework-bundle ###
APP_ENV=dev
APP_SECRET=c2927f273163f7225a358e3a1bbbed8a
#TRUSTED_PROXIES=127.0.0.1,127.0.0.2
#TRUSTED_HOSTS='^localhost|example\.com$'
###< symfony/framework-bundle ###
```

Благодаря использованию рецептов Symfony Flex, любой пакет может добавить свои переменные окружения в этот файл.

Файл `.env` хранится в репозитории и содержит значения *по умолчанию* для продакшена. Вы можете задать свои значения, создав файл `.env.local`. Этот файл не хранится в репозитории, поэтому изначально

игнорируется в `.gitignore`.

Никогда не храните конфиденциальную информацию в этих файлах. Позже мы рассмотрим, как управлять такими видами данных.

5.4 Логирование всех действий

По умолчанию возможности отладки и логирования ограничены в новых проектах. Давайте добавим дополнительные инструменты, которые помогут нам в решении проблем как в процессе разработке, так и в продакшене:

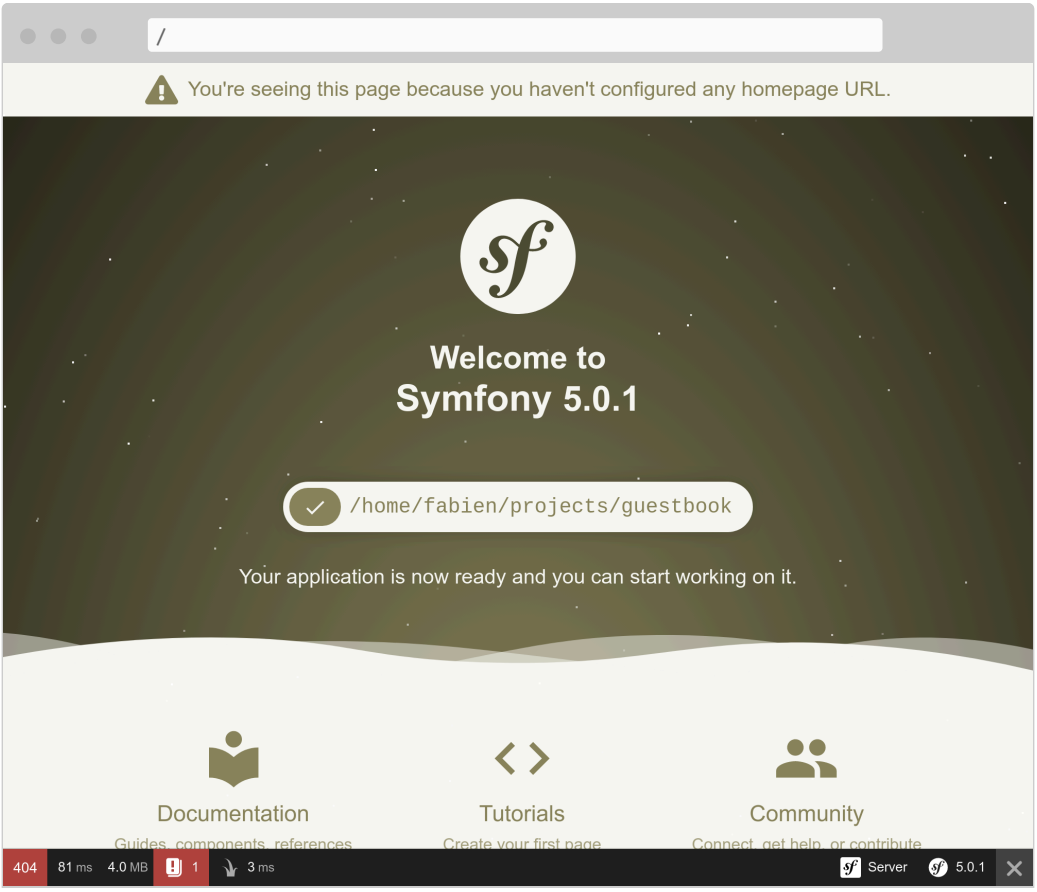
```
$ symfony composer req logger
```

Установим инструменты отладки только в окружении разработчика:

```
$ symfony composer req debug --dev
```

5.5 Изучение средств отладки Symfony

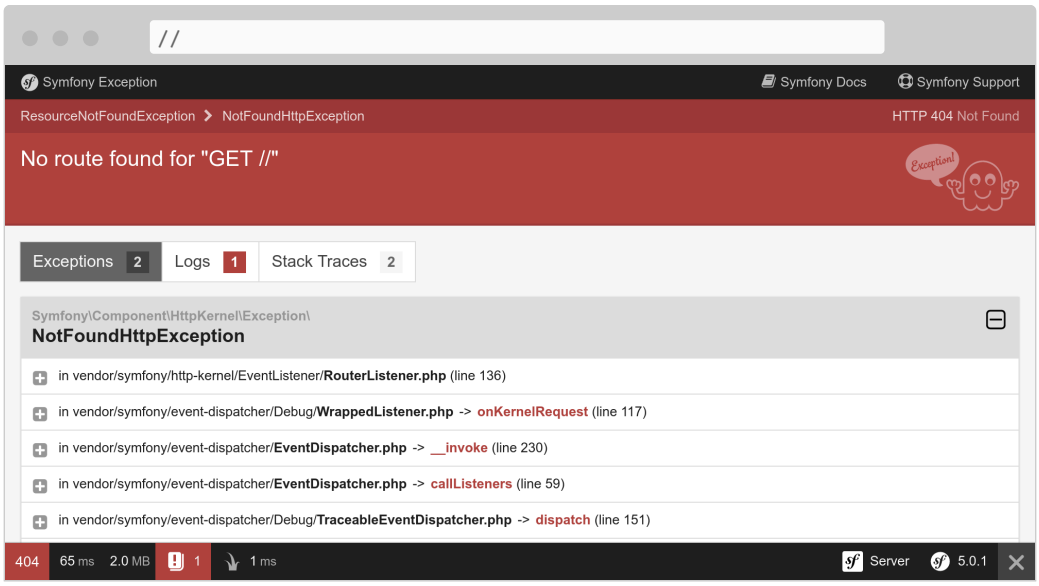
При обновлении главной страницы в нижней части экрана должна появиться панель отладки:



Первое, на что вы, скорее всего, обратите внимание — надпись **404** на красном фоне. Помните, это просто страница-заглушка, поскольку мы ещё не определили домашнюю страницу. И даже если эта страница достаточно хорошо выглядит, она всё ещё остаётся страницей ошибки. Так что правильный код статуса HTTP для этой страницы — 404, а не 200. Благодаря панели отладки, вы сразу же получите всю необходимую информацию.

Нажмите на маленький восклицательный знак и вы увидите сообщение «настоящего» исключения в логах профилировщика Symfony. Если вы хотите увидеть трассировку стека, нажмите на ссылку «Exception» в левом меню.

Когда возникает проблема с кодом, вы увидите похожую страницу об ошибке со всей необходимой информацией для отладки:



Уделите немного времени и поизучайте данные в профилировщике Symfony, нажимая по разным ссылкам.

Логи весьма полезны при отладке. В Symfony есть удобная команда для отображения последних строк всех логов (веб-сервера, PHP и вашего приложения):

```
$ symfony server:log
```

Проведем небольшой эксперимент. Откройте `public/index.php` и сделайте ошибку в PHP-коде (например, добавьте foobar посередине кода). Обновите страницу в браузере и понаблюдайте за логом:

```
Dec 21 10:04:59 |DEBUG| PHP    PHP Parse error:  syntax error, unexpected 'use'
(T_USE) in public/index.php on line 5 path="/usr/bin/php7.42" php="7.42.0"
Dec 21 10:04:59 |ERROR| SERVER GET  (500) / ip="127.0.0.1"
```

Логи отображаются разными цветами, чтобы привлечь ваше внимание к ошибкам.

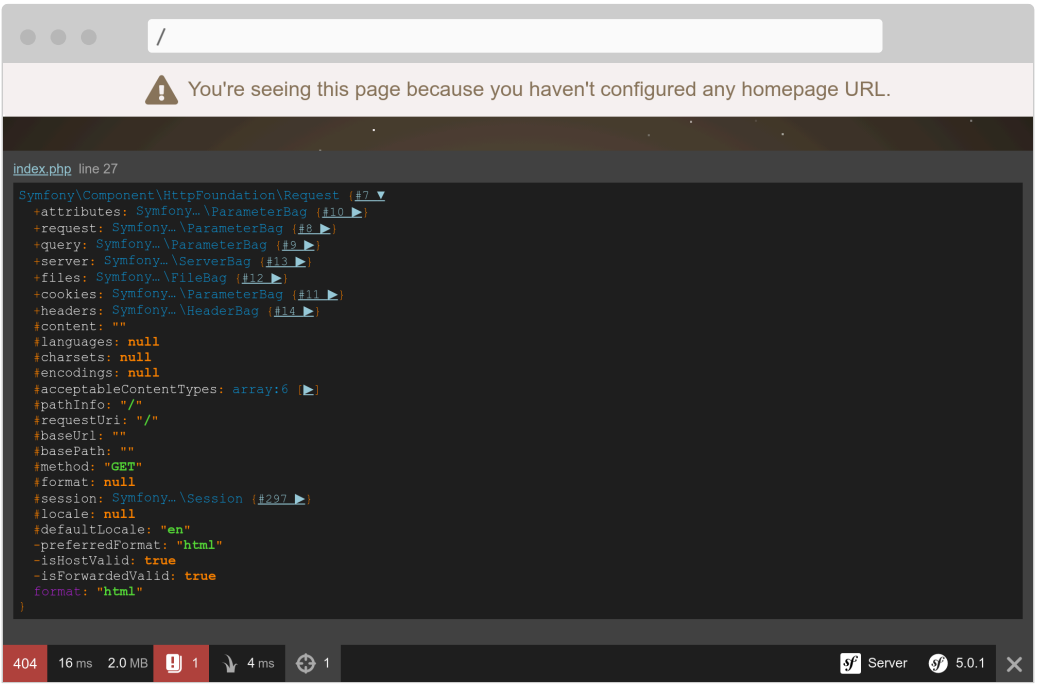
Symfony-функция `dump()` — ещё один замечательный помощник во время отладки. Она глобально доступна и отображает значение переменных в удобном интерактивном формате.

Временно измените файл `public/index.php`, чтобы вывести объект Request:

```
--- a/public/index.php
+++ b/public/index.php
```

```
@@ -23,5 +23,8 @@ if ($trustedHosts = $_SERVER['TRUSTED_HOSTS'] ??
$_ENV['TRUSTED_HOSTS'] ?? false
 $kernel = new Kernel($_SERVER['APP_ENV'], (bool) $_SERVER['APP_DEBUG']);
 $request = Request::createFromGlobals();
 $response = $kernel->handle($request);
+
+dump($request);
+
 $response->send();
 $kernel->terminate($request, $response);
```

После обновления страницы обратите внимание на новую иконку с мишенью. Она позволит вам посмотреть детали объекта. Щёлкните по ней, чтобы перейти на отдельную страницу с полной информацией об объекте:



Отмените это изменение кода перед коммитом других изменений, которые были сделаны на этом шаге:

```
$ git checkout public/index.php
```

5.6 Настройка среды разработки

В локальном окружении разработки при генерации исключения Symfony отображает страницу с сообщением исключения и его трассировкой. К отображаемому пути файла в трассировке добавляется ссылка, кликнув на которую можно открыть файл на нужной строке прямо в вашей IDE. Но чтобы воспользоваться этой возможностью, сначала вам нужно настроить IDE. Symfony поддерживает множество разных IDE; я использую Visual Studio Code для данного проекта:

```
--- a/config/packages/framework.yaml
+++ b/config/packages/framework.yaml
@@ -14,3 +14,5 @@ framework:
     #fragments: true
     php_errors:
         log: true
+
+    ide: vscode
```

Ссылка в имени файла появляется не только при генерации исключения. Так, например, контроллер на панели отладки может стать кликабельным, если настроить IDE.

5.7 Отладка в продакшене

Отладка на продакшен-серверах всегда сложнее. К примеру, в таком случае у вас нет доступа к профилировщику Symfony. А в логах не так много подробной информации. Но всё же можно посмотреть последние записи логов:

```
$ symfony logs
```

Вы даже можете подключиться через SSH из веб-контейнера:

```
$ symfony ssh
```

Не бойтесь — вы не сможете так просто всё сломать. Большая часть файловой системы доступна только для чтения. Поэтому сделать срочное исправление прямо на продакшене не получится. Однако вы

узнаете про гораздо более подходящий способ сделать это позже в книге.

Двигаемся дальше

- *Обучающий видеоролик по файлам окружений и конфигураций на SymfonyCast*;
- *Обучающий видеоролик по переменным окружения на SymfonyCast*;
- *Обучающий видеоролик по панели отладки и профилировщику на SymfonyCasts*;
- *Управление несколькими файлами .env* в приложениях Symfony.

Шаг 6
Создание контроллера

Наш проект гостевой книги уже работает в продакшене, однако мы немного схитрили. У проекта пока ещё нет ни одной веб-страницы, а на главной странице по-прежнему отображается ошибка 404. Давайте исправим это.

Когда приходит HTTP-запрос, например, на главную страницу (`http://localhost:8000/`), Symfony пытается найти *маршрут* , соответствующий *пути запроса* (`/` в данном случае). *Маршрут* — это связующее звено между путём запроса и *callback-функцией PHP*, которая создаёт *ответ* HTTP для этого запроса.

Эти callback-функции PHP называются «контроллерами». В Symfony большинство контроллеров реализовано в виде PHP-классов. Конечно, вы можете вручную создать такой класс, но для быстроты разработки давайте посмотрим, как Symfony в этом может нам помочь.

6.1 Ускорение разработки с

помощью бандла Maker

Для лёгкой генерации контроллеров мы можем использовать пакет `symfony/maker-bundle`:

```
$ symfony composer req maker --dev
```

Так как бандл Maker полезен лишь в процессе разработки, не забудьте указать флаг `--dev`, чтобы пакет не загружался при развёртывании приложения в продакшене.

Бандл Maker поможет вам сгенерировать множество различных классов. Мы будем использовать его на протяжении всей книги. Каждый «генератор» определён в отдельной команде, при этом все команды принадлежат одному и тому же пространству имён команды `make`.

Компонент Symfony Console имеет встроенную команду `list`, которая выводит список всех команд в указанном пространстве имен; используйте её, чтобы посмотреть все доступные генераторы бандла Maker:

```
$ symfony console list make
```

6.2 Выбор формата конфигурации

Перед созданием нашего первого контроллера в проекте, нам необходимо определиться с используемыми форматами для написания конфигураций. По умолчанию Symfony поддерживает YAML, XML, PHP и аннотации.

YAML идеален для *конфигурации пакетов*. Этот формат будет использоваться в файлах директории `config/`. Зачастую, когда вы устанавливаете новый пакет, рецепт этого пакета добавляет новый файл с расширением `.yaml` в эту директорию.

Для *конфигурации, связанной с PHP-кодом*, лучше всего подойдут *аннотации*, поскольку они указываются рядом с кодом. Сейчас объясню на примере. Когда приходит запрос, определённая конфигурация должна передать Symfony, что текущий путь запроса

должен обрабатываться соответствующим контроллером (то есть PHP-классом). При использовании конфигурационных форматов на YAML, XML или PHP потребуется включать два файла (сам конфигурационный файл и PHP-файл контроллера). В то же время с помощью аннотаций можно всё сконфигурировать непосредственно в самом классе контроллера.

Для работы с аннотациями нужно добавить ещё одну зависимость:

```
$ symfony composer req annotations
```

Вам может быть интересно узнать, какой именно пакет нужно установить, чтобы добавить поддержку той или иной функциональной возможности? Как правило, вам не нужно этого знать. Поскольку в большинстве случаев Symfony показывает отсутствующий пакет в сообщении об ошибке. К примеру, выполнение команды `symfony make:controller` без установленного пакета `annotations` выбросит исключение с подсказкой о том, какой пакет следует установить.

6.3 Генерация контроллера

Создайте свой первый *контроллер* с помощью команды `make:controller`:

```
$ symfony console make:controller ConferenceController
```

Команда создает класс `ConferenceController` в директории `src/Controller/`. Сгенерированный класс содержит немного шаблонного кода, который может быть изменён:

src/Controller/ConferenceController.php

```
namespace App\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Annotation\Route;

class ConferenceController extends AbstractController
{
    /**
     * @Route("/conference", name="conference")
```

```
     */
    public function index()
    {
        return $this->render('conference/index.html.twig', [
            'controller_name' => 'ConferenceController',
        ]);
    }
}
```

Аннотация `@Route("/conference", name="conference")` — это именно то, что делает метод `index()` контроллером (конфигурация находится рядом с кодом, который она настраивает).

При переходе в браузере по пути `/conference` выполняется контроллер, который возвращает HTTP-ответ.

Настройте маршрут так, чтобы он соответствовал главной странице:

```
--- a/src/Controller/ConferenceController.php
+++ b/src/Controller/ConferenceController.php
@@ -8,7 +8,7 @@ use Symfony\Component\Routing\Annotation\Route;
 class ConferenceController extends AbstractController
 {
     /**
-     * @Route("/conference", name="conference")
+     * @Route("/", name="homepage")
      */
     public function index()
     {
```

Параметр маршрута `name` пригодится нам, когда понадобится получить ссылку на главную страницу. Таким образом, вместо жёстко заданного в коде пути `/`, мы будем использовать имя маршрута.

Теперь вернём обычный HTML-код вместо шаблона по умолчанию:

```
--- a/src/Controller/ConferenceController.php
+++ b/src/Controller/ConferenceController.php
@@ -3,6 +3,7 @@
 namespace App\Controller;

 use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
+use Symfony\Component\HttpFoundation\Response;
 use Symfony\Component\Routing\Annotation\Route;

 class ConferenceController extends AbstractController
```

```
@@ -12,8 +13,13 @@ class ConferenceController extends AbstractController
      */
     public function index()
     {
-        return $this->render('conference/index.html.twig', [
-            'controller_name' => 'ConferenceController',
-        ]);
+        return new Response(<<<EOF
+<html>
+    <body>
+        <img src="/images/under-construction.gif" />
+    </body>
+</html>
+EOF
+        );
     }
 }
```

Перезагрузите главную страницу в браузере:

Основная задача контроллера — вернуть объект `Response` в качестве ответа на HTTP-запрос.

6.4 Добавление пасхального яйца

Для демонстрации того, как в ответе можно использовать информацию из запроса, давайте добавим небольшое *пасхальное яйцо*. При переходе на главную страницу, если в строке запроса мы находим что-то типа `?hello=Fabien`, то показываем приветственное сообщение.

```
--- a/src/Controller/ConferenceController.php
+++ b/src/Controller/ConferenceController.php
```

```
@@ -3,6 +3,7 @@
 namespace App\Controller;

 use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
+use Symfony\Component\HttpFoundation\Request;
 use Symfony\Component\HttpFoundation\Response;
 use Symfony\Component\Routing\Annotation\Route;

@@ -11,11 +12,17 @@ class ConferenceController extends AbstractController
     /**
      * @Route("/", name="homepage")
      */
-    public function index()
+    public function index(Request $request)
     {
+        $greet = '';
+        if ($name = $request->query->get('hello')) {
+            $greet = sprintf('<h1>Hello %s!</h1>', htmlspecialchars($name));
+        }
+
         return new Response(<<<EOF
 <html>
     <body>
+        $greet
         <img src="/images/under-construction.gif" />
     </body>
 </html>
```

Посмотреть данные запроса в Symfony можно через объект `Request`. Если в объявлении типа для аргумента контроллера указать объект запроса, Symfony автоматически передаст его вам. Далее используем этот объект, чтобы получить параметр `name` из строки запроса и добавляем его в заголовок `<h1>`.

Чтобы увидеть получившийся результат, сначала в браузере перейдите на главную страницу (`/`), а после по пути `/?hello=Fabien`.

Обратите внимание на вызов функции `htmlspecialchars()`, который нужен, чтобы защититься от XSS-атак. Когда мы начнём использовать шаблонизатор, он автоматически защитит нас от подобных проблем с безопасностью.

Также отмечу, что имя можно сделать частью URL-адреса:

```
--- a/src/Controller/ConferenceController.php
```

```
+++ b/src/Controller/ConferenceController.php
@@ -9,13 +9,19 @@ use Symfony\Component\Routing\Annotation\Route;
 class ConferenceController extends AbstractController
 {
     /**
-     * @Route("/", name="homepage")
+     * @Route("/hello/{name}", name="homepage")
      */
-    public function index()
+    public function index(string $name = '')
     {
+        $greet = '';
+        if ($name) {
+            $greet = sprintf('<h1>Hello %s!</h1>', htmlspecialchars($name));
+        }
+
         return new Response(<<<EOF
 <html>
     <body>
+        $greet
         <img src="/images/under-construction.gif" />
     </body>
 </html>
```

Часть маршрута вроде {name} называется *параметром маршрута* — он работает так же, как и знак подстановки. Теперь можно перейти по пути /hello, а затем на /hello/Fabien — результат будет одинаковый. *Значение* параметра {name} можно получить, добавив *одноимённый* аргумент в контроллер (то есть $name).

Двигаемся дальше

- Система *маршрутизации* в Symfony;
- *Обучающий видеоролик по маршрутам, контроллерам и страницам на SymfonyCasts*;
- *Аннотации*; в PHP;
- Компонент *HttpFoundation*;
- Методика атаки через *межсайтовый скриптинг (Cross-Site Scripting или XSS)*;
- *Шпаргалка по маршрутизации в Symfony*.

Шаг 7

Подготовка базы данных

Сайт гостевой книги конференции предназначен для сбора отзывов во время конференций. Нам необходимо хранить комментарии, оставленные участниками конференции.

Лучше всего комментарий описывается следующей структурой данных: автор, его электронная почта, текст сообщения и фотография (необязательно). Такого рода информацию лучше всего хранить в традиционной реляционной базе данных.

Мы будем использовать сервер базы данных PostgreSQL.

7.1 Добавление PostgreSQL в Docker Compose

На нашей локальной машине, для управления сервисами мы решили использовать Docker. Создайте файл `docker-compose.yaml` и добавьте PostgreSQL в качестве сервиса:

docker-compose.yaml

```
version: '3'

services:
    database:
        image: postgres:11-alpine
        environment:
            POSTGRES_USER: main
            POSTGRES_PASSWORD: main
            POSTGRES_DB: main
        ports: [5432]
```

В результате будет установлен сервер PostgreSQL версии 11 и настроены переменные окружения для имени базы данных и учётных данных. Конкретные значения сейчас не важны.

Также откроем порт PostgreSQL (`5432`) у контейнера для доступа к нему с локального хоста, чтобы получить доступ к базе данных с нашего компьютера.

Модуль `pdo_pgsql` должен был быть установлен на предыдущем шаге, вместе с установкой PHP.

7.2 Запуск Docker Compose

Запустите Docker Compose в фоновом режиме (`-d`):

```
$ docker-compose up -d
```

Подождите немного, пока база данных запустится, а затем проверьте, что всё работает нормально:

```
$ docker-compose ps

        Name                      Command             State           Ports
-------------------------------------------------------------------------------------
guestbook_database_1   docker-entrypoint.sh postgres   Up      0.0.0.0:32780->5432/tcp
```

Если работающих контейнеров нет, или в столбце `State` не отображается `Up`, проверьте логи Docker Compose:

```
$ docker-compose logs
```

7.3 Обращение к локальной базе данных

Использование консольной программы `psql` может пригодится в отдельных случаях. Хотя для этого вам нужно помнить учётные данные и имя базы. Вам также нужно знать локальный порт, на котором запущена база данных. Docker выбирает произвольный порт для того, чтобы вы могли одновременно работать над разными проектами, использующими PostgreSQL (локальный порт отображается в выводе команды `docker-compose ps`).

Если вы запускаете `psql` с помощью Symfony CLI, вам не нужно ничего помнить.

Symfony CLI автоматически обнаруживает сервисы Docker, запущенные для проекта, и устанавливает переменные окружения, необходимые `psql` для подключения к базе данных.

Благодаря этим соглашениям, доступ к базе данных с помощью команды `symfony run` становится намного проще:

```
$ symfony run psql
```

Если на вашем локальном хосте не установлена команда `psql`, вы можете запустить её через `docker`:

```
$ docker exec -it guestbook_database_1 psql -U main -W main
```

7.4 Добавление PostgreSQL в SymfonyCloud

Добавление такого сервиса, как PostgreSQL, в инфраструктуру продакшена на SymfonyCloud, делается через изменения в файле

`.symfony/services.yaml`, который пока ещё пуст:

.symfony/services.yaml

```
db:
    type: postgresql:11
    disk: 1024
    size: S
```

Сервис `db` — это PostgreSQL 11 (как и в Docker), который мы разместим в небольшом контейнере с диском объёмом 1 Гб.

Также необходимо «привязать» БД к контейнеру приложения, который описан в `.symfony.cloud.yaml`:

.symfony.cloud.yaml

```
relationships:
    database: "db:postgresql"
```

Сервис `db` типа `postgresql` указан как `database` в контейнере приложения.

Последним шагом будет добавление PHP-модуля `pdo_pgsql`:

.symfony.cloud.yaml

```
runtime:
    extensions:
        - pdo_pgsql
        # other extensions here
```

Вот полный результат изменений в файле `.symfony.cloud.yaml`:

```
--- a/.symfony.cloud.yaml
+++ b/.symfony.cloud.yaml
@@ -4,6 +4,7 @@ type: php:7.3

 runtime:
     extensions:
+        - pdo_pgsql
         - apcu
         - mbstring
         - sodium
@@ -12,6 +13,9 @@ runtime:
 build:
     flavor: none
```

```
+relationships:
+    database: "db:postgresql"
+
 web:
     locations:
         "/":
```

Зафиксируйте изменения в репозитории, а затем повторно разверните в SymfonyCloud:

```
$ git add .
$ git commit -m'Configuring the database'
$ symfony deploy
```

7.5 Доступ к базе данных на SymfonyCloud

PostgreSQL теперь работает как локально через Docker, так и на продакешене в SymfonyCloud.

Как мы только что увидели, при запуске `symfony run psql` происходит автоматическое подключение к базе данных, размещённой в контейнере Docker. Это происходит благодаря переменным окружения, установленным командой `symfony run`.

Если вы хотите подключиться к PostgreSQL, расположенном в контейнерах на продакшене, вы можете открыть SSH-туннель между вашей локальной машиной и инфраструктурой SymfonyCloud:

```
$ symfony tunnel:open --expose-env-vars
```

По умолчанию сервисы SymfonyCloud не отображаются в переменных окружения на локальной машине. Вы должны сделать это явно, используя флаг `--expose-env-vars`. Почему? Подключение к базе данных на продакшене является опасной операцией. Вы можете испортить *реальные* данные. Указывая этот флаг, вы подтверждаете, что *действительно* хотите это сделать.

Теперь подключитесь к удалённой базе данных PostgreSQL с помощью команды `symfony run psql`, как раньше:

```
$ symfony run psql
```

Когда завершите работу, не забудьте закрыть туннель:

```
$ symfony tunnel:close
```

Для выполнения SQL-запросов к базе данных на продакшене, вместо использования командной оболочки, вы можете использовать команду `symfony sql`.

7.6 Просмотр переменных окружения

Docker Compose и SymfonyCloud отлично работают с Symfony благодаря переменным окружения.

Просмотрите все переменные окружения, установленные `symfony`, выполнив `symfony var:export`:

```
$ symfony var:export

PGHOST=127.0.0.1
PGPORT=32781
PGDATABASE=main
PGUSER=main
PGPASSWORD=main
# ...
```

Переменные окружения, начинающиеся с `PG*` используются утилитой `psql`. А остальные?

Когда туннель к SymfonyCloud открыт с установленным флагом `--expose-env-vars`, команда `var:export` возвращает переменные из удалённого окружения:

```
$ symfony tunnel:open --expose-env-vars
$ symfony var:export
$ symfony tunnel:close
```

Двигаемся дальше

- *Сервисы SymfonyCloud*;
- *Туннель SymfonyCloud*;
- *Документация PostgreSQL*;
- *Команды docker-compose.*.

Шаг 8

Описание структуры данных

Для работы с базой данных в PHP мы будем использовать *Doctrine* — набор библиотек для управления базами данных:

```
$ symfony composer req orm
```

Эта команда устанавливает несколько зависимостей: Doctrine DBAL (слой абстракции базы данных), Doctrine ORM (библиотека для работы с содержимым базы данных через PHP-объекты) и Doctrine Migrations.

8.1 Настройка Doctrine ORM

Как узнать, подключена ли Doctrine к базе данных? В рецепт Doctrine добавлен конфигурационный файл `config/packages/doctrine.yaml`, в котором находятся параметры подключения. Основным параметром в этом файле является *DSN-строка* (Data Source Name — «имя источника данных»), содержащая всю информацию о подключении:

учётные данные, хост, порт и т.д. По умолчанию Doctrine ищет переменную среды DATABASE_URL.

8.2 Разбираемся в соглашениях по именованию переменных окружения в Symfony

Можно инициализировать переменную DATABASE_URL в файлах .env или .env.local. Благодаря рецепту пакета, пример значения переменной DATABASE_URL уже присутствует в файле .env. Это громоздкое решение, поскольку локальный порт PostgreSQL, открытый через Docker, может измениться. Есть вариант и получше.

Вместо записи переменной DATABASE_URL в файле, мы можем добавить ко всем командам префикс symfony и переменная окружения установится автоматически во всех сервисах, запущенных в Docker и/или SymfonyCloud (при открытом туннеле).

Docker Compose и SymfonyCloud отлично работают с Symfony благодаря переменным окружения.

Проверьте все установленные переменные окружения командой symfony var:export:

```
$ symfony var:export


DATABASE_URL=postgres://main:main@127.0.0.1:32781/
main?sslmode=disable&charset=utf8
# ...
```

Помните *имя сервиса* database, используемое в конфигурациях Docker и SymfonyCloud? Имена сервисов используются в качестве префиксов для определения переменных окружения, таких как DATABASE_URL. Если ваши сервисы названы в соответствии с соглашениями Symfony, никакой другой конфигурации не требуется.

Базы данных — это не единственный сервис, который использует соглашения Symfony. Например, то же самое можно сказать и о Mailer (через переменную окружения MAILER_DSN).

8.3 Изменение начального значения DATABASE_URL в файле .env

Всё же изменим переменную окружения `DATABASE_DSN` в файле `.env`, и укажем, что подключением по умолчанию должен быть PostgreSQL:

```
--- a/.env
+++ b/.env
@@ -25,5 +25,5 @@ APP_SECRET=447c9fa8420eb53bbd4492194b87de8f
 # For an SQLite database, use: "sqlite:///%kernel.project_dir%/var/data.db"
 # For a PostgreSQL database, use:
"postgresql://db_user:db_password@127.0.0.1:5432/
db_name?serverVersion=11&charset=utf8"
 # IMPORTANT: You MUST configure your server version, either here or in config/
packages/doctrine.yaml
-DATABASE_URL=mysql://db_user:db_password@127.0.0.1:3306/
db_name?serverVersion=5.7
+DATABASE_URL=postgresql://127.0.0.1:5432/db?serverVersion=11&charset=utf8
 ###< doctrine/doctrine-bundle ###
```

Почему эту информацию необходимо дублировать в двух разных местах? Потому что в *момент сборки* на некоторых облачных платформах URL-адрес базы данных может быть ещё неизвестен, но Doctrine необходимо знать драйвер базы данных, чтобы создать собственную конфигурацию. Таким образом, хост, имя пользователя и пароль не имеют значения.

8.4 Создание классов сущностей

Класс, описывающий объект конференции со следующими свойствами:

- *city* — город, в котором проводится конференция;
- *year* — год проведения конференции;
- *international* — флаг, указывающий, является ли конференция местной или международной (SymfonyLive или SymfonyCon).

Бандл Maker может помочь нам создать класс (класс *сущности*),

который представляет собой конференцию:

```
$ symfony console make:entity Conference
```

Эта команда является интерактивной: она проведёт вас через процесс добавления всех необходимых полей. Используйте следующие ответы (большинство из них являются ответами по умолчанию, так что вы можете нажать клавишу «Enter», чтобы их использовать):

- `city`, `string`, `255`, `no`;
- `year`, `string`, `4`, `no`;
- `isInternational`, `boolean`, `no`.

Полный результат выполнения команды указан ниже:

```
 created: src/Entity/Conference.php
 created: src/Repository/ConferenceRepository.php

 Entity generated! Now let's add some fields!
 You can always add more fields later manually or by re-running this command.

 New property name (press <return> to stop adding fields):
 > city

 Field type (enter ? to see all types) [string]:
 >

 Field length [255]:
 >

 Can this field be null in the database (nullable) (yes/no) [no]:
 >

 updated: src/Entity/Conference.php

 Add another property? Enter the property name (or press <return> to stop
adding fields):
 > year

 Field type (enter ? to see all types) [string]:
 >

 Field length [255]:
 > 4

 Can this field be null in the database (nullable) (yes/no) [no]:
```

```
 >

 updated: src/Entity/Conference.php

 Add another property? Enter the property name (or press <return> to stop
adding fields):
 > isInternational

 Field type (enter ? to see all types) [boolean]:
 >

 Can this field be null in the database (nullable) (yes/no) [no]:
 >

 updated: src/Entity/Conference.php

 Add another property? Enter the property name (or press <return> to stop
adding fields):
 >


  Success!


 Next: When you're ready, create a migration with make:migration
```

Класс `Conference` находится в пространстве имён `App\Entity\`.

Также команда сгенерировала класс *репозитория* для работы с Doctrine: `App\Repository\ConferenceRepository`.

Сгенерированный код выглядит следующим образом (представлена только небольшая часть файла):

src/App/Entity/Conference.php

```php
namespace App\Entity;

use Doctrine\ORM\Mapping as ORM;

/**
 * @ORM\Entity(repositoryClass="App\Repository\ConferenceRepository")
 */
class Conference
{
    /**
     * @ORM\Id()
     * @ORM\GeneratedValue()
```

```
     * @ORM\Column(type="integer")
     */
    private $id;

    /**
     * @ORM\Column(type="string", length=255)
     */
    private $city;

    // ...

    public function getCity(): ?string
    {
        return $this->city;
    }

    public function setCity(string $city): self
    {
        $this->city = $city;

        return $this;
    }

    // ...
}
```

Обратите внимание, что это обычный PHP-класс без элементов Doctrine. Аннотации используются для добавления метаданных, позволяющих Doctrine связать класс сущности с соответствующей таблицей в базе данных.

Doctrine добавила свойство `id` для хранения первичного ключа строки в таблице базы данных. Этот ключ (`@ORM\Id()`) создаётся автоматически (`@ORM\GeneratedValue()`) с помощью стратегии, которая зависит от движка базы данных.

Теперь сгенерируйте класс сущности для комментариев к конференции:

```
$ symfony console make:entity Comment
```

Введите следующие ответы:

- `author`, `string`, `255`, `no`;
- `text`, `text`, `no`;

- email, string, 255, no;
- createdAt, datetime, no.

8.5 Связывание сущностей

Обе сущности, конференция и комментарий, должны быть взаимосвязаны. Конференция может иметь ноль или более комментариев, это называется связью *один-ко-многим*.

Используйте команду make:entity ещё раз, чтобы добавить эту связь в класс Conference:

```
$ symfony console make:entity Conference


 Your entity already exists! So let's add some new fields!

 New property name (press <return> to stop adding fields):
 > comments

 Field type (enter ? to see all types) [string]:
 > OneToMany

 What class should this entity be related to?:
 > Comment

 A new property will also be added to the Comment class...

 New field name inside Comment [conference]:
 >

 Is the Comment.conference property allowed to be null (nullable)? (yes/no)
[yes]:
 > no

 Do you want to activate orphanRemoval on your relationship?
 A Comment is "orphaned" when it is removed from its related Conference.
 e.g. $conference->removeComment($comment)

 NOTE: If a Comment may *change* from one Conference to another, answer "no".

 Do you want to automatically delete orphaned App\Entity\Comment objects
(orphanRemoval)? (yes/no) [no]:
 > yes
```

```
updated: src/Entity/Conference.php
updated: src/Entity/Comment.php
```

Если в качестве ответа на вопрос о типе данных вы введёте ?, то вы получите список всех поддерживаемых типов:

```
Main types
  * string
  * text
  * boolean
  * integer (or smallint, bigint)
  * float

Relationships / Associations
  * relation (a wizard will help you build the relation)
  * ManyToOne
  * OneToMany
  * ManyToMany
  * OneToOne

Array/Object Types
  * array (or simple_array)
  * json
  * object
  * binary
  * blob

Date/Time Types
  * datetime (or datetime_immutable)
  * datetimetz (or datetimetz_immutable)
  * date (or date_immutable)
  * time (or time_immutable)
  * dateinterval

Other Types
  * decimal
  * guid
  * json_array
```

Взгляните на список изменений в классах сущностей после добавления этой связи:

```
--- a/src/Entity/Comment.php
+++ b/src/Entity/Comment.php
@@ -36,6 +36,12 @@ class Comment
      */
     private $createdAt;
```

```
+    /**
+     * @ORM\ManyToOne(targetEntity="App\Entity\Conference",
inversedBy="comments")
+     * @ORM\JoinColumn(nullable=false)
+     */
+    private $conference;
+
     public function getId(): ?int
     {
         return $this->id;
@@ -88,4 +94,16 @@ class Comment

         return $this;
     }
+
+    public function getConference(): ?Conference
+    {
+        return $this->conference;
+    }
+
+    public function setConference(?Conference $conference): self
+    {
+        $this->conference = $conference;
+
+        return $this;
+    }
 }
--- a/src/Entity/Conference.php
+++ b/src/Entity/Conference.php
@@ -2,6 +2,8 @@

 namespace App\Entity;

+use Doctrine\Common\Collections\ArrayCollection;
+use Doctrine\Common\Collections\Collection;
 use Doctrine\ORM\Mapping as ORM;

 /**
@@ -31,6 +33,16 @@ class Conference
      */
     private $isInternational;

+    /**
+     * @ORM\OneToMany(targetEntity="App\Entity\Comment",
mappedBy="conference", orphanRemoval=true)
+     */
+    private $comments;
+
```

```
+    public function __construct()
+    {
+        $this->comments = new ArrayCollection();
+    }
+
     public function getId(): ?int
     {
         return $this->id;
@@ -71,4 +83,35 @@ class Conference

         return $this;
     }
+
+    /**
+     * @return Collection|Comment[]
+     */
+    public function getComments(): Collection
+    {
+        return $this->comments;
+    }
+
+    public function addComment(Comment $comment): self
+    {
+        if (!$this->comments->contains($comment)) {
+            $this->comments[] = $comment;
+            $comment->setConference($this);
+        }
+
+        return $this;
+    }
+
+    public function removeComment(Comment $comment): self
+    {
+        if ($this->comments->contains($comment)) {
+            $this->comments->removeElement($comment);
+            // set the owning side to null (unless already changed)
+            if ($comment->getConference() === $this) {
+                $comment->setConference(null);
+            }
+        }
+
+        return $this;
+    }
 }
```

Всё, что вам нужно для управления связями, было сгенерировано автоматически. Теперь это ваш код, поэтому не стесняйтесь его изменять, как вам нравится.

8.6 Добавление дополнительных свойств

Я только что понял, что мы забыли добавить одно свойство к сущности комментария: участники, возможно, захотят приложить фотографию с конференции, чтобы наглядно проиллюстрировать про свои отзывы.

Выполните команду `make:entity` ещё раз и добавьте свойство/столбец `photoFilename` типа `string` с возможностью иметь значение `null`, так как загрузка фотографии не обязательна:

```
$ symfony console make:entity Comment
```

8.7 Миграция базы данных

Модель проекта теперь полностью описана двумя сгенерированными классами.

Далее нам нужно создать таблицы базы данных, связанные с этими сущностями.

Doctrine Migrations идеально подходит для такой задачи. Этот пакет был установлен ранее в виде зависимости для пакета `orm`.

Миграция — это класс, описывающий изменения, необходимые для обновления схемы базы данных с текущего состояния на новое, определённой в аннотациях сущности. Поскольку на данный момент база данных пуста, миграция должна состоять из создания двух таблиц.

Давайте посмотрим, что сгенерировала Doctrine:

```
$ symfony console make:migration
```

Обратите внимание на сгенерированное имя файла в выводе командной строки (имя, которое выглядит как `src/Migrations/Version20191019083640.php`):

src/Migrations/Version20191019083640.php

```
namespace DoctrineMigrations;

use Doctrine\DBAL\Schema\Schema;
use Doctrine\Migrations\AbstractMigration;

final class Version20191019083640 extends AbstractMigration
{
    public function up(Schema $schema) : void
    {
        // this up() migration is auto-generated, please modify it to your needs
        $this->abortIf($this->connection->getDatabasePlatform()->getName() !==
'postgresql', 'Migration can only be executed safely on \'postgresql\'.');

        $this->addSql('CREATE SEQUENCE comment_id_seq INCREMENT BY 1 MINVALUE 1
START 1');
        $this->addSql('CREATE SEQUENCE conference_id_seq INCREMENT BY 1
MINVALUE 1 START 1');
        $this->addSql('CREATE TABLE comment (id INT NOT NULL, conference_id INT
NOT NULL, author VARCHAR(255) NOT NULL, text TEXT NOT NULL, email VARCHAR(255)
NOT NULL, created_at TIMESTAMP(0) WITHOUT TIME ZONE NOT NULL, photo_filename
VARCHAR(255) DEFAULT NULL, PRIMARY KEY(id))');
        $this->addSql('CREATE INDEX IDX_9474526C604B8382 ON comment
(conference_id)');
        $this->addSql('CREATE TABLE conference (id INT NOT NULL, city
VARCHAR(255) NOT NULL, year VARCHAR(4) NOT NULL, is_international BOOLEAN NOT
NULL, PRIMARY KEY(id))');
        $this->addSql('ALTER TABLE comment ADD CONSTRAINT FK_9474526C604B8382
FOREIGN KEY (conference_id) REFERENCES conference (id) NOT DEFERRABLE INITIALLY
IMMEDIATE');
    }

    public function down(Schema $schema) : void
    {
        // ...
    }
}
```

8.8 Обновление локальной базы данных

Теперь вы можете запустить сгенерированную ранее миграцию для обновления схемы локальной базы данных:

```
$ symfony console doctrine:migrations:migrate
```

Схема локальной базы данных теперь актуальна и подготовлена для хранения данных.

8.9 Обновление базы данных в продакшене

Шаги, необходимые для миграции базы данных на продакшене, такие же, с которыми вы уже знакомы: фиксация изменений и развёртывание.

При развёртывании проекта, SymfonyCloud не только обновляет код, но и выполняет миграцию базы данных, если таковая имеется (это выясняется при помощи команды `doctrine:migrations:migrate`).

Двигаемся дальше

- *Базы данных и Doctrine ORM* в приложениях Symfony;
- *Видеокурс по Doctrine на SymfonyCasts*;
- *Работа с ассоциациями/связями в Doctrine*;
- *Документация DoctrineMigrationsBundle*.

Шаг 9

Создание административной панели

Именно администраторы проекта будут добавлять предстоящие конференции в базу данных. *Административная панель* — это защищённый раздел сайта, где *администраторы проекта* могут изменять данные, модерировать отзывы и многое другое.

Можно быстро сгенерировать панель администрирования на базе модели проекта, используя один из бандлов. EasyAdmin как раз то, что нам нужно.

9.1 Настройка бандла EasyAdmin

Для начала добавьте бандл EasyAdmin в зависимости проекта:

```
$ symfony composer req admin
```

Для настройки EasyAdmin по его Flex-рецепту был создан новый конфигурационный файл:

config/packages/easy_admin.yaml

```
#easy_admin:
#    entities:
#        # List the entity class name you want to manage
#        - App\Entity\Product
#        - App\Entity\Category
#        - App\Entity\User
```

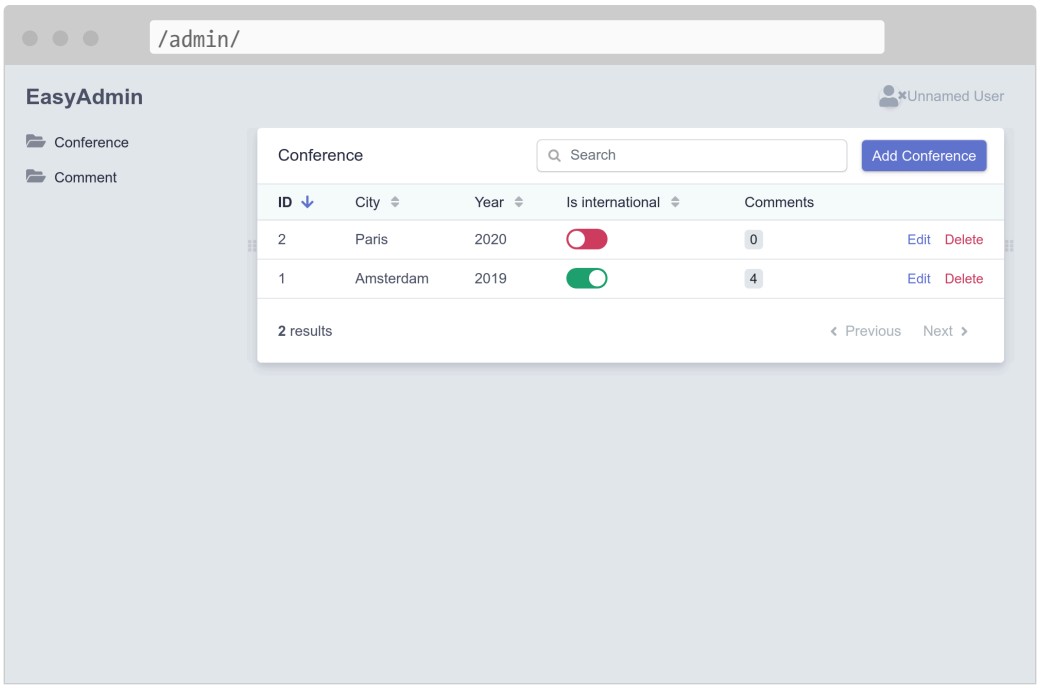
Почти все установленные пакеты имеют подобный файл с конфигурацией в директории `config/packages/`. Чаще всего настройки по умолчанию отлично подходят для большинства приложений.

Раскомментируйте первую пару строк и добавьте классы моделей проекта:

config/packages/easy_admin.yaml

```
easy_admin:
    entities:
        - App\Entity\Conference
        - App\Entity\Comment
```

Перейдите в браузере по пути `/admin` к уже готовой административной панели. И вуаля! У нас уже есть красивый и многофункциональный интерфейс для управления конференциями и комментариями:



Почему административная панель доступна по адресу `/admin`? Всё потому, что использовался префикс по умолчанию, заданный в файле `config/routes/easy_admin.yaml`:

config/routes/easy_admin.yaml

```
easy_admin_bundle:
    resource: '@EasyAdminBundle/Controller/EasyAdminController.php'
    prefix: /admin
    type: annotation
```

Вы можете изменить его на что угодно.

В данный момент невозможно добавить новые конференции и комментарии, поскольку произойдёт ошибка конвертации объекта в строку — `Object of class App\Entity\Conference could not be converted to string`. Бандл EasyAdmin попытается отобразить конференцию с комментариями к ней, но в итоге не сможет этого сделать, потому что нет строкового представления обоих сущностей. Добавление метода `__toString()` в класс `Conference` поможет устранить ошибку:

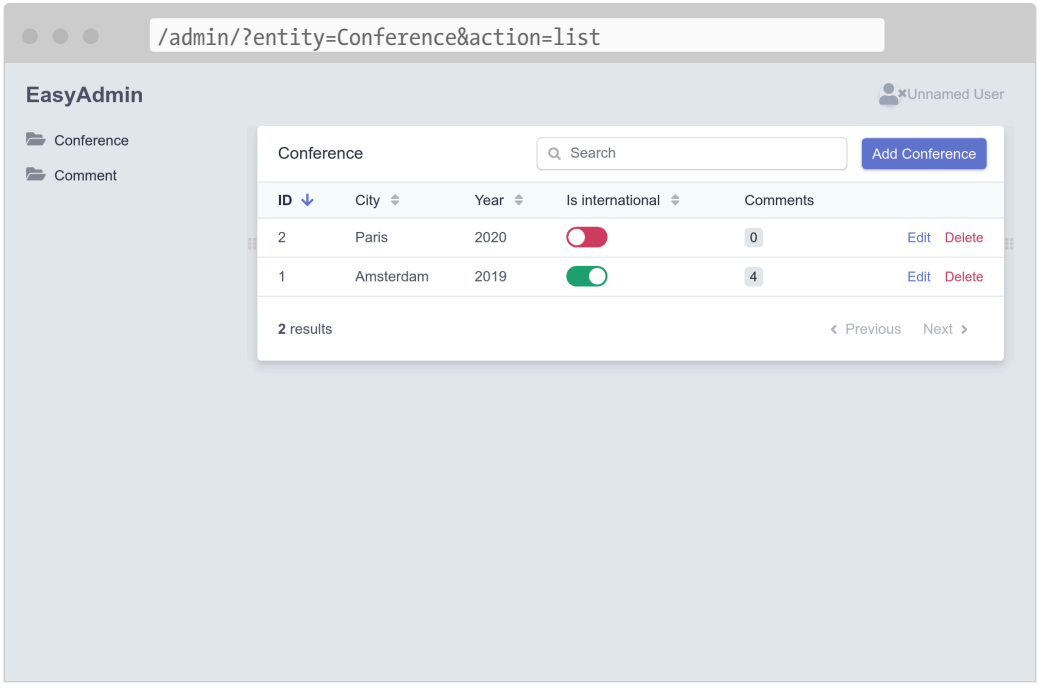
```
--- a/src/Entity/Conference.php
+++ b/src/Entity/Conference.php
@@ -43,6 +43,11 @@ class Conference
         $this->comments = new ArrayCollection();
     }
```

```
+    public function __toString(): string
+    {
+        return $this->city.' '.$this->year;
+    }
+
     public function getId(): ?int
     {
         return $this->id;
```

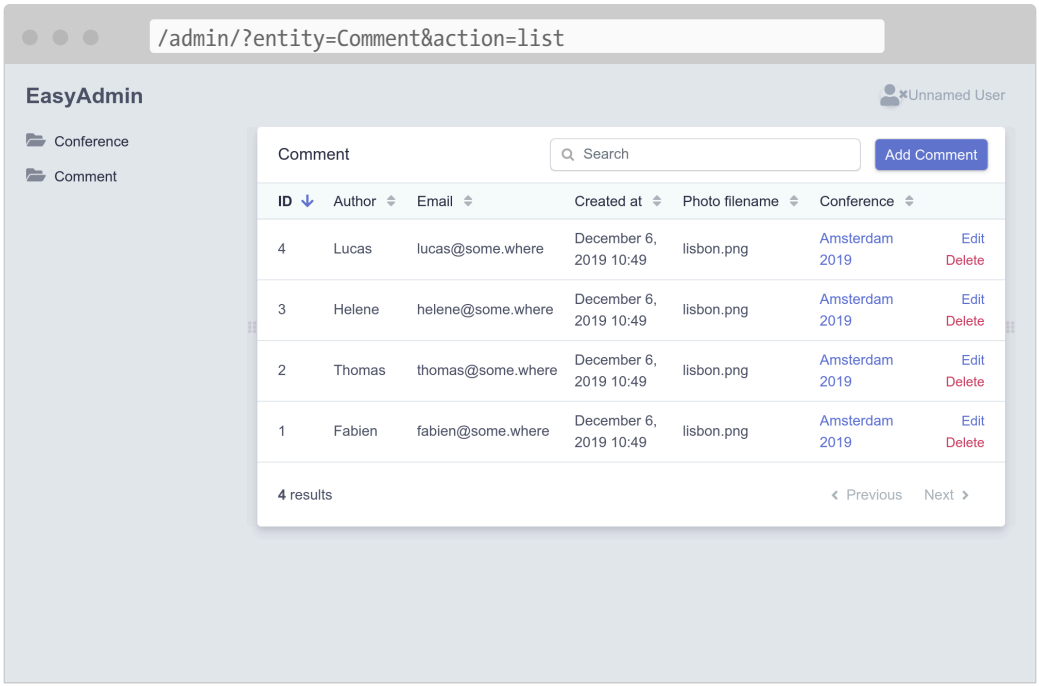
То же самое сделайте в классе `Comment`:

```
--- a/src/Entity/Comment.php
+++ b/src/Entity/Comment.php
@@ -48,6 +48,11 @@ class Comment
      */
     private $photoFilename;

+    public function __toString(): string
+    {
+        return (string) $this->getEmail();
+    }
+
     public function getId(): ?int
     {
         return $this->id;
```

Теперь вы можете добавлять, изменять и удалять конференции непосредственно из административной панели. Изучите его интерфейс и добавьте хотя бы одну конференцию.



Добавьте несколько комментариев без фотографий. Пока установите дату вручную, затем в следующих шагах мы сделаем автозаполнение столбца `createdAt`.



9.2 Настройка EasyAdmin

Административная панель по умолчанию работает хорошо, хотя она

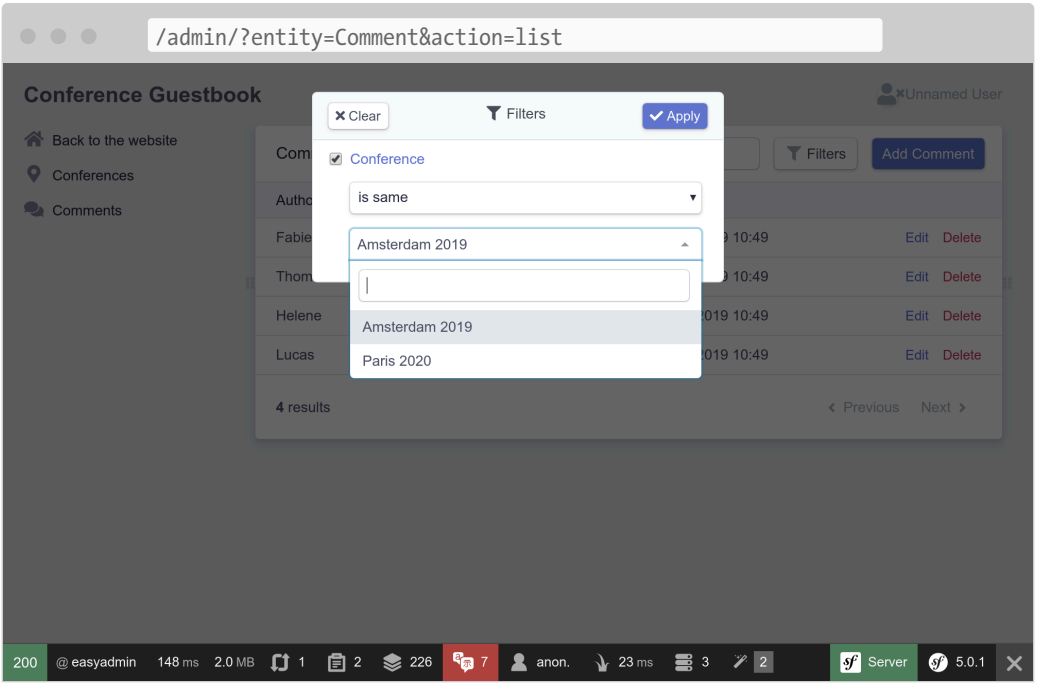
может по-разному настраиваться, чтобы улучшить удобство её использования. Внесем несколько простых изменений для демонстрации доступных вариантов. Заменим стандартную конфигурацию на следующую:

config/packages/easy_admin.yaml

```
easy_admin:
    site_name: Conference Guestbook

    design:
        menu:
            - { route: 'homepage', label: 'Back to the website', icon: 'home' }
            - { entity: 'Conference', label: 'Conferences', icon: 'map-marker' }
            - { entity: 'Comment', label: 'Comments', icon: 'comments' }

    entities:
        Conference:
            class: App\Entity\Conference

        Comment:
            class: App\Entity\Comment
            list:
                fields:
                    - author
                    - { property: 'email', type: 'email' }
                    - { property: 'createdAt', type: 'datetime' }
                sort: ['createdAt', 'ASC']
                filters: ['conference']
            edit:
                fields:
                    - { property: 'conference' }
                    - { property: 'createdAt', type: datetime, type_options: {
attr: { readonly: true } } }
                    - 'author'
                    - { property: 'email', type: 'email' }
                    - text
```

Мы переопределили секцию `design`, чтобы добавить иконки в меню, а также ссылку на главную страницу сайта.

В секции `Comment` перечисление полей позволяет расположить их в нужном нам порядке. Некоторые поля были дополнены дополнительными настройками. К примеру, поле с датой создания доступно только для чтения. Секция `filters` определяет, какие фильтры будут находится рядом с обычным полем поиска.



Это всего лишь небольшая часть возможных настроек в EasyAdmin.

Ознакомьтесь с административной панелью, отфильтруйте комментарии по какой-нибудь конференции или, например, найдите их по адресу электронной почты. Однако есть последняя неразрешённая проблема — любой пользователь может войти в панель администрирования. Мы это обязательно исправим в следующих шагах.

Двигаемся дальше

- *Документация EasyAdmin*;
- *Видеокурс по EasyAdminBundle на SymfonyCasts*;
- *Справочник по конфигурированию Symfony*.

Шаг 10

Создание пользовательского интерфейса

Теперь всё готово для создания первой версии пользовательского интерфейса сайта. Пока мы не будем акцентировать внимание на красивом внешнем виде, а лишь сделаем его просто функциональным.

Помните экранирование, которое нам пришлось добавить в контроллере для пасхального яйца, чтобы избежать проблем с безопасностью? По этой причине мы не будем использовать PHP в наших шаблонах. Вместо него мы воспользуемся Twig. Помимо экранирования вывода, *Twig* предоставляет много полезных возможностей, которые мы будем использовать (например, наследование шаблонов).

10.1 Установка Twig

Нам не нужно добавлять Twig в виде отдельной зависимости, поскольку он уже был установлен как *промежуточная зависимость* от EasyAdmin. Но что если в будущем вы решите перейти на другой бандл для создания административной панели? Например, на тот, который использует API и React на фронтенде. Вероятно, он больше не будет зависеть от Twig, а значит, когда вы удалите EasyAdmin, Twig будет автоматически удалён.

Поэтому для полной уверенности давайте укажем Composer, что проект действительно зависит от Twig, вне зависимости используется ли EasyAdmin. Для этого достаточно добавить его, как и любую другую зависимость:

```
$ symfony composer req twig
```

Теперь Twig является частью основных зависимостей проекта в файле `composer.json`:

```
--- a/composer.json
+++ b/composer.json
@@ -14,6 +14,7 @@
         "symfony/framework-bundle": "4.4.*",
         "symfony/maker-bundle": "^1.0@dev",
         "symfony/orm-pack": "dev-master",
+        "symfony/twig-pack": "^1.0",
         "symfony/yaml": "4.4.*"
     },
     "require-dev": {
```

10.2 Использование Twig для шаблонов

Все страницы сайта будут использовать один и тот же *макет*. При установке Twig автоматически была создана директория `templates/`, вместе с примером макета в файле `base.html.twig`.

templates/base.html.twig

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>{% block title %}Welcome!{% endblock %}</title>
        {% block stylesheets %}{% endblock %}
    </head>
    <body>
        {% block body %}{% endblock %}
        {% block javascripts %}{% endblock %}
    </body>
</html>
```

Макет может определять элементы `block` — специальные области, в которые *дочерние шаблоны*, *расширяющие* макет, будут подставлять своё содержимое.

Создадим шаблон для домашней страницы проекта в файле `templates/conference/index.html.twig`:

templates/conference/index.html.twig

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

{% block title %}Conference Guestbook{% endblock %}

{% block body %}
    <h2>Give your feedback!</h2>

    {% for conference in conferences %}
        <h4>{{ conference }}</h4>
    {% endfor %}
{% endblock %}
```

Шаблон *наследует* `base.html.twig` и переопределяет блоки `title` и `body`.

Разделитель {% %} в шаблоне указывает на *действия* и *структуру* .

Разделитель {{ }} используется для *отображения* чего-либо. Например, {{ conference }} выведет строковое представление конференции (результат вызова метода `__toString` в объекте `Conference`).

10.3 Использование Twig в

контроллере

Обновите контроллер, чтобы отрисовать шаблон Twig:

```
--- a/src/Controller/ConferenceController.php
+++ b/src/Controller/ConferenceController.php
@@ -2,24 +2,21 @@

 namespace App\Controller;

+use App\Repository\ConferenceRepository;
 use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
 use Symfony\Component\HttpFoundation\Response;
 use Symfony\Component\Routing\Annotation\Route;
+use Twig\Environment;

 class ConferenceController extends AbstractController
 {
     /**
      * @Route("/", name="homepage")
      */
-    public function index()
+    public function index(Environment $twig, ConferenceRepository
$conferenceRepository)
     {
-        return new Response(<<<EOF
-<html>
-    <body>
-        <img src="/images/under-construction.gif" />
-    </body>
-</html>
-EOF
-        );
+        return new Response($twig->render('conference/index.html.twig', [
+            'conferences' => $conferenceRepository->findAll(),
+        ]));
     }
 }
```

Здесь много всего интересного.

Для отрисовки шаблона, нам необходим объект Twig —`Environment` (главная точка входа Twig). Обратите внимание, для того чтобы получить экземпляр Twig, достаточно всего лишь указать его тип (так называемый type-hint) в аргументах метода контроллера. Гибкость Symfony позволяет автоматически внедрить зависимость нужного типа.

Нам также нужен репозиторий для конференций, чтобы получить все конференции из базы данных.

Метод `render()` в коде контроллера передаёт массив переменных в шаблон и отрисовывает его. В нашем примере мы передаём список объектов `Conference` в переменной `conferences`.

Контроллер — это обычный PHP-класс. Чтобы использовать зависимости совершенно необязательно наследовать класс `AbstractController`. Вы можете удалить его (но всё же не делайте этого, так как в следующих шагах мы будем использовать некоторые его методы для упрощения).

10.4 Создание страницы для конференции

У каждой конференции должна быть собственная страница с комментариями. Добавление новой страницы включает в себя создание контроллера, определение маршрута (route) и создание соответствующего шаблона.

Добавьте метод `show()` в контроллер `src/Controller/ConferenceController.php`:

```
--- a/src/Controller/ConferenceController.php
+++ b/src/Controller/ConferenceController.php
@@ -2,7 +2,9 @@

 namespace App\Controller;

+use App\Entity\Conference;
+use App\Repository\CommentRepository;
 use App\Repository\ConferenceRepository;
 use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
 use Symfony\Component\HttpFoundation\Response;
 use Symfony\Component\Routing\Annotation\Route;
@@ -19,4 +21,15 @@ class ConferenceController extends AbstractController
             'conferences' => $conferenceRepository->findAll(),
         ]));
     }
+
+    /**
+     * @Route("/conference/{id}", name="conference")
```

```
+     */
+    public function show(Environment $twig, Conference $conference,
CommentRepository $commentRepository)
+    {
+        return new Response($twig->render('conference/show.html.twig', [
+            'conference' => $conference,
+            'comments' => $commentRepository->findBy(['conference' =>
$conference], ['createdAt' => 'DESC']),
+        ]));
+    }
 }
```

Данный метод работает несколько иначе, чем остальные, которые мы встречали ранее. Мы указываем, что экземпляр `Conference` должен быть внедрён в метод. Однако в базе данных могут быть несколько конференций и Symfony может определить, какую из них вы хотите получить по идентификатору `{id}`, переданному в строке запроса (`id` — это первичный ключ в таблице `conference` базы данных).

Получить комментарии конкретной конференции можно с помощью метода `findBy()`, который в качестве первого аргумента принимает критерий (criteria), то есть условие, по которому нужно получить интересующие нас данные.

Последним шагом будет создание файла `templates/conference/show.html.twig`:

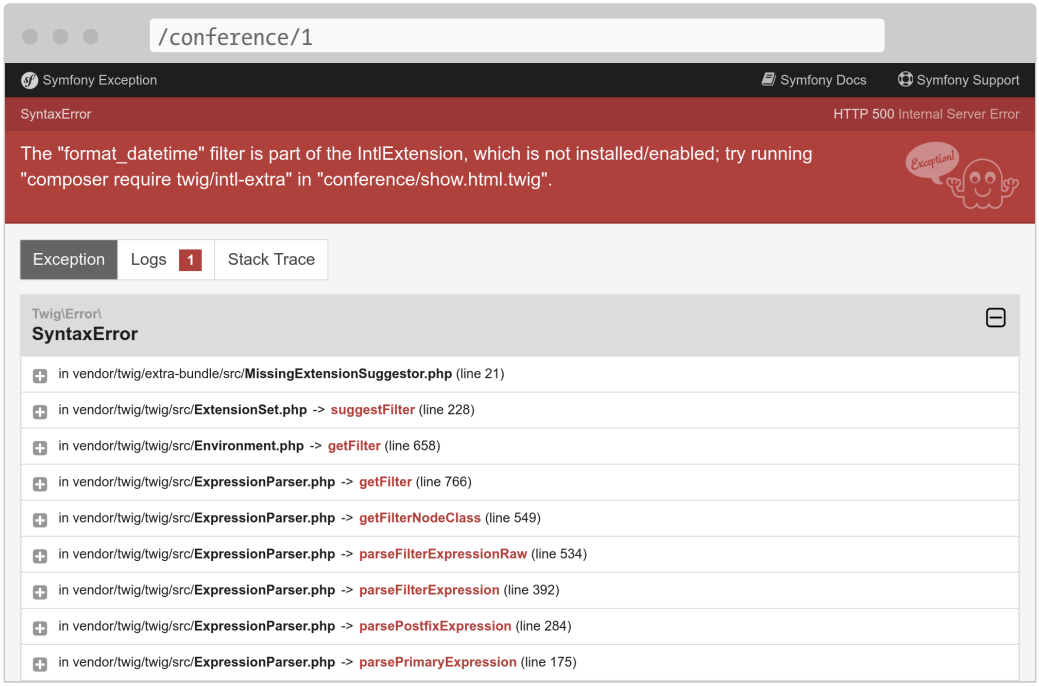
templates/conference/show.html.twig

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

{% block title %}Conference Guestbook - {{ conference }}{% endblock %}

{% block body %}
    <h2>{{ conference }} Conference</h2>

    {% if comments|length > 0 %}
        {% for comment in comments %}
            {% if comment.photofilename %}
                <img src="{{ asset('uploads/photos/' ~ comment.photofilename)
}}" />
            {% endif %}

            <h4>{{ comment.author }}</h4>
            <small>
                {{ comment.createdAt|format_datetime('medium', 'short') }}
            </small>
```

```
            <p>{{ comment.text }}</p>
        {% endfor %}
    {% else %}
        <div>No comments have been posted yet for this conference.</div>
    {% endif %}
{% endblock %}
```

Для вызова *фильтров* Twig в шаблоне используется разделитель `|`, который изменяет значение исходной переменной. Например, `comments|length` отображает количество комментариев, а `comment.createdAt|format_datetime('medium', 'short')` форматирует дату в понятное для чтения представление.

Попробуйте посетить страницу «первой» конференции по пути `/conference/1` и обратите внимание на следующую ошибку:



Эта ошибка из-за использования фильтра `format_datetime`, которого нет в ядре Twig. Подобное сообщение об ошибке подскажет вам, какой пакет должен быть установлен для исправления проблемы:

```
$ symfony composer require twig/intl-extra
```

Теперь страница работает правильно.

10.5 Перелинковка страниц

Остался последний шаг, чтобы закончить первую версию интерфейса — создать ссылки на страницы конференций с главной страницы:

```
--- a/templates/conference/index.html.twig
+++ b/templates/conference/index.html.twig
@@ -7,5 +7,8 @@

     {% for conference in conferences %}
         <h4>{{ conference }}</h4>
+        <p>
+            <a href="/conference/{{ conference.id }}">View</a>
+        </p>
     {% endfor %}
 {% endblock %}
```

Но жёстко заданный путь — плохая идея по ряду причин. Самая главная из них состоит в том, что если вы измените путь (например, с `/conference/{id}` на `/conferences/{id}`), потребуется также изменить и все ссылки вручную.

Вместо этого используйте *Twig-функцию* `path()` и укажите *название маршрута*:

```
--- a/templates/conference/index.html.twig
+++ b/templates/conference/index.html.twig
@@ -8,7 +8,7 @@
     {% for conference in conferences %}
         <h4>{{ conference }}</h4>
         <p>
-            <a href="/conference/{{ conference.id }}">View</a>
+            <a href="{{ path('conference', { id: conference.id }) }}">View</a>
         </p>
     {% endfor %}
 {% endblock %}
```

Функция `path()` генерирует путь к странице, исходя из названия переданного маршрута. Значения параметров маршрута передаются в виде массива.

10.6 Постраничный вывод

комментариев

При тысячах посетителей мы получим довольно много комментариев. Если мы будем отображать все комментарии на одной странице, она станет слишком большой.

В репозитории комментариев создайте метод `getCommentPaginator()`, который будет возвращать объект *Paginator*, в зависимости от конференции и смещении (начальной позиции):

```
--- a/src/Repository/CommentRepository.php
+++ b/src/Repository/CommentRepository.php
@@ -3,8 +3,10 @@
 namespace App\Repository;

 use App\Entity\Comment;
+use App\Entity\Conference;
 use Doctrine\Bundle\DoctrineBundle\Repository\ServiceEntityRepository;
 use Doctrine\Common\Persistence\ManagerRegistry;
+use Doctrine\ORM\Tools\Pagination\Paginator;

 /**
  * @method Comment|null find($id, $lockMode = null, $lockVersion = null)
@@ -14,11 +16,27 @@ use Doctrine\Common\Persistence\ManagerRegistry;
  */
 class CommentRepository extends ServiceEntityRepository
 {
+    public const PAGINATOR_PER_PAGE = 2;
+
     public function __construct(ManagerRegistry $registry)
     {
         parent::__construct($registry, Comment::class);
     }

+    public function getCommentPaginator(Conference $conference, int $offset):
Paginator
+    {
+        $query = $this->createQueryBuilder('c')
+            ->andWhere('c.conference = :conference')
+            ->setParameter('conference', $conference)
+            ->orderBy('c.createdAt', 'DESC')
+            ->setMaxResults(self::PAGINATOR_PER_PAGE)
+            ->setFirstResult($offset)
+            ->getQuery()
+        ;
+
+        return new Paginator($query);
+    }
```

```
+
      // /**
      //  * @return Comment[] Returns an array of Comment objects
      //  */
```

Для облегчения тестирования показываем не более 2 комментариев на каждой странице.

Чтобы управлять постраничной навигацией в шаблоне Twig, передайте объект Doctrine Paginator вместо Doctrine Collection:

```
--- a/src/Controller/ConferenceController.php
+++ b/src/Controller/ConferenceController.php
@@ -6,6 +6,7 @@ use App\Entity\Conference;
 use App\Repository\CommentRepository;
 use App\Repository\ConferenceRepository;
 use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
+use Symfony\Component\HttpFoundation\Request;
 use Symfony\Component\HttpFoundation\Response;
 use Symfony\Component\Routing\Annotation\Route;
 use Twig\Environment;
@@ -25,11 +26,16 @@ class ConferenceController extends AbstractController
      /**
       * @Route("/conference/{id}", name="conference")
       */
-    public function show(Environment $twig, Conference $conference,
CommentRepository $commentRepository)
+    public function show(Request $request, Environment $twig, Conference
$conference, CommentRepository $commentRepository)
      {
+        $offset = max(0, $request->query->getInt('offset', 0));
+        $paginator = $commentRepository->getCommentPaginator($conference,
$offset);
+
          return new Response($twig->render('conference/show.html.twig', [
              'conference' => $conference,
-             'comments' => $commentRepository->findBy(['conference' =>
$conference], ['createdAt' => 'DESC']),
+             'comments' => $paginator,
+             'previous' => $offset - CommentRepository::PAGINATOR_PER_PAGE,
+             'next' => min(count($paginator), $offset +
CommentRepository::PAGINATOR_PER_PAGE),
          ]));
      }
 }
```

Контроллер получает параметр `offset` из строки запроса (`$request->query`) в виде целого числа (`getInt()`), который по умолчанию равен 0,

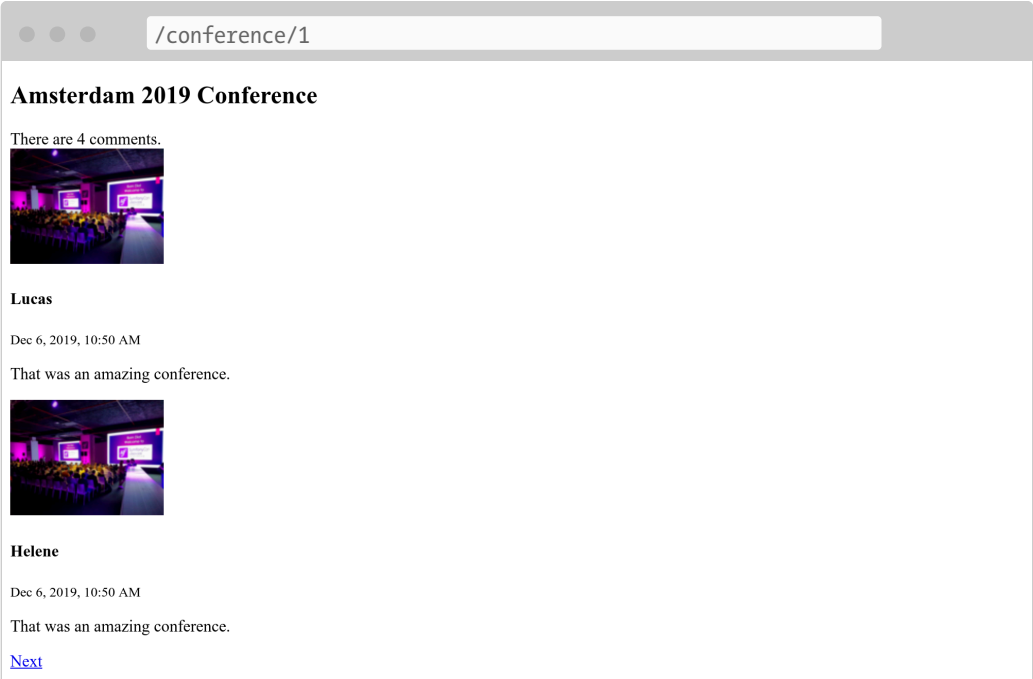
если он отсутствует.

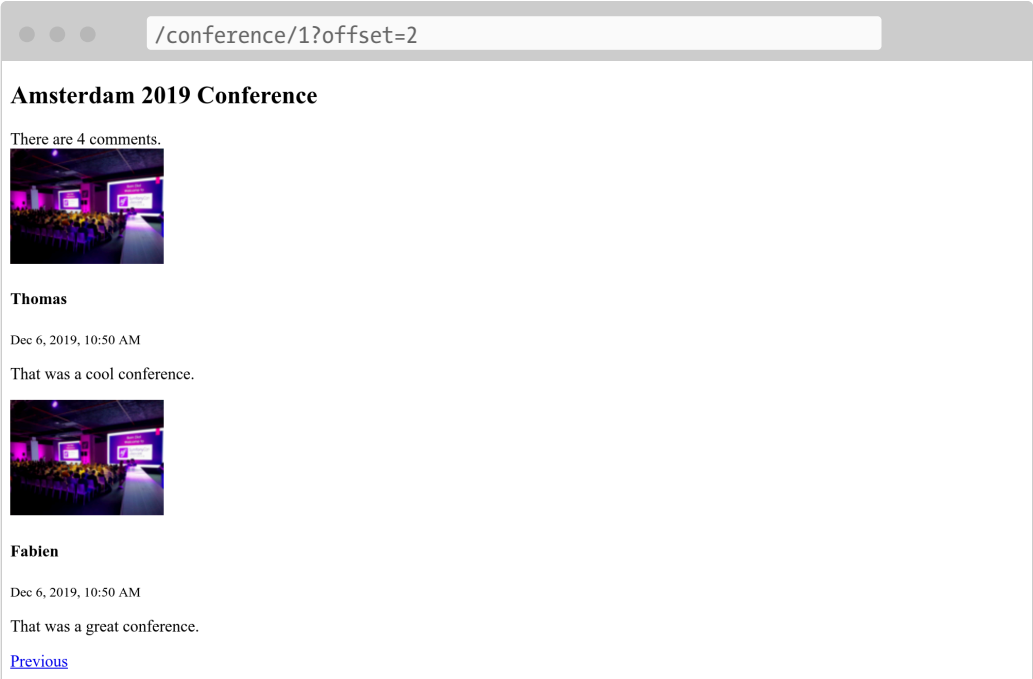
Вычисления смещения `previous` и `next` производятся на основе информации из объекта Paginator.

Наконец, обновите шаблон, чтобы добавить ссылки на следующую и предыдущую страницы:

```
index 0c9e7d2..14b51fd 100644
--- a/templates/conference/show.html.twig
+++ b/templates/conference/show.html.twig
@@ -6,6 +6,8 @@
     <h2>{{ conference }} Conference</h2>

     {% if comments|length > 0 %}
+        <div>There are {{ comments|length }} comments.</div>
+
         {% for comment in comments %}
             {% if comment.photofilename %}
                 <img src="{{ asset('uploads/photos/' ~ comment.photofilename)
}}" />
@@ -18,6 +20,13 @@

             <p>{{ comment.text }}</p>
         {% endfor %}
+
+        {% if previous >= 0 %}
+            <a href="{{ path('conference', { id: conference.id, offset:
previous }) }}">Previous</a>
+        {% endif %}
+        {% if next < comments|length %}
+            <a href="{{ path('conference', { id: conference.id, offset: next
}) }}">Next</a>
+        {% endif %}
     {% else %}
         <div>No comments have been posted yet for this conference.</div>
     {% endif %}
```

Теперь вы сможете перемещаться по комментариям с помощью ссылок «Previous» и «Next»:





10.7 Рефакторинг контроллера

Вероятно, вы заметили, что оба метода в контроллере `ConferenceController` используют окружение Twig в качестве аргумента. Однако вместо того чтобы внедрять эту зависимость в каждый метод, достаточно добавить её лишь в конструктор. Этим мы сократим список аргументов и сделаем его менее избыточным:

```
--- a/src/Controller/ConferenceController.php
+++ b/src/Controller/ConferenceController.php
@@ -13,12 +13,19 @@ use Twig\Environment;

 class ConferenceController extends AbstractController
 {
+    private $twig;
+
+    public function __construct(Environment $twig)
+    {
+        $this->twig = $twig;
+    }
+
     /**
      * @Route("/", name="homepage")
      */
-    public function index(Environment $twig, ConferenceRepository
$conferenceRepository)
+    public function index(ConferenceRepository $conferenceRepository)
     {
-        return new Response($twig->render('conference/index.html.twig', [
+        return new Response($this->twig->render('conference/index.html.twig', [
             'conferences' => $conferenceRepository->findAll(),
         ]));
     }
@@ -26,12 +33,12 @@ class ConferenceController extends AbstractController
     /**
      * @Route("/conference/{id}", name="conference")
      */
-    public function show(Request $request, Environment $twig, Conference
$conference, CommentRepository $commentRepository)
+    public function show(Request $request, Conference $conference,
CommentRepository $commentRepository)
     {
         $offset = max(0, $request->query->getInt('offset', 0));
         $paginator = $commentRepository->getCommentPaginator($conference,
$offset);

-        return new Response($twig->render('conference/show.html.twig', [
+        return new Response($this->twig->render('conference/show.html.twig', [
             'conference' => $conference,
             'comments' => $paginator,
             'previous' => $offset - CommentRepository::PAGINATOR_PER_PAGE,
```

Двигаемся дальше

- *Документация по Twig*;
- *Создание и использование шаблонов* в Symfony-приложениях;
- *Учебный видеоролик по Twig на SymfonyCasts*;
- *Функции и фильтры Twig доступные только в Symfony*;
- *Базовый контроллер AbstractController*.

Шаг 11
Работа с ветками

Существует множество способов организации работы с кодом в проекте. Однако фиксировать изменения непосредственно в master-ветке Git и напрямую развёртывать код в продакшен без тестирования, пожалуй, не лучший вариант.

Тестирование — это не только модульные или функциональные тесты, но и проверка поведения приложения с использованием реальных данных. Если вы или ваши *заинтересованные стороны* можете просмотреть приложение в том самом виде, в котором оно предстанет перед конечными пользователями, это становится огромным преимуществом и позволяет вам развёртывать приложения с уверенностью. Особенно полезно, когда нетехнические специалисты могут проверять новую функциональность.

Для упрощения и чтобы избежать повторения на следующих шагах мы продолжим работу в master-ветке Git, однако давайте посмотрим, как это можно организовать получше.

11.1 Организация рабочего процесса с помощью Git

Создание отдельной ветки на каждую новую функциональность или исправление бага — один из простых и эффективных вариантов организации рабочего процесса.

11.2 Описание инфраструктуры

Возможно, вы пока ещё не осознали это, но хранение инфраструктурных файлов рядом с кодом, очень удобно. Docker и SymfonyCloud используют конфигурационные файлы для описания инфраструктуры проекта. Когда для работы новой функциональности необходим дополнительный сервис, изменения в коде и инфраструктуре находятся в одном патче.

11.3 Создание веток

Рабочий процесс начинается с создания ветки в Git:

```
$ git checkout -b sessions-in-redis
```

Команда из примера создаёт ветку `sessions-in-redis` из ветки `master`. Это действие создаёт ответвление кода и конфигурации инфраструктуры.

11.4 Хранение сессий в Redis

Вероятно, вы уже поняли из названия ветки, что для хранения сессий вместо файловой системы мы будем использовать хранилище в Redis.

Необходимые шаги для реализации этого вполне типичны:

1. Создайте новую ветку в Git;

2. Обновите конфигурацию Symfony, если потребуется;
3. При необходимости напишите и/или обновите код;
4. Обновите конфигурацию PHP, добавив PHP-модуль Redis;
5. Обновите инфраструктуру Docker и SymfonyCloud, добавив сервис Redis;
6. Протестируйте локально;
7. Протестируйте удалённо;
8. Выполните слияние ветки с основной веткой;
9. Разверните в продакшене;
10. Удалите ветку.

Все изменения, выполненные с 2 по 5 шаг, можно применить одним патчем:

```
--- a/.symfony.cloud.yaml
+++ b/.symfony.cloud.yaml
@@ -4,6 +4,7 @@ type: php:7.3

 runtime:
     extensions:
+        - redis
         - pdo_pgsql
         - apcu
         - mbstring
@@ -14,6 +15,7 @@ build:

 relationships:
     database: "db:postgresql"
+    redis: "rediscache:redis"

 web:
     locations:
--- a/.symfony/services.yaml
+++ b/.symfony/services.yaml
@@ -2,3 +2,6 @@ db:
     type: postgresql:11
     disk: 1024
     size: S
+
+rediscache:
+    type: redis:5.0
```

```
--- a/config/packages/framework.yaml
+++ b/config/packages/framework.yaml
@@ -6,7 +6,7 @@ framework:
     # Enables session support. Note that the session will ONLY be started if
you read or write from it.
     # Remove or comment this section to explicitly disable session support.
     session:
-        handler_id: null
+        handler_id: '%env(REDIS_URL)%'
         cookie_secure: auto
         cookie_samesite: lax

--- a/docker-compose.yaml
+++ b/docker-compose.yaml
@@ -8,3 +8,7 @@ services:
             POSTGRES_PASSWORD: main
             POSTGRES_DB: main
         ports: [5432]
+
+    redis:
+        image: redis:5-alpine
+        ports: [6379]
```

Разве это не *замечательно*?

«Перезагрузите» Docker, чтобы запустить сервис Redis:

```
$ docker-compose stop
$ docker-compose up -d
```

Протестируйте сайт на своём компьютере, просматривая различные страницы. Поскольку мы не вносили никаких визуальных изменений и пока не используем сессии, всё должно работать как и раньше.

11.5 Развёртывание ветки

Перед развёртыванием в продакшене, мы должны протестировать ветку в окружении ему идентичному. Нам необходимо убедиться, что всё работает корректно в Symfony-окружении `prod` (локальный сайт использует окружение `dev`).

Для начала зафиксируйте изменения в новой ветке:

```
$ git add .
$ git commit -m'Configure redis sessions'
```

Теперь давайте создадим *SymfonyCloud-окружение* на основе *Git-ветки*:

```
$ symfony env:create
```

Данная команда создаёт новое окружение в следующем порядке:

- Ветка наследует код и инфраструктуру от текущей Git ветки (`sessions-in-redis`);
- Данные поступают из основного окружения (т.е. продакшена) путём создания последовательных слепков всех служебных данных, включая файлы (например, загруженные пользователями) и базы данных;
- Создаётся новый выделенный кластер для развёртывания кода, данных и инфраструктуры.

Поскольку развёртывание проходит те же этапы, что и развёртывание в продакшене, миграции базы данных также будут выполнены. Это отличный способ проверить, что миграции работают с данными в продакшене.

Окружения, отличные от `master`, на самом деле очень на него похожи, но есть небольшие отличия. Например, отправка электронных писем отключена по умолчанию.

После завершения развёртывания откройте новую ветку в браузере:

```
$ symfony open:remote
```

Обратите внимание, что все команды SymfonyCloud работают в текущей Git-ветке. Выполненная выше команда откроет URL-адрес для только что развёрнутой ветки `sessions-in-redis`. Адрес будет выглядеть примерно так: `https://sessions-in-redis-xxx.eu.s5y.io/`.

Протестируйте сайт в новом окружении: вы увидите те же данные, что и в основном окружении.

Если добавить новые конференции в основное окружение (`master`), они не появятся в окружении `sessions-in-redis` и наоборот, так как

окружения являются независимыми и полностью изолированными друг от друга.

Если код изменится в основной ветке, вы всегда можете выполнить перебазирование Git-ветки и развернуть обновлённую версию, разрешив конфликты в коде и инфраструктуре.

В том числе вы можете синхронизировать данные с основного окружения в окружение `sessions-in-redis`:

```
$ symfony env:sync
```

11.6 Предварительная отладка развёртывания в продакшене

По умолчанию все SymfonyCloud-окружения используют те же настройки, что и окружение `master`/`prod` (то же, что и окружение `prod` в Symfony). Это позволяет протестировать приложение в реальных условиях (как и в продакшене). Таким образом создаётся ощущение разработки и тестирования непосредственно на продакшен-серверах, но без связанных с этим рисков. Мне напоминает это старые добрые времена, когда мы развёртывали проекты через FTP.

При возникновении проблемы вы можете переключиться на Symfony-окружение `dev`:

```
$ symfony env:debug
```

Как только закончите, переключитесь обратно к продакшен-настройкам:

```
$ symfony env:debug --off
```

Никогда не активируйте окружение `dev` и никогда не используйте профилировщик Symfony, находясь в ветке `master`, это приведёт к тому, что ваше приложение станет очень медленным и откроет множество серьёзных дыр безопасности.

11.7 Предварительное тестирование развёртывания в продакшене

Возможность предварительного просмотра будущей версии сайта с продакшен-данными открывает широкие возможности: от визуального регрессионного тестирования до тестирования производительности. *Blackfire* — идеальный инструмент для такого рода задач.

Перейдите к шагу «Производительность», чтобы узнать больше про использование Blackfire для тестирования кода перед развёртыванием.

11.8 Развёртывание в продакшене

Если вы полностью удовлетворены изменениями в ветке с новой функциональностью, выполните слияние кода и инфраструктуры в master-ветку в Git:

```
$ git checkout master
$ git merge sessions-in-redis
```

И разверните:

```
$ symfony deploy
```

При развёртывании в SymfonyCloud отправляются только изменения в коде и инфраструктуре; данные останутся такими же, как и были до развёртывания.

11.9 Очистка

В завершение выполните очистку, удалив Git-ветку и SymfonyCloud-окружение:

```
$ git branch -d sessions-in-redis
$ symfony env:delete --env=sessions-in-redis --no-interaction
```

Двигаемся дальше

- *Ветвление в Git*;
- *Документация по Redis*.

Шаг 12
Обработка событий

В текущем макете отсутствует шапка с навигационным меню, с помощью которой можно было бы вернуться на главную страницу или перейти на следующую конференцию.

12.1 Добавление шапки сайта

Всё, что необходимо отображать на всех страницах, например, шапка сайта, должно быть частью основного макета:

```
--- a/templates/base.html.twig
+++ b/templates/base.html.twig
@@ -6,6 +6,15 @@
         {% block stylesheets %}{% endblock %}
     </head>
     <body>
+        <header>
+            <h1><a href="{{ path('homepage') }}">Guestbook</a></h1>
+            <ul>
+            {% for conference in conferences %}
+                <li><a href="{{ path('conference', { id: conference.id })
}}">{{ conference }}</a></li>
```

```
+            {% endfor %}
+            </ul>
+            <hr />
+        </header>
         {% block body %}{% endblock %}
         {% block javascripts %}{% endblock %}
     </body>
```

Добавление этого кода в макет означает, что все шаблоны, расширяющие его, должны определить переменную `conferences`, которую необходимо создать и передать из соответствующих контроллеров.

Поскольку у нас всего два контроллера, вы можете сделать следующее:

```
--- a/src/Controller/ConferenceController.php
+++ b/src/Controller/ConferenceController.php
@@ -32,9 +32,10 @@ class ConferenceController extends AbstractController
      /**
       * @Route("/conference/{slug}", name="conference")
       */
-    public function show(Conference $conference, CommentRepository
$commentRepository)
+    public function show(Conference $conference, CommentRepository
$commentRepository, ConferenceRepository $conferenceRepository)
     {
         return new Response($this->twig->render('conference/show.html.twig', [
+            'conferences' => $conferenceRepository->findAll(),
             'conference' => $conference,
             'comments' => $commentRepository->findBy(['conference' =>
$conference], ['createdAt' => 'DESC']),
         ]));
```

Представьте себе, что нам необходимо обновить десятки контроллеров. И с каждым новым контроллером мы будем вынуждены делать то же самое. Это не очень практично. Должен быть способ лучше.

У Twig есть понятие глобальных переменных. *Глобальная переменная* доступна во всех отрисованных шаблонах. Вы можете определить их в конфигурационном файле, но там можно задать только статические значения. Чтобы добавить все конференции в глобальную переменную Twig, мы создадим обработчик событий.

12.2 Исследуем события Symfony

В Symfony есть встроенный компонент диспетчера событий (Event Dispatcher). Диспетчер *отправляет* определённые *события* в конкретные моменты времени, а *обработчики* обрабатывают эти события. Обработчики событий позволяют взаимодействовать с внутренностями фреймворка, то есть выполнять некоторый код, в ответ на события, генерируемые в фреймворке.

Например, есть события, которые позволяют взаимодействовать с жизненным циклом HTTP-запросов. При обработке запроса диспетчер генерирует события в следующих случаях: когда создан объект запроса, когда должен быть выполнен контроллер, когда готов HTTP-ответ, либо когда выброшено исключение. *Обработчик* может реагировать на одно или несколько разных событий, и выполнять некоторую логику в зависимости от контекста события.

События — это то, что помогает фреймворку быть более гибким и многофункциональным. Многие компоненты Symfony, такие как Security, Messenger, Workflow или Mailer, активно используют их.

Другим примером встроенных событий и обработчиков в действии является жизненный цикл команды: вы можете создать обработчик, код которого будет выполняться перед запуском *любой* команды.

Любой пакет или бандл может также отправлять свои собственные события, чтобы сделать код расширяемым.

Вместо использования конфигурационного файла, описывающего, на какие события должен реагировать обработчик, создайте *подписчика*. Подписчик — это обработчик со статическим методом `getSubscribedEvents()`, который возвращает свою конфигурацию. Это позволяет автоматически регистрировать подписчиков в диспетчере Symfony.

12.3 Реализация подписчика

Теперь, когда вы всё знаете, используйте бандл maker, чтобы сгенерировать подписчика:

```
$ symfony console make:subscriber TwigEventSubscriber
```

Команда спросит вас о том, какое событие вы хотите обрабатывать. Выберите событие `Symfony\Component\HttpKernel\Event\ControllerEvent`, отправляемое непосредственно перед вызовом контроллера. Самое время для определения глобальной переменной `conferences`, чтобы Twig имел к ней доступ во время отрисовки шаблона контроллером. Обновите вашего подписчика следующим образом:

```
--- a/src/EventSubscriber/TwigEventSubscriber.php
+++ b/src/EventSubscriber/TwigEventSubscriber.php
@@ -2,14 +2,25 @@

 namespace App\EventSubscriber;

+use App\Repository\ConferenceRepository;
 use Symfony\Component\EventDispatcher\EventSubscriberInterface;
 use Symfony\Component\HttpKernel\Event\ControllerEvent;
+use Twig\Environment;

 class TwigEventSubscriber implements EventSubscriberInterface
 {
+    private $twig;
+    private $conferenceRepository;
+
+    public function __construct(Environment $twig, ConferenceRepository
$conferenceRepository)
+    {
+        $this->twig = $twig;
+        $this->conferenceRepository = $conferenceRepository;
+    }
+
     public function onControllerEvent(ControllerEvent $event)
     {
-        // ...
+        $this->twig->addGlobal('conferences', $this->conferenceRepository-
>findAll());
     }

     public static function getSubscribedEvents()
```

Теперь вы можете добавить сколько угодно контроллеров: переменная `conferences` всегда будет доступна в Twig.

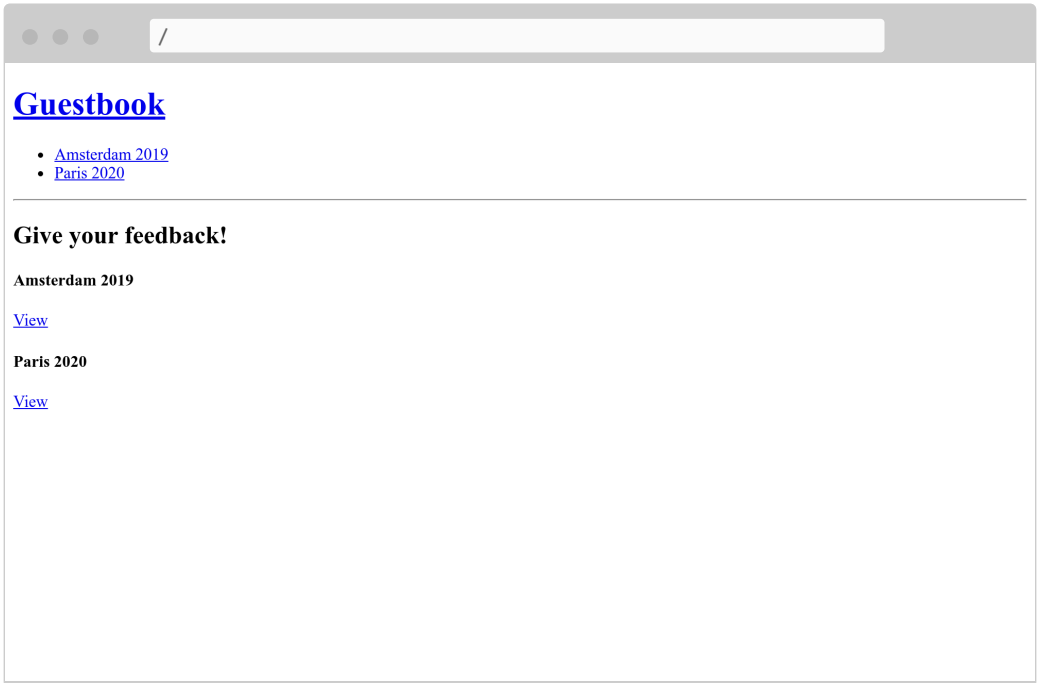
Более производительную альтернативу мы рассмотрим позднее.

12.4 Сортировка конференций по годам и городам

Представление списка конференций по годам позволит упростить просмотр. Для получения и сортировки всех конференций мы могли бы создать отдельный метод, однако вместо этого переопределим стандартную реализацию метода `findAll()`, чтобы данная сортировка применялась всегда по умолчанию:

```
--- a/src/Repository/ConferenceRepository.php
+++ b/src/Repository/ConferenceRepository.php
@@ -19,6 +19,11 @@ class ConferenceRepository extends ServiceEntityRepository
         parent::__construct($registry, Conference::class);
     }

+    public function findAll()
+    {
+        return $this->findBy([], ['year' => 'ASC', 'city' => 'ASC']);
+    }
+
     // /**
     //  * @return Conference[] Returns an array of Conference objects
     //  */
```

В конце этого шага сайт должен выглядеть следующим образом:



Двигаемся дальше

- *Рабочий процесс запроса-ответа* в приложениях Symfony;
- *Встроенные HTTP-события в Symfony*;
- *Встроенные события Symfony Console*.

Шаг 13

Жизненный цикл объектов Doctrine

Было бы неплохо, если при создании нового комментария значение поля `createdAt` автоматически заполнялось текущими датой и временем.

Doctrine может по-разному манипулировать объектами и их свойствами в различных стадиях жизненного цикла (до вставки записи в базу данных, после обновления записи и т.д.).

13.1 Определение обратных вызовов событий жизненного цикла

Если логика не требует доступа к сервису и применяется только к одному типу сущности, можно определить обратный вызов в классе сущности:

```
--- a/src/Entity/Comment.php
+++ b/src/Entity/Comment.php
@@ -6,6 +6,7 @@ use Doctrine\ORM\Mapping as ORM;

 /**
  * @ORM\Entity(repositoryClass="App\Repository\CommentRepository")
+ * @ORM\HasLifecycleCallbacks()
  */
 class Comment
 {
@@ -100,6 +101,14 @@ class Comment
         return $this;
     }

+    /**
+     * @ORM\PrePersist
+     */
+    public function setCreatedAtValue()
+    {
+        $this->createdAt = new \DateTime();
+    }
+
     public function getConference(): ?Conference
     {
         return $this->conference;
```

Событие `@ORM\PrePersist` срабатывает, когда объект впервые сохранятся в базе данных. В этот момент вызывается метод `setCreatedAtValue()`, который использует текущие дату и время в качестве значения для свойства `createdAt`.

13.2 Добавление слагов для конференций

Сейчас URL-адреса конференций вроде `/conference/1` не очень понятны. Более того, они раскрывают детали реализации приложения (значение первичного ключа 1 доступно пользователю).

Почему бы не использовать URL-адреса вида `/conference/paris-2020`? Они выглядят намного лучше и красивее. Фрагмент адреса `paris-2020` — это *слаг* (человеко-понятная часть URL-адреса) конференции.

Добавьте новое свойство `slug` в класс конференции (строка длиной до

255 символов, которая не может пустой):

```
$ symfony console make:entity Conference
```

Создайте файл миграции, чтобы добавить новый столбец:

```
$ symfony console make:migration
```

А затем выполните новую миграцию:

```
$ symfony console doctrine:migrations:migrate
```

Увидели ошибку? Это было ожидаемо, потому что мы указали, что свойство `slug` не должно быть пустым (содержать значение `null`). Но дело в том, что во время выполнения миграции как раз существующие записи в базе данных конференций будут иметь значение `null`. Давайте исправим это изменив миграцию:

```
--- a/src/Migrations/Version00000000000000.php
+++ b/src/Migrations/Version00000000000000.php
@@ -22,7 +22,9 @@ final class Version00000000000000 extends AbstractMigration
         // this up() migration is auto-generated, please modify it to your
needs
         $this->abortIf($this->connection->getDatabasePlatform()->getName() !==
'postgresql', 'Migration can only be executed safely on \'postgresql\'.');

-        $this->addSql('ALTER TABLE conference ADD slug VARCHAR(255) NOT NULL');
+        $this->addSql('ALTER TABLE conference ADD slug VARCHAR(255)');
+        $this->addSql("UPDATE conference SET slug=CONCAT(LOWER(city), '-',
year)");
+        $this->addSql('ALTER TABLE conference ALTER COLUMN slug SET NOT NULL');
     }

     public function down(Schema $schema) : void
```

Мы применили некоторую хитрость: сначала добавляем столбец слага с возможностью иметь значение по умолчанию —`null`, далее создаём слаг для существующих записей (то есть заполняем новый столбец значениями, отличными от `null`), а затем изменяем столбец слага так, чтобы он он не позволял хранить значение `null`.

В реальном проекте использование выражения `CONCAT(LOWER(city), '-', year)` может быть недостаточным. В таком случае понадобится использовать «настоящий» сервис для генерации слага (слагер).

Теперь миграция должна выполниться без ошибок:

```
$ symfony console doctrine:migrations:migrate
```

Поскольку приложение вскоре будет использовать слаги для поиска каждой конференции, давайте улучшим сущность Conference, чтобы гарантировать уникальность значений слагов в базе данных:

```
--- a/src/Entity/Conference.php
+++ b/src/Entity/Conference.php
@@ -5,9 +5,11 @@ namespace App\Entity;
 use Doctrine\Common\Collections\ArrayCollection;
 use Doctrine\Common\Collections\Collection;
 use Doctrine\ORM\Mapping as ORM;
+use Symfony\Bridge\Doctrine\Validator\Constraints\UniqueEntity;

 /**
  * @ORM\Entity(repositoryClass="App\Repository\ConferenceRepository")
+ * @UniqueEntity("slug")
  */
 class Conference
 {
@@ -39,7 +41,7 @@ class Conference
     private $comments;

     /**
-     * @ORM\Column(type="string", length=255)
+     * @ORM\Column(type="string", length=255, unique=true)
      */
     private $slug;
```

Как вы могли догадаться, нам нужно выполнить процедуру миграции:

```
$ symfony console make:migration


$ symfony console doctrine:migrations:migrate
```

13.3 Генерация слагов

Во многих языках создать слаг, который хорошо читается в URL-адресе (где должно быть закодировано все, кроме ASCII-символов), не так-то просто. К примеру, как поменять *é‘* на e?

Чтобы не изобретать велосипед, давайте воспользуемся Symfony-компонентом `String`, который не только облегчает работу со строками, но и содержит *слагер*:

```
$ symfony composer req string
```

В класс `Conference` добавьте метод `computeSlug()`, который исходя из данных конференции сгенерирует слаг:

```
--- a/src/Entity/Conference.php
+++ b/src/Entity/Conference.php
@@ -6,6 +6,7 @@ use Doctrine\Common\Collections\ArrayCollection;
 use Doctrine\Common\Collections\Collection;
 use Doctrine\ORM\Mapping as ORM;
 use Symfony\Bridge\Doctrine\Validator\Constraints\UniqueEntity;
+use Symfony\Component\String\Slugger\SluggerInterface;

 /**
  * @ORM\Entity(repositoryClass="App\Repository\ConferenceRepository")
@@ -60,6 +61,13 @@ class Conference
         return $this->id;
     }

+    public function computeSlug(SluggerInterface $slugger)
+    {
+        if (!$this->slug || '-' === $this->slug) {
+            $this->slug = (string) $slugger->slug((string) $this)->lower();
+        }
+    }
+
     public function getCity(): ?string
     {
         return $this->city;
```

Метод `computeSlug()` генерирует слаг только в том случае, если значение слага отсутствует, либо указанный слаг имеет специальное значение `-`. Но зачем оно нужно? Поскольку слаг не может быть пустым, при добавлении конференции в административной панели нам нужно указать некое специальное значение (в нашем случае — `-`) в

соответствующем поле, чтобы сообщить приложению, что оно должно автоматически сгенерировать слаг.

13.4 Определение сложных обратных вызовов жизненного цикла

По аналогии со свойством `createdAt`, свойство `slug` должно автоматически обновляться при каждом изменении конференции путем вызова метода `computeSlug()`.

Но так как метод зависит от реализации `SluggerInterface`, мы не можем добавить событие `prePersist` так, как делали это раньше (нет способа внедрить слагер).

Вместо этого создайте обработчик сущности Doctrine:

src/EntityListener/ConferenceEntityListener.php

```
namespace App\EntityListener;

use App\Entity\Conference;
use Doctrine\ORM\Event\LifecycleEventArgs;
use Symfony\Component\String\Slugger\SluggerInterface;

class ConferenceEntityListener
{
    private $slugger;

    public function __construct(SluggerInterface $slugger)
    {
        $this->slugger = $slugger;
    }

    public function prePersist(Conference $conference, LifecycleEventArgs
$event)
    {
        $conference->computeSlug($this->slugger);
    }

    public function preUpdate(Conference $conference, LifecycleEventArgs $event)
    {
        $conference->computeSlug($this->slugger);
    }
```

```
}
```

Обратите внимание, что слаг генерируется как при создании новой конференции (`prePersist()`), так и при её обновлении (`preUpdate()`).

13.5 Настройка сервиса в контейнере

До сих пор мы не упомянули один из главных компонентов Symfony — контейнер внедрения зависимостей, который управляет *сервисами*: создаёт и внедряет их по мере необходимости.

Сервис — это «глобальный» объект с определённой функциональностью (mailer — отправка электронных писем, logger — логирование, slugger — генерация URL-адресов, и т.д.) в отличие от *объектов данных* (к примеру, экземпляров сущности Doctrine).

Вы редко будете работать с контейнером напрямую, поскольку он автоматически внедряет сервисы, когда это вам необходимо: внедрение объектов-сервисов происходит, когда вы указываете типы соответствующих сервисов в качестве аргументов контроллера.

Теперь вы знаете, что обработчик события в предыдущем примере был зарегистрирован через контейнер. Когда класс реализует определённые интерфейсы, контейнер знает, что класс должен быть зарегистрирован соответствующим образом.

К сожалению, не всегда такой тип автоматизации возможен, особенно в сторонних пакетах. Одним из таких примеров является только что нами написанный обработчик сущности; он не может автоматически управляться сервисом контейнеров Symfony, так как не реализует ни одного интерфейса и не наследуется от уже «известного» контейнеру класса.

Нам нужно частично объявить обработчик в контейнере. Связывание зависимостей можно пропустить, так как это может быть выполнено контейнером самостоятельно, но требуется вручную добавить пару *тегов*, чтобы зарегистрировать обработчик в диспетчере событий Doctrine:

```
--- a/config/services.yaml
+++ b/config/services.yaml
@@ -25,3 +25,7 @@ services:

     # add more service definitions when explicit configuration is needed
     # please note that last definitions always *replace* previous ones
+    App\EntityListener\ConferenceEntityListener:
+        tags:
+            - { name: 'doctrine.orm.entity_listener', event: 'prePersist',
entity: 'App\Entity\Conference'}
+            - { name: 'doctrine.orm.entity_listener', event: 'preUpdate',
entity: 'App\Entity\Conference'}
```

Не путайте обработчики событий Doctrine и обработчики событий Symfony. Даже если они очень похожи, они по-разному работают изнутри.

13.6 Использование слагов в приложении

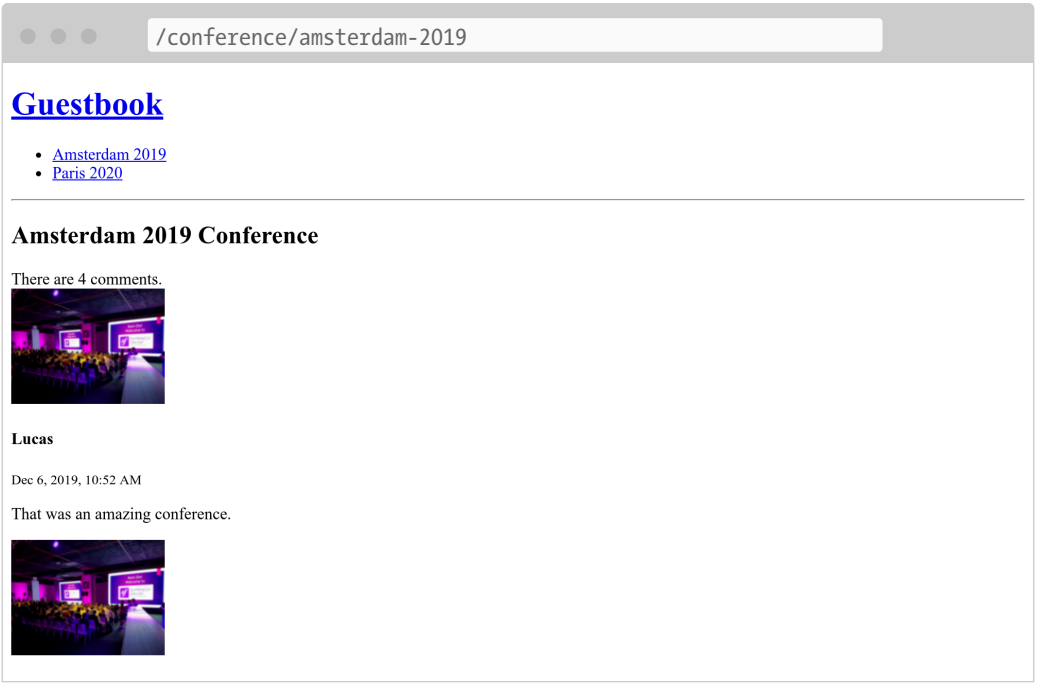
Попробуйте добавить несколько конференций в административной панели, либо измените город или год проведения уже созданных конференций; слаг не обновится, только если вы не укажете в его поле специальное значение — -.

Осталось сделать последнее изменение — заменить в контроллерах и шаблонах параметр маршрутов конференций с `id` на `slug`:

```
--- a/src/Controller/ConferenceController.php
+++ b/src/Controller/ConferenceController.php
@@ -31,7 +31,7 @@ class ConferenceController extends AbstractController
     }

     /**
-     * @Route("/conference/{id}", name="conference")
+     * @Route("/conference/{slug}", name="conference")
      */
     public function show(Request $request, Conference $conference,
CommentRepository $commentRepository)
     {
--- a/templates/base.html.twig
```

```
+++ b/templates/base.html.twig
@@ -10,7 +10,7 @@
             <h1><a href="{{ path('homepage') }}">Guestbook</a></h1>
             <ul>
             {% for conference in conferences %}
-                <li><a href="{{ path('conference', { id: conference.id })
}}">{{ conference }}</a></li>
+                <li><a href="{{ path('conference', { slug: conference.slug })
}}">{{ conference }}</a></li>
             {% endfor %}
             </ul>
             <hr />
--- a/templates/conference/show.html.twig
+++ b/templates/conference/show.html.twig
@@ -22,10 +22,10 @@
         {% endfor %}

         {% if previous >= 0 %}
-            <a href="{{ path('conference', { id: conference.id, offset:
previous }) }}">Previous</a>
+            <a href="{{ path('conference', { slug: conference.slug, offset:
previous }) }}">Previous</a>
         {% endif %}
         {% if next < comments|length %}
-            <a href="{{ path('conference', { id: conference.id, offset: next
}) }}">Next</a>
+            <a href="{{ path('conference', { slug: conference.slug, offset:
next }) }}">Next</a>
         {% endif %}
     {% else %}
         <div>No comments have been posted yet for this conference.</div>
--- a/templates/conference/index.html.twig
+++ b/templates/conference/index.html.twig
@@ -8,7 +8,7 @@
     {% for conference in conferences %}
         <h4>{{ conference }}</h4>
         <p>
-            <a href="{{ path('conference', { id: conference.id }) }}">View</a>
+            <a href="{{ path('conference', { slug: conference.slug })
}}">View</a>
         </p>
     {% endfor %}
 {% endblock %}
```

Теперь можно перейти к странице конференции через её слаг:



Двигаемся дальше

- *Система событий Doctrine* (обратные вызовы и обработчики событий жизненного цикла, обработчики сущностей и подписчики жизненного цикла);
- *Документация компонента String*;
- *Сервис-контейнер*;
- *Шпаргалка по сервисам Symfony*.

Шаг 14

Получение обратной связи с помощью форм

Пришло время добавить возможность получения обратной связи от наших посетителей. Они смогут оставлять комментарии с помощью *HTML-формы*.

14.1 Создание типа формы

Используйте бандл Maker для создания класса формы:

```
$ symfony console make:form CommentFormType Comment
```

```
created: src/Form/CommentFormType.php
```

```
  Success!


 Next: Add fields to your form and start using it.
 Find the documentation at https://symfony.com/doc/current/forms.html
```

Класс `App\Form\CommentFormType` определяет форму для сущности `App\Entity\Comment`:

src/App/Form/CommentFormType.php

```php
namespace App\Form;

use App\Entity\Comment;
use Symfony\Component\Form\AbstractType;
use Symfony\Component\Form\FormBuilderInterface;
use Symfony\Component\OptionsResolver\OptionsResolver;

class CommentFormType extends AbstractType
{
    public function buildForm(FormBuilderInterface $builder, array $options)
    {
        $builder
            ->add('author')
            ->add('text')
            ->add('email')
            ->add('createdAt')
            ->add('photoFilename')
            ->add('conference')
        ;
    }

    public function configureOptions(OptionsResolver $resolver)
    {
        $resolver->setDefaults([
            'data_class' => Comment::class,
        ]);
    }
}
```

Тип формы задаёт *поля формы*, связанные с моделью. Он выполняет преобразование между отправленными данными и свойствами класса модели. По умолчанию для определения конфигурации каждого поля, Symfony использует метаданные (например, метаданные Doctrine) сущности `Comment`. К примеру, поле `text` будет отрисовано как `textarea`, так как в базе данных используется столбец для хранения текста, а не строки.

14.2 Отображение формы

Для отображения формы, создайте её в контроллере и передайте в шаблон:

```
--- a/src/Controller/ConferenceController.php
+++ b/src/Controller/ConferenceController.php
@@ -2,7 +2,9 @@

 namespace App\Controller;

+use App\Entity\Comment;
 use App\Entity\Conference;
+use App\Form\CommentFormType;
 use App\Repository\CommentRepository;
 use App\Repository\ConferenceRepository;
 use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
@@ -35,6 +37,9 @@ class ConferenceController extends AbstractController
      */
     public function show(Request $request, Conference $conference,
CommentRepository $commentRepository)
     {
+        $comment = new Comment();
+        $form = $this->createForm(CommentFormType::class, $comment);
+
         $offset = max(0, $request->query->getInt('offset', 0));
         $paginator = $commentRepository->getCommentPaginator($conference,
$offset);

@@ -43,6 +48,7 @@ class ConferenceController extends AbstractController
             'comments' => $paginator,
             'previous' => $offset - CommentRepository::PAGINATOR_PER_PAGE,
             'next' => min(count($paginator), $offset +
CommentRepository::PAGINATOR_PER_PAGE),
+            'comment_form' => $form->createView(),
         ]));
     }
 }
```

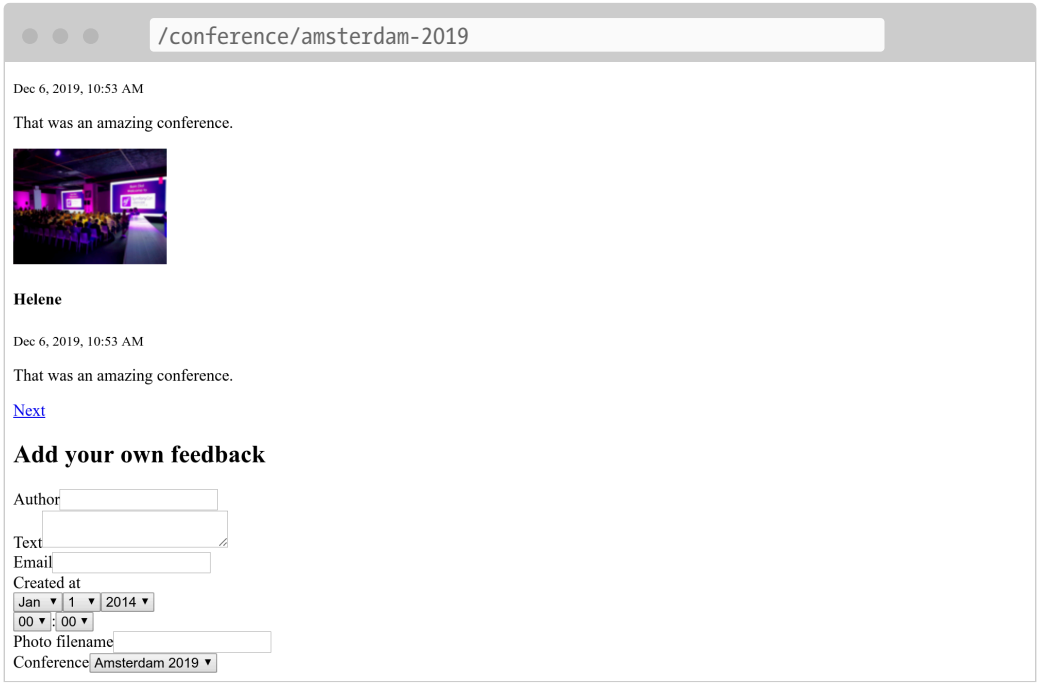
Никогда не следует инициализировать класс формы напрямую. Для упрощения создания форм, используйте метод `createForm()` класса `AbstractController`.

Чтобы передать форму в шаблон, используйте метод `createView()`, который преобразует данные в подходящий для использования в шаблонах формат.

Отобразить форму в шаблоне можно с помощь Twig-функции `form`:

```
--- a/templates/conference/show.html.twig
+++ b/templates/conference/show.html.twig
@@ -21,4 +21,8 @@
     {% else %}
         <div>No comments have been posted yet for this conference.</div>
     {% endif %}
+
+    <h2>Add your own feedback</h2>
+
+    {{ form(comment_form) }}
 {% endblock %}
```

После обновления страницы конференции в браузере, обратите внимание, что каждое поле формы использует HTML-элемент, соответствующий типу модели:



Функция `form()` создаёт HTML-форму, используя всю информацию, определённую в типе формы. В том числе эта функция добавляет обязательный для поля загрузки файла атрибут `enctype=multipart/form-data` к тегу `<form>`. Более того, эта функция также берёт на себя отображение сообщений об ошибках после отправки формы. Переписав шаблоны по умолчанию можно изменить абсолютно всё, однако для нашего проекта это не понадобится.

14.3 Настройка типа формы

Несмотря на то, что поля формы конфигурируются по их типу в модели, вы можете изменить стандартную конфигурацию непосредственно в классе типа формы:

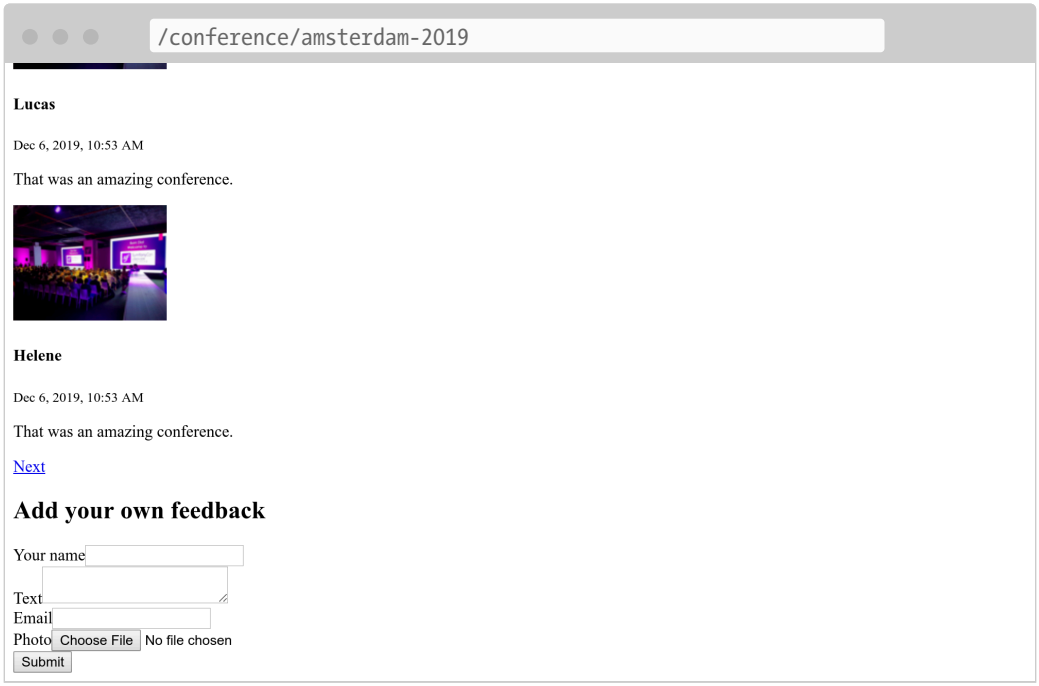
```
--- a/src/Form/CommentFormType.php
+++ b/src/Form/CommentFormType.php
@@ -4,20 +4,31 @@ namespace App\Form;

 use App\Entity\Comment;
 use Symfony\Component\Form\AbstractType;
+use Symfony\Component\Form\Extension\Core\Type\EmailType;
+use Symfony\Component\Form\Extension\Core\Type\FileType;
+use Symfony\Component\Form\Extension\Core\Type\SubmitType;
 use Symfony\Component\Form\FormBuilderInterface;
 use Symfony\Component\OptionsResolver\OptionsResolver;
+use Symfony\Component\Validator\Constraints\Image;

 class CommentFormType extends AbstractType
 {
     public function buildForm(FormBuilderInterface $builder, array $options)
     {
         $builder
-            ->add('author')
+            ->add('author', null, [
+                'label' => 'Your name',
+            ])
             ->add('text')
-            ->add('email')
-            ->add('createdAt')
-            ->add('photoFilename')
-            ->add('conference')
+            ->add('email', EmailType::class)
+            ->add('photo', FileType::class, [
+                'required' => false,
+                'mapped' => false,
+                'constraints' => [
+                    new Image(['maxSize' => '1024k'])
+                ],
+            ])
+            ->add('submit', SubmitType::class)
         ;
     }
```

Обратите внимание, что мы добавили кнопку отправки. Это позволит нам продолжить использовать простое выражение {{

`form(comment_form) }}` в шаблоне.

Некоторые поля не могут быть автоматически сконфигурированы, например, `photoFilename` — одно из них. Сущность `Comment` должна только сохранять имя файла с фотографией, форма в свою очередь берёт на себя обработку загрузки файла. Для этого мы добавили поле `photo` с отключённой опцией `mapped`, таким образом это поле не будет связано ни с одним свойством в сущности `Comment`. Мы будем управлять этим полем вручную, чтобы реализовать определённую логику (например, сохранять загруженную фотографию на диск).

В качестве примера настройки, у некоторых полей мы также изменили метки по умолчанию.



14.4 Валидация моделей

Тип формы определяет её внешний вид (используя валидацию HTML5). Пример сгенерированной HTML-формы:

```
<form name="comment_form" method="post" enctype="multipart/form-data">
    <div id="comment_form">
        <div >
            <label for="comment_form_author" class="required">Your name</label>
            <input type="text" id="comment_form_author"
name="comment_form[author]" required="required" maxlength="255" />
```

```
        </div>
        <div >
            <label for="comment_form_text" class="required">Text</label>
            <textarea id="comment_form_text" name="comment_form[text]"
required="required"></textarea>
        </div>
        <div >
            <label for="comment_form_email" class="required">Email</label>
            <input type="email" id="comment_form_email"
name="comment_form[email]" required="required" />
        </div>
        <div >
            <label for="comment_form_photo">Photo</label>
            <input type="file" id="comment_form_photo"
name="comment_form[photo]" />
        </div>
        <div >
            <button type="submit" id="comment_form_submit"
name="comment_form[submit]">Submit</button>
        </div>
        <input type="hidden" id="comment_form__token"
name="comment_form[_token]" value="DwqsEanxc48jofxsqbGBVLQBqlVJ_Tg4u9-BL1Hjgac"
/>
    </div>
</form>
```

Наша форма содержит поле для электронного адреса (атрибут типа со значением `email`). Кроме этого, большинство полей обязательны для заполнения (имеют атрибут `required`). Обратите внимание, что форма также включает в себя скрытое поле `_token`, используемое для защиты от *CSRF-атак*.

Если при отправке формы HTML-валидация не срабатывает, то на сервер могут попасть некорректные данные. Например, это может случиться при использовании HTTP-клиентов, таких как cURL, игнорирующих правила валидации.

Нам также необходимо добавить некоторые правила валидации для модели `Comment`:

```
--- a/src/Entity/Comment.php
+++ b/src/Entity/Comment.php
@@ -3,6 +3,7 @@
 namespace App\Entity;

 use Doctrine\ORM\Mapping as ORM;
```

```
+use Symfony\Component\Validator\Constraints as Assert;

 /**
  * @ORM\Entity(repositoryClass="App\Repository\CommentRepository")
@@ -19,16 +20,20 @@ class Comment

     /**
      * @ORM\Column(type="string", length=255)
+     * @Assert\NotBlank
      */
     private $author;

     /**
      * @ORM\Column(type="text")
+     * @Assert\NotBlank
      */
     private $text;

     /**
      * @ORM\Column(type="string", length=255)
+     * @Assert\NotBlank
+     * @Assert\Email
      */
     private $email;
```

14.5 Обработка формы

Мы написали достаточно кода, чтобы отобразить форму.

Теперь в контроллере нам необходимо обработать отправку формы и сохранить данные из неё в базе данных:

```
--- a/src/Controller/ConferenceController.php
+++ b/src/Controller/ConferenceController.php
@@ -7,6 +7,7 @@ use App\Entity\Conference;
 use App\Form\CommentFormType;
 use App\Repository\CommentRepository;
 use App\Repository\ConferenceRepository;
+use Doctrine\ORM\EntityManagerInterface;
 use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
 use Symfony\Component\HttpFoundation\Request;
 use Symfony\Component\HttpFoundation\Response;
@@ -16,10 +17,12 @@ use Twig\Environment;
 class ConferenceController extends AbstractController
 {
```

```
     private $twig;
+    private $entityManager;

-    public function __construct(Environment $twig)
+    public function __construct(Environment $twig, EntityManagerInterface
$entityManager)
     {
         $this->twig = $twig;
+        $this->entityManager = $entityManager;
     }

     /**
@@ -39,6 +42,15 @@ class ConferenceController extends AbstractController
     {
         $comment = new Comment();
         $form = $this->createForm(CommentFormType::class, $comment);
+        $form->handleRequest($request);
+        if ($form->isSubmitted() && $form->isValid()) {
+            $comment->setConference($conference);
+
+            $this->entityManager->persist($comment);
+            $this->entityManager->flush();
+
+            return $this->redirectToRoute('conference', ['slug' => $conference-
>getSlug()]);
+        }

         $offset = max(0, $request->query->getInt('offset', 0));
         $paginator = $commentRepository->getCommentPaginator($conference,
$offset);
```

После отправки формы объект `Comment` будет обновлён в соответствии с полученными данными.

Конференция должна быть такой же, как указано в URL-адресе (мы удалили его из формы).

В случае, если данные заполненной формы некорректные, мы по-прежнему перенаправляем пользователя на страницу конференции, однако теперь форма будет заполнена введёнными ранее значениями, а также отобразит, какие ошибки были допущены при её заполнении.

Теперь попробуйте заполнить и отправить форму. Всё должно отработать правильно и данные сохраниться в базе данных (проверьте в административной панели). Тем не менее, есть одна проблема — фотографии. Они не сохраняются, так как мы ещё не реализовали обработку их в контроллере.

14.6 Загрузка файлов

Чтобы мы могли отображать загруженные фотографии на странице конференции, нам нужно сохранять их в публичной директории. Поэтому мы будем хранить фотографии в директории `public/uploads/photos`:

```
--- a/src/Controller/ConferenceController.php
+++ b/src/Controller/ConferenceController.php
@@ -10,6 +10,7 @@ use App\Repository\ConferenceRepository;
 use Doctrine\ORM\EntityManagerInterface;
 use Doctrine\ORM\Tools\Pagination\Paginator;
 use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
+use Symfony\Component\HttpFoundation\File\Exception\FileException;
 use Symfony\Component\HttpFoundation\Request;
 use Symfony\Component\HttpFoundation\Response;
 use Symfony\Component\Routing\Annotation\Route;
@@ -37,7 +38,7 @@ class ConferenceController extends AbstractController
     /**
      * @Route("/conference/{slug}", name="conference")
      */
-    public function show(Request $request, Conference $conference, CommentRepository $commentRepository)
+    public function show(Request $request, Conference $conference, CommentRepository $commentRepository, string $photoDir)
     {
         $comment = new Comment();
         $form = $this->createForm(CommentFormType::class, $comment);
@@ -45,6 +46,15 @@ class ConferenceController extends AbstractController
         $form->handleRequest($request);
         if ($form->isSubmitted() && $form->isValid()) {
             $comment->setConference($conference);
+            if ($photo = $form['photo']->getData()) {
+                $filename = bin2hex(random_bytes(6)).'.'.$photo->guessExtension();
+                try {
+                    $photo->move($photoDir, $filename);
+                } catch (FileException $e) {
+                    // unable to upload the photo, give up
+                }
+                $comment->setPhotoFilename($filename);
+            }

             $this->entityManager->persist($comment);
             $this->entityManager->flush();
```

Для загруженного файла фотографии мы сначала сгенерируем

случайное имя. Затем переместим этот файл в выбранную ранее специальную директорию для хранения фотографий. И наконец, мы сохраним имя файла в объекте Comment.

Вы заметили новый аргумент в методе `show()`? В данном случае аргумент `$photoDir` — это обычная строка, а не сервис. Но как тогда Symfony поймёт, что мы хотим получить в таком аргументе? Контейнер Symfony помимо сервисов может также хранить *параметры*. Параметры — это скалярные значения, которые помогают настраивать различные сервисы. Их можно явно внедрять в сервисы, либо они могут быть *привязаны по имени*:

```
--- a/config/services.yaml
+++ b/config/services.yaml
@@ -10,6 +10,8 @@ services:
     _defaults:
         autowire: true      # Automatically injects dependencies in your
services.
         autoconfigure: true # Automatically registers your services as
commands, event subscribers, etc.
+        bind:
+            $photoDir: "%kernel.project_dir%/public/uploads/photos"

     # makes classes in src/ available to be used as services
     # this creates a service per class whose id is the fully-qualified class
name
```

Свойство `bind` позволяет Symfony внедрять соответствующее значение каждый раз, когда в сервисе есть аргумент `$photoDir`.

Попробуйте загрузить PDF-файл вместо фотографии. Вы увидите сообщения об ошибках в действии. Сейчас они выглядят ужасно, но пока не обращайте внимание на это — мы сделаем всё красиво в последующих шагах, когда будем работать над дизайном сайта. Для стилизации элементов форм нам достаточно поменять одну строчку в конфигурации.

14.7 Отладка форм

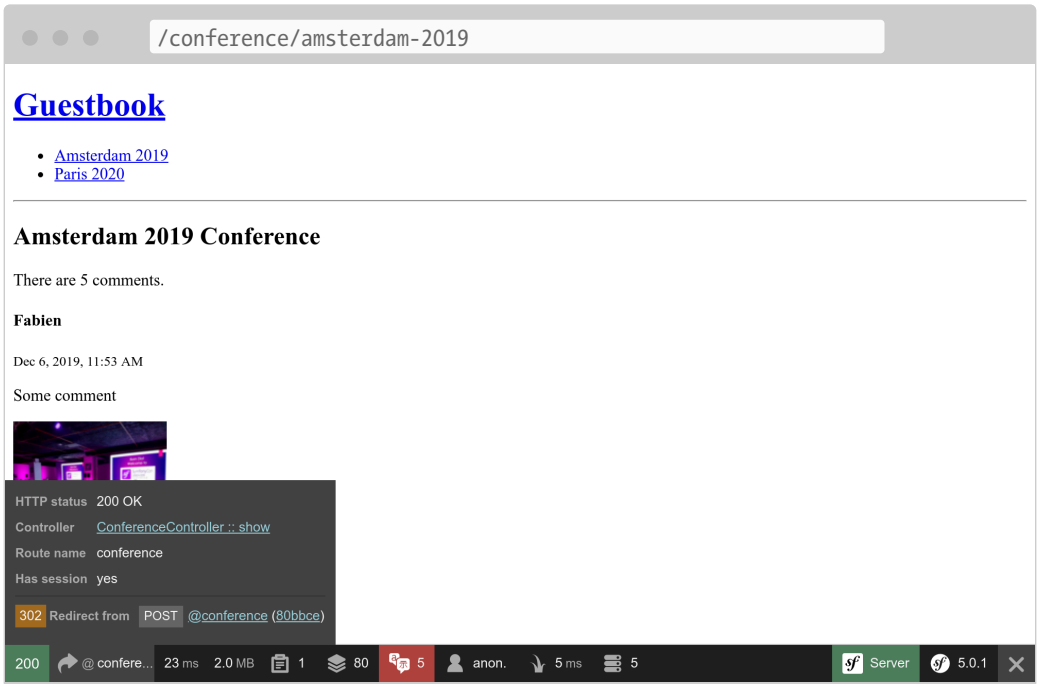
После отправки формы, если что-то не работает как нужно, используйте панель «Form» в профилировщике Symfony. На данной странице вы сможете увидеть информацию о форме, всех её параметрах, отправленные данные, включая то, как они были

преобразованы. Если форма содержит ошибки, они так же будут показаны.

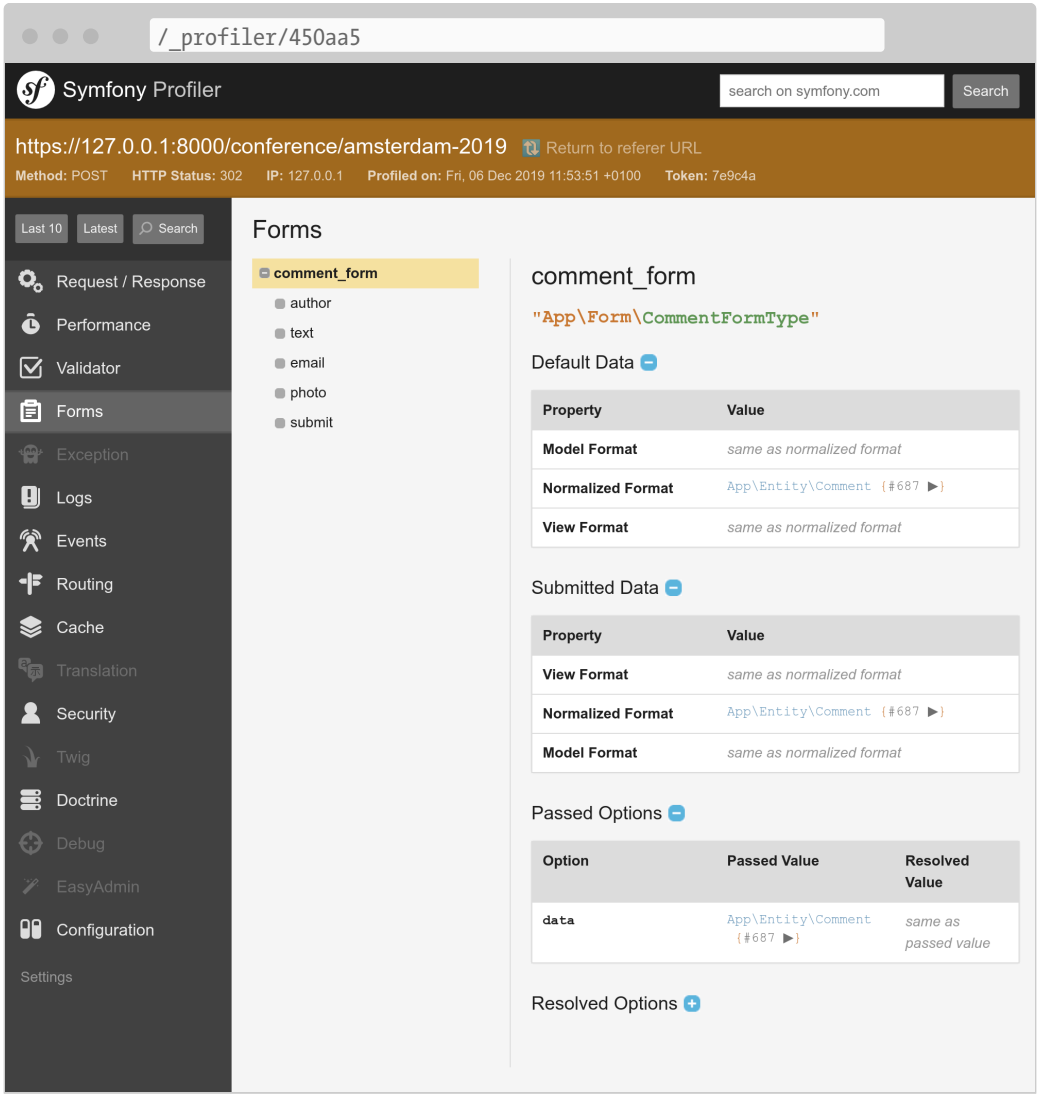
Порядок взаимодействия с формой, как правило, выглядит следующим образом:

- Форма отображается на странице;
- Пользователь отправляет форму через POST-запрос;
- Сервер перенаправляет пользователя на другую или ту же самую страницу, на которой находится форма.

Но как нам использовать профилировщик в случае успешной отправки формы? Из-за перенаправления мы никогда не увидим отладочную панель после отправки POST-запроса. Не беда — на перенаправленной странице в панели отладки наведите на зелёную область с надписью «200». Вы увидите, что запрос был перенаправлен (об этом указывает надпись «302»), а рядом есть ссылка на профилировщик (в скобках).



Перейдите по ссылке, чтобы открыть профилировщик этого POST-запроса, затем перейдите на панель «Form».



14.8 Отображение загруженных фотографий в административной панели

На данный момент административная панель показывает всего лишь название файла фотографии, хотя мы ожидаем увидеть само изображение:

```
--- a/config/packages/easy_admin.yaml
+++ b/config/packages/easy_admin.yaml
@@ -8,6 +8,7 @@ easy_admin:
                fields:
                    - author
```

```
                    - { property: 'email', type: 'email' }
+                   - { property: 'photoFilename', type: 'image', 'base_path':
"/uploads/photos", label: 'Photo' }
                    - { property: 'createdAt', type: 'datetime' }
            edit:
                fields:
```

14.9 Игнорирование загруженных фотографий в Git

Не торопитесь фиксировать изменения! Чтобы загруженные файлы фотографий не попали в Git-репозиторий, добавьте директорию `/public/uploads` в файл `.gitignore`:

```
--- a/.gitignore
+++ b/.gitignore
@@ -1,3 +1,4 @@
+/public/uploads

 ###> symfony/framework-bundle ###
 /.env.local
```

14.10 Хранение загруженных файлов в продакшене

На последнем шаге мы рассмотрим возможность хранения загруженных файлов на продакшен-серверах. Зачем нам нужно что-то делать для этого? Всё из-за того, что большинство современных облачных платформ по ряду причин используют контейнеры только для чтения. SymfonyCloud — не исключение.

В Symfony-проекте далеко не всё может быть только для чтения. Мы усердно старались закешировать как можно больше данных при сборке контейнера (во время прогрева кеша), однако Symfony всё ещё нужно сохранять файлы пользовательского кеша, логов, сессий (если они хранятся в файловой системе) и т.д.

Откройте файл `.symfony.cloud.yaml`. В нем уже есть точка *монтирования* с возможностью записи, указанная для директории `var/`. Директория `var/` — единственное место в файловой системе, в которую Symfony может что-то записывать (кеш, логи и т.п.).

Создадим новую точку монтирования для загруженных фотографий:

```
--- a/.symfony.cloud.yaml
+++ b/.symfony.cloud.yaml
@@ -26,6 +26,7 @@ disk: 512

 mounts:
     "/var": { source: local, source_path: var }
+    "/public/uploads": { source: local, source_path: uploads }

 hooks:
     build: |
```

Теперь вы можете развернуть код и убедиться в том, что файлы фотографий сохраняются в директории `public/uploads/`, так же как и в локальной версии.

Двигаемся дальше

- *Видеокурс по формам на SymfonyCast*;
- *Как настроить отображение Symfony-форм в HTML*;
- *Валидация Symfony-форм*;
- *Справочник по типам Symfony-форм*;
- *Документация по FlysystemBundle*, с помощью которого можно организовать хранение файлов на специальных облачных хостингах, таких как AWS S3, Azure и Google Cloud Storage;
- *Параметры конфигурации Symfony*.
- *Правила Symfony-валидации*;
- *Шпаргалка по Symfony-формам*.

Шаг 15

Защита административной панели

Административная панель должна быть доступна только доверенным лицам. Защитить её можно с помощью компонента Symfony Security.

Как и Twig, компонент безопасности был уже установлен через промежуточные зависимости. Тем не менее, укажем его явно в файле `composer.json`:

```
$ symfony composer req security
```

15.1 Определение сущности для пользователей

Несмотря на то, что посетители не смогут создавать учётные записи на

сайте самостоятельно, мы создадим полнофункциональную систему аутентификации для администратора. Поэтому у нас будет только один пользователь — администратор сайта.

На первом шаге давайте определим сущность `User`. Чтобы избежать возможной путаницы, назовём её `Admin`.

Для интеграции сущности `Admin` с системой аутентификации Symfony Security, она должна соответствовать определённым требованиям. Например, наличие свойства `password` является обязательным.

Для создания сущности `Admin` выполните специальную команду `make:user` вместо обычной `make:entity`.

```
$ symfony console make:user Admin
```

Ответьте на вопросы в терминале: использовать ли Doctrine для хранения администраторов (`yes`), какое из свойств использовать для отображения имени администратора (`username`), должен ли каждый пользователь иметь пароль (`yes`).

Созданный класс содержит методы вроде `getRoles()`, `eraseCredentials()` и многие другие, необходимые Symfony для системы аутентификации.

Используйте команду `make:entity`, если вам нужно добавить дополнительные свойства в сущность `Admin`.

Также давайте добавим метод `__toString()`, так как его использует EasyAdmin:

```
--- a/src/Entity/Admin.php
+++ b/src/Entity/Admin.php
@@ -74,6 +74,11 @@ class Admin implements UserInterface
         return $this;
     }

+    public function __toString(): string
+    {
+        return $this->username;
+    }
+
     /**
      * @see UserInterface
      */
```

Помимо создания самой сущности `Admin`, команда также обновит

конфигурацию безопасности и свяжет сущность с системой аутентификации:

```
--- a/config/packages/security.yaml
+++ b/config/packages/security.yaml
@@ -1,7 +1,15 @@
 security:
+    encoders:
+        App\Entity\Admin:
+            algorithm: auto
+
     # https://symfony.com/doc/current/security.html#where-do-users-come-from-user-providers
     providers:
-        in_memory: { memory: null }
+        # used to reload user from session & other features (e.g. switch_user)
+        app_user_provider:
+            entity:
+                class: App\Entity\Admin
+                property: username
     firewalls:
         dev:
             pattern: ^/(_(profiler|wdt)|css|images|js)/
```

Выбор наилучшего алгоритма для хеширования паролей (который со временем будет меняться) мы оставим на усмотрение Symfony.

Пришло время создать миграцию и применить её к базе данных:

```
$ symfony console make:migration
$ symfony console doctrine:migrations:migrate -n
```

15.2 Создание пароля для администратора

Так как у нас будет всего лишь один администратор, мы не будем разрабатывать отдельную систему для создания администраторских учётных записей. В качестве логина используем `admin` и захешируем пароль.

Придумайте любой пароль и при помощи следующей команды захешируйте его:

```
$ symfony console security:encode-password


Symfony Password Encoder Utility
================================

 Type in your password to be encoded:
 >

 ------------------
-------------------------------------------------------------------------------------------------
  Key                 Value
 ------------------
-------------------------------------------------------------------------------------------------
  Encoder used        Symfony\Component\Security\Core\Encoder\MigratingPasswordEncoder
  Encoded password
$argon2id$v=19$m=65536,t=4,p=1$BQG+jovPcunctc3OxG5PxQ$TiGbx451NKdo+g9vLtfkMy4KjASKSOcnNxjij4g
 ------------------
-------------------------------------------------------------------------------------------------

 ! [NOTE] Self-salting encoder used: the encoder generated its own built-in salt.


 [OK] Password encoding succeeded
```

15.3 Создание администратора

Добавьте администратора, используя следующий SQL-запрос:

```
$ symfony run psql -c "INSERT INTO admin (id, username, roles, password) \
  VALUES (nextval('admin_id_seq'), 'admin', '[\"ROLE_ADMIN\"]', \

'\$argon2id\$v=19\$m=65536,t=4,p=1\$BQG+jovPcunctc3OxG5PxQ\$TiGbx451NKdo+g9vLtfkMy4KjASKSOcnN
```

Обратите внимание на экранирование знака `$` в столбце пароля; экранируйте их все!

15.4 Настройка аутентификации

Теперь, когда у нас есть пользователь с правами администратора, мы можем защитить административную панель. Symfony поддерживает

несколько стратегий аутентификации. Давайте воспользуемся классической и достаточно популярной *системой аутентификации с помощью формы*.

Выполните команду `make:auth`, чтобы обновить конфигурацию безопасности, сгенерировать шаблон с формой входа, а также создать *аутентификатор* (класс для управления аутентификацией):

```
$ symfony console make:auth
```

Выберите `1` для создания аутентификатора с формой входа, назовите класс аутентификатора — `AppAuthenticator`, контроллер — `SecurityController` и ответьте `yes`, чтобы добавить маршрут для выхода из системы по пути `/logout`.

Для связывания вновь созданных классов, команда обновит настройки безопасности:

```
--- a/config/packages/security.yaml
+++ b/config/packages/security.yaml
@@ -16,6 +16,13 @@ security:
             security: false
         main:
             anonymous: lazy
+            guard:
+                authenticators:
+                    - App\Security\AppAuthenticator
+            logout:
+                path: app_logout
+                # where to redirect after logout
+                # target: app_any_route

             # activate different ways to authenticate
             # https://symfony.com/doc/current/security.html#firewalls-
authentication
```

Благодаря подсказке во время выполнения команды, для перенаправления пользователя в случае успешного входа, нам также необходимо изменить маршрут в методе `onAuthenticationSuccess()`:

```
--- a/src/Security/AppAuthenticator.php
+++ b/src/Security/AppAuthenticator.php
@@ -94,8 +94,7 @@ class AppAuthenticator extends AbstractFormLoginAuthenticator
implements Passwor
             return new RedirectResponse($targetPath);
         }
```

```
-        // For example : return new RedirectResponse($this->urlGenerator-
>generate('some_route'));
-        throw new \Exception('TODO: provide a valid redirect inside
'.__FILE__);
+        return new RedirectResponse($this->urlGenerator-
>generate('easyadmin'));
     }

     protected function getLoginUrl()
```

Как узнать, что для доступа к EasyAdmin необходимо использовать маршрут `easyadmin`? Я не знаю. Но я запустил команду, которая показывает список всех доступных маршрутов вместе с их путями:

```
$ symfony console debug:router
```

15.5 Добавление правил контроля доступа для авторизации

Система безопасности состоит из двух частей: *аутентификация* и *авторизация* . При создании администратора мы добавили ему роль `ROLE_ADMIN`. Чтобы ограничить доступ к разделу `/admin` только для пользователей имеющих эту роль, необходимо добавить правило в `access_control`:

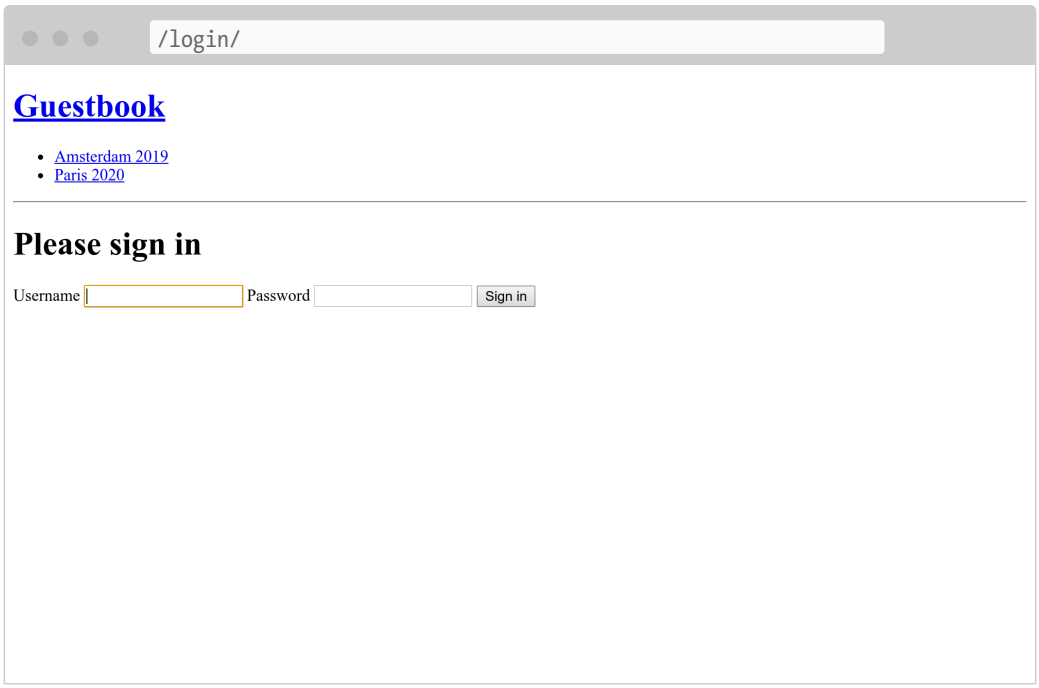
```
--- a/config/packages/security.yaml
+++ b/config/packages/security.yaml
@@ -33,5 +33,5 @@ security:
     # Easy way to control access for large sections of your site
     # Note: Only the *first* access control that matches will be used
     access_control:
-        # - { path: ^/admin, roles: ROLE_ADMIN }
+        - { path: ^/admin, roles: ROLE_ADMIN }
         # - { path: ^/profile, roles: ROLE_USER }
```

Правила `access_control` ограничивают доступ с помощью регулярных выражений. При переходе по URL-адресу, который начинается с `/admin`, система безопасности проверит наличие роли `ROLE_ADMIN` у

авторизованного пользователя.

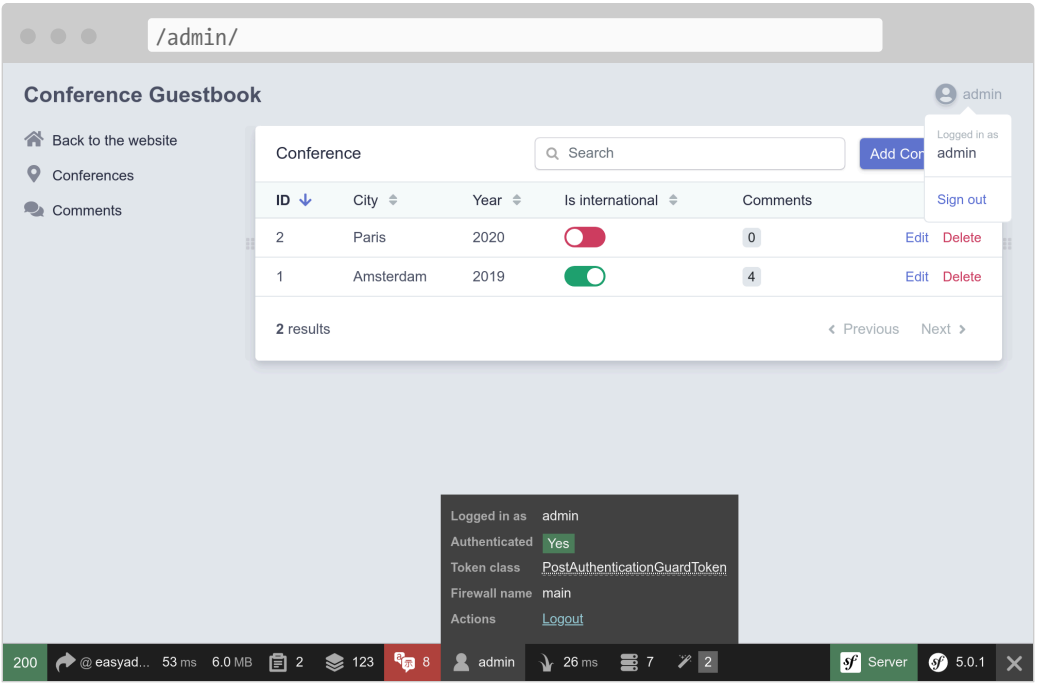
15.6 Аутентификация через форму входа

Теперь при открытии административной панели вы автоматически окажетесь на странице входа, где вам будет предложено ввести логин и пароль:



Войдите в систему, используя логин `admin` и незашифрованный пароль, который был захеширован вами ранее. Если вы корректно скопировали мою SQL-команду, то пароль будет `admin`.

Обратите внимание, что EasyAdmin автоматически распознает систему аутентификации Symfony:



Попробуйте нажать на ссылку «Sign out». Всё готово! Теперь у вас есть полностью защищённая административная панель.

Если вы хотите создать полноценную систему аутентификации с использованием формы, используйте команду `make:registration-form`.

Двигаемся дальше

- *Документация по Symfony Security*;
- *Обучающие видеоролики по безопасности на SymfonyCasts*;
- *Руководство по созданию формы входа* в Symfony-приложениях;
- *Шпаргалка по безопасности в Symfony*.

Шаг 16

Защита от спама с помощью API

Отзыв может оставить кто угодно, включая роботов и спамеров. Поэтому, чтобы снизить поток спама, мы можем либо добавить в форму капчу, либо использовать сторонние API.

Я решил использовать бесплатный сервис *Akismet*, чтобы показать, как можно работать с API и как выполнять внешние запросы.

16.1 Регистрация в Akismet

Зарегистрируйте бесплатный аккаунт на *akismet.com* и получите ключ Akismet API.

16.2 Добавление компонента Symfony HTTPClient

Мы будем обращаться к API Akismet напрямую вместо использования соответствующей библиотеки для этого. Выполнение HTTP-запросов самостоятельно более эффективно (кроме этого даёт использовать инструменты отладки Symfony, включая профилировщик Symfony).

Чтобы выполнить запрос к API воспользуемся Symfony-компонентом HttpClient:

```
$ symfony composer req http-client
```

16.3 Создание класса для проверки на спам

В директории `src/` создадим новый класс `SpamChecker`, который будет содержать логику отправки запроса к API Akismet и обработку его ответа:

src/SpamChecker.php

```
namespace App;

use App\Entity\Comment;
use Symfony\Contracts\HttpClient\HttpClientInterface;

class SpamChecker
{
    private $client;
    private $endpoint;

    public function __construct(HttpClientInterface $client, string $akismetKey)
    {
        $this->client = $client;
        $this->endpoint = sprintf('https://%s.rest.akismet.com/1.1/comment-
check', $akismetKey);
    }

    /**
     * @return int Spam score: 0: not spam, 1: maybe spam, 2: blatant spam
```

```
     *
     * @throws \RuntimeException if the call did not work
     */
    public function getSpamScore(Comment $comment, array $context): int
    {
        $response = $this->client->request('POST', $this->endpoint, [
            'body' => array_merge($context, [
                'blog' => 'https://guestbook.example.com',
                'comment_type' => 'comment',
                'comment_author' => $comment->getAuthor(),
                'comment_author_email' => $comment->getEmail(),
                'comment_content' => $comment->getText(),
                'comment_date_gmt' => $comment->getCreatedAt()->format('c'),
                'blog_lang' => 'en',
                'blog_charset' => 'UTF-8',
                'is_test' => true,
            ]),
        ]);

        $headers = $response->getHeaders();
        if ('discard' === ($headers['x-akismet-pro-tip'][0] ?? '')) {
            return 2;
        }

        $content = $response->getContent();
        if (isset($headers['x-akismet-debug-help'][0])) {
            throw new \RuntimeException(sprintf('Unable to check for spam: %s
(%s).', $content, $headers['x-akismet-debug-help'][0]));
        }

        return 'true' === $content ? 1 : 0;
    }
}
```

Метод HTTP-клиента `request()`отправляет POST-запрос на URL-адрес Akismet (`$this->endpoint`) и передаёт массив параметров.

Метод `getSpamScore()` возвращает 3 значения в зависимости от ответа на API-вызов:

- 2: если комментарий является явным спамом;
- 1: если комментарий может быть спамом;
- 0: если комментарий не спам (так называемый ham).

Используйте специальный адрес электронной почты `akismet-guaranteed-spam@example.com`, чтобы антиспам-сервис пометил такое сообщение как спам и вы таким образом смогли проверить его работу.

16.4 Использование переменных окружения

Класс `SpamChecker` зависит от параметра `$akismetKey`. Как и в случае с директорией для загруженных файлов, мы можем переместить ключ в параметр контейнера, используя свойство `bind`:

```
--- a/config/services.yaml
+++ b/config/services.yaml
@@ -12,6 +12,7 @@ services:
         autoconfigure: true # Automatically registers your services as
commands, event subscribers, etc.
         bind:
             $photoDir: "%kernel.project_dir%/public/uploads/photos"
+            $akismetKey: "%env(AKISMET_KEY)%"

     # makes classes in src/ available to be used as services
     # this creates a service per class whose id is the fully-qualified class
name
```

Разумеется, мы не будем хранить значение ключа Akismet в конфигурационном файле `services.yaml`, поэтому используем переменную окружения (`AKISMET_KEY`).

Затем каждый разработчик может сам определить переменную окружения в терминале или хранить ключ в файле `.env.local`:

.env.local

```
AKISMET_KEY=abcdef
```

Однако в продакшене должна быть определена только «реальная» переменная окружения в терминале.

Это неплохой рабочий вариант, хотя управление множеством переменных окружения может стать обременительным. В таком случае у Symfony есть «лучшая» альтернатива, когда речь заходит о

хранении конфиденциальных данных.

16.5 Хранение конфиденциальных данных

Вместо использования множества переменных окружения, в Symfony есть *хранилище*, в котором можно поместить много конфиденциальных данных. Среди одной из ключевых особенностей — можно сохранить хранилище в репозитории (но без ключа, чтобы его открыть). Другое замечательное преимущество состоит в том, что для каждого окружения может быть создано собственное хранилище.

На деле такие конфиденциальные данные являются неявными переменными окружения.

Добавьте ключ Akismet в хранилище:

```
$ symfony console secrets:set AKISMET_KEY


 Please type the secret value:
 >

 [OK] Secret "AKISMET_KEY" encrypted in "config/secrets/dev/"; you can commit
it.
```

Поскольку мы запускаем эту команду впервые, она сгенерировала два ключа в директорию `config/secret/dev/`. Затем эта команда сохранила секретную строку `AKISMET_KEY` в этой же директории.

Для хранения конфиденциальных данных в процессе разработки вы можете сохранить в репозитории хранилище вместе с ключами в директории `config/secret/dev/`.

Все конфиденциальные данные также можно переопределить путём определения одноимённой переменной окружения.

16.6 Проверка комментариев на спам

При отправке нового комментария одним из простых способов проверить его на спам — вызвать антиспам-сервис перед сохранением данных в базе данных:

```
--- a/src/Controller/ConferenceController.php
+++ b/src/Controller/ConferenceController.php
@@ -7,6 +7,7 @@ use App\Entity\Conference;
 use App\Form\CommentFormType;
 use App\Repository\CommentRepository;
 use App\Repository\ConferenceRepository;
+use App\SpamChecker;
 use Doctrine\ORM\EntityManagerInterface;
 use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
 use Symfony\Component\HttpFoundation\File\Exception\FileException;
@@ -39,7 +40,7 @@ class ConferenceController extends AbstractController
     /**
      * @Route("/conference/{slug}", name="conference")
      */
-    public function show(Request $request, Conference $conference,
CommentRepository $commentRepository, string $photoDir)
+    public function show(Request $request, Conference $conference,
CommentRepository $commentRepository, SpamChecker $spamChecker, string
$photoDir)
     {
         $comment = new Comment();
         $form = $this->createForm(CommentFormType::class, $comment);
@@ -58,6 +59,17 @@ class ConferenceController extends AbstractController
             }

             $this->entityManager->persist($comment);
+
+            $context = [
+                'user_ip' => $request->getClientIp(),
+                'user_agent' => $request->headers->get('user-agent'),
+                'referrer' => $request->headers->get('referer'),
+                'permalink' => $request->getUri(),
+            ];
+            if (2 === $spamChecker->getSpamScore($comment, $context)) {
+                throw new \RuntimeException('Blatant spam, go away!');
+            }
+
             $this->entityManager->flush();

             return $this->redirectToRoute('conference', ['slug' => $conference-
```

```
>getSlug()]);
```

Убедитесь, что всё работает правильно.

16.7 Управление конфиденциальными данными в продакшене

Для продакшена SymfonyCloud поддерживает установку *конфиденциальных переменных окружения*:

```
$ symfony var:set --sensitive AKISMET_KEY=abcdef
```

Однако, как отмечалось выше, использование механизма Symfony для управления конфиденциальными данными может быть более предпочтительным вариантом. Но не с точки зрения безопасности, а в плане управления секретными данными в команде проекта. Поскольку все конфиденциальные данные хранятся в репозитории, то чтобы использовать их в продакшене нужна только специальная переменная окружения с ключом дешифрования. Благодаря такому подходу каждый участник команды может добавить новые защищённые переменные окружения для использования в продакшене, даже если у него нет к нему доступа. Хотя для настройки этого процесса нужно кое-что сделать.

Прежде всего, сгенерируйте пару ключей для использования в продакшене:

```
$ APP_ENV=prod symfony console secrets:generate-keys
```

Повторно добавьте ключ Akismet в хранилище продакшена, но теперь уже с его действительным значением:

```
$ APP_ENV=prod symfony console secrets:set AKISMET_KEY
```

Последний шаг — отправьте ключ дешифрования в SymfonyCloud, установив специальную для этого переменную:

```
$ symfony var:set --sensitive SYMFONY_DECRYPTION_SECRET=`php -r 'echo
base64_encode(include("config/secrets/prod/prod.decrypt.private.php"));'`
```

Можно добавить все файлы в репозиторий, так как файл с ключом дешифрования уже игнорируется в `.gitignore`, поэтому он никогда не будет зафиксирован. Для большей безопасности лучше удалите его с вашего компьютера, потому что он уже есть в продакшене и больше не понадобится:

```
$ rm -f config/secrets/prod/prod.decrypt.private.php
```

Двигаемся дальше

- *Документация компонента HttpClient*;
- *Процессоры переменных окружения*;
- *Шпаргалка по компоненту Symfony HttpClient*.

Шаг 17
Тестирование

Поскольку мы всё больше и больше добавляем новой функциональности в приложение, наверное, сейчас самое время обсудить тестирование.

Забавный факт: я нашёл баг во время написания тестов в этой главе.

Symfony использует PHPUnit для модульного тестирования. Давайте установим его:

```
$ symfony composer req phpunit
```

17.1 Написание модульных тестов

Давайте начнём с тестирования класса `SpamChecker`. Создайте образец теста для него:

```
$ symfony console make:unit-test SpamCheckerTest
```

Протестировать SpamChecker непросто, потому что этот класс взаимодействует с удалённым API-интерфейсом Akismet, который нам

явно не стоит использовать во время тестирования. Поэтому будем *имитировать* работу этого API.

Давайте напишем наш первый тест для случая, когда API возвращает ошибку:

```
--- a/tests/SpamCheckerTest.php
+++ b/tests/SpamCheckerTest.php
@@ -2,12 +2,26 @@

 namespace App\Tests;

+use App\Entity\Comment;
+use App\SpamChecker;
 use PHPUnit\Framework\TestCase;
+use Symfony\Component\HttpClient\MockHttpClient;
+use Symfony\Component\HttpClient\Response\MockResponse;
+use Symfony\Contracts\HttpClient\ResponseInterface;

 class SpamCheckerTest extends TestCase
 {
-    public function testSomething()
+    public function testSpamScoreWithInvalidRequest()
     {
-        $this->assertTrue(true);
+        $comment = new Comment();
+        $comment->setCreatedAtValue();
+        $context = [];
+
+        $client = new MockHttpClient([new MockResponse('invalid',
['response_headers' => ['x-akismet-debug-help: Invalid key']])]);
+        $checker = new SpamChecker($client, 'abcde');
+
+        $this->expectException(\RuntimeException::class);
+        $this->expectExceptionMessage('Unable to check for spam: invalid
(Invalid key).');
+        $checker->getSpamScore($comment, $context);
     }
 }
```

С помощью класса `MockHttpClient` можно создать фиктивную реализацию любого HTTP-сервера. Для этого классу нужно передать массив с экземплярами `MockResponse`, каждый из которых содержит ожидаемые тело и заголовки ответа.

Затем вызываем метод `getSpamScore()` и проверяем, что было выброшено исключение, используя встроенный в PHPUnit метод `expectException()`.

Запустите тесты и посмотрите, что все они успешно выполнились:

```
$ symfony run bin/phpunit
```

Давайте добавим тесты на позитивный сценарий:

```
--- a/tests/SpamCheckerTest.php
+++ b/tests/SpamCheckerTest.php
@@ -24,4 +24,32 @@ class SpamCheckerTest extends TestCase
         $this->expectExceptionMessage('Unable to check for spam: invalid
(Invalid key).');
         $checker->getSpamScore($comment, $context);
     }
+
+    /**
+     * @dataProvider getComments
+     */
+    public function testSpamScore(int $expectedScore, ResponseInterface
$response, Comment $comment, array $context)
+    {
+        $client = new MockHttpClient([$response]);
+        $checker = new SpamChecker($client, 'abcde');
+
+        $score = $checker->getSpamScore($comment, $context);
+        $this->assertSame($expectedScore, $score);
+    }
+
+    public function getComments(): iterable
+    {
+        $comment = new Comment();
+        $comment->setCreatedAtValue();
+        $context = [];
+
+        $response = new MockResponse('', ['response_headers' => ['x-akismet-
pro-tip: discard']]);
+        yield 'blatant_spam' => [2, $response, $comment, $context];
+
+        $response = new MockResponse('true');
+        yield 'spam' => [1, $response, $comment, $context];
+
+        $response = new MockResponse('false');
+        yield 'ham' => [0, $response, $comment, $context];
+    }
 }
```

В PHPUnit есть провайдеры данных, при помощи которых в разных тестах можно использовать одни и те же тестовые данные.

17.2 Написание функциональных тестов для контроллеров

Тестирование контроллеров несколько отличается от тестирования «обычного» PHP-класса, поскольку процесс происходит в контексте HTTP-запроса.

Установите несколько дополнительных зависимостей, чтобы выполнять функциональные тесты:

```
$ symfony composer require browser-kit --dev
```

Создайте функциональный тест для контроллера конференций:

tests/Controller/ConferenceControllerTest.php

```
namespace App\Tests\Controller;

use Symfony\Bundle\FrameworkBundle\Test\WebTestCase;

class ConferenceControllerTest extends WebTestCase
{
    public function testIndex()
    {
        $client = static::createClient();
        $client->request('GET', '/');

        $this->assertResponseIsSuccessful();
        $this->assertSelectorTextContains('h2', 'Give your feedback');
    }
}
```

Первый тест проверяет, что главная страница возвращает статус 200 в HTTP-ответе.

Переменная `$client` предназначена для имитации браузера. Вместо отправки HTTP-запросов на сервер, происходит обращение к приложению Symfony напрямую. Такой подход имеет свои преимущества: он намного быстрее реального взаимодействия между клиентом и сервером, а ещё позволяет контролировать состояние сервисов после каждого HTTP-запроса.

Такие проверки, как `assertResponseIsSuccessful`, добавлены поверх PHPUnit, чтобы упростить вам работу. В Symfony ещё много подобных проверок.

Мы использовали / в качестве URL-адреса, вместо получения его через маршрутизатор. Это сделано не случайно, так как тестирование URL-адресов, используемых пользователем, являются частью того, что мы хотим протестировать. Если вы поменяете адрес, тесты уже не будут успешно выполняться. Это послужит полезным напоминанием о том, что вам, скорее всего, нужно сделать редирект со старого адреса на новый, чтобы поисковые машины и ссылающиеся сайты узнали об этом.

Создадим тест с помощью бандла maker:

```
$ symfony console make:functional-test Controller\\ConferenceController
```

Тесты PHPUnit выполняются в отдельном окружении `test`. Нам нужно создать переменную с секретным ключом `AKISMET_KEY` для этого окружения:

```
$ APP_ENV=test symfony console secrets:set AKISMET_KEY
```

Запустите новые тесты только для этого класса:

```
$ symfony run bin/phpunit tests/Controller/ConferenceControllerTest.php
```

Если тест не проходит, имеет смысл посмотреть на состояние объекта Response — вывести на экран результат выполнения метода `$client->getResponse()` через конструкцию `echo`.

17.3 Определение фикстур

Для тестирования списка комментариев, пагинации и отправки формы нужно заполнить базу данных какими-нибудь данными. При этом немаловажно поддерживать такие данные однородными между запусками тестов. И вот тут приходят на помощь фикстуры.

Установите бандл для фикстур в Doctrine:

```
$ symfony composer req orm-fixtures --dev
```

Во время установки была создана новая директория `src/DataFixtures/` вместе с примером класса, который можно редактировать как угодно. А пока добавьте в него две конференции и один комментарий:

```
--- a/src/DataFixtures/AppFixtures.php
+++ b/src/DataFixtures/AppFixtures.php
@@ -2,6 +2,8 @@

 namespace App\DataFixtures;

+use App\Entity\Comment;
+use App\Entity\Conference;
 use Doctrine\Bundle\FixturesBundle\Fixture;
 use Doctrine\Common\Persistence\ObjectManager;

@@ -9,8 +11,24 @@ class AppFixtures extends Fixture
 {
     public function load(ObjectManager $manager)
     {
-        // $product = new Product();
-        // $manager->persist($product);
+        $amsterdam = new Conference();
+        $amsterdam->setCity('Amsterdam');
+        $amsterdam->setYear('2019');
+        $amsterdam->setIsInternational(true);
+        $manager->persist($amsterdam);
+
+        $paris = new Conference();
+        $paris->setCity('Paris');
+        $paris->setYear('2020');
+        $paris->setIsInternational(false);
+        $manager->persist($paris);
+
+        $comment1 = new Comment();
+        $comment1->setConference($amsterdam);
+        $comment1->setAuthor('Fabien');
+        $comment1->setEmail('fabien@example.com');
+        $comment1->setText('This was a great conference.');
+        $manager->persist($comment1);

         $manager->flush();
     }
```

После применения фикстур все ранее сохранённые данные будут удалены, включая пользователя-администратора. Чтобы этого не произошло, добавим его в фикстуры:

```
--- a/src/DataFixtures/AppFixtures.php
```

```
+++ b/src/DataFixtures/AppFixtures.php
@@ -2,13 +2,22 @@

 namespace App\DataFixtures;

+use App\Entity\Admin;
 use App\Entity\Comment;
 use App\Entity\Conference;
 use Doctrine\Bundle\FixturesBundle\Fixture;
 use Doctrine\Common\Persistence\ObjectManager;
+use Symfony\Component\Security\Core\Encoder\EncoderFactoryInterface;

 class AppFixtures extends Fixture
 {
+    private $encoderFactory;
+
+    public function __construct(EncoderFactoryInterface $encoderFactory)
+    {
+        $this->encoderFactory = $encoderFactory;
+    }
+
     public function load(ObjectManager $manager)
     {
         $amsterdam = new Conference();
@@ -30,6 +39,12 @@ class AppFixtures extends Fixture
         $comment1->setText('This was a great conference.');
         $manager->persist($comment1);

+        $admin = new Admin();
+        $admin->setRoles(['ROLE_ADMIN']);
+        $admin->setUsername('admin');
+        $admin->setPassword($this->encoderFactory->getEncoder(Admin::class)-
>encodePassword('admin', null));
+        $manager->persist($admin);
+
         $manager->flush();
     }
 }
```

Если вы не помните, какой сервис вам нужно использовать для выполнения какой-либо задачи, воспользуетесь командой `debug:autowiring`, указав в качестве аргумента ключевое слово для поиска:

```
$ symfony console debug:autowiring encoder
```

17.4 Применение фикстур

Загрузите данные из фикстур в базу данных. **Помните**, что это действие удалит *все* записи в базе данных (если вы не хотите этого, ничего не делайте и перейдите к следующему разделу).

```
$ symfony console doctrine:fixtures:load
```

17.5 Сканирование сайта в функциональных тестах

Как мы уже видели, используемый в тестах HTTP-клиент имитирует работу браузера. Таким образом, можно переходить по страницам сайта, как при использовании браузера без графического интерфейса.

Напишите новый тест, который щёлкает по любой ссылке конференции на главной странице:

```
--- a/tests/Controller/ConferenceControllerTest.php
+++ b/tests/Controller/ConferenceControllerTest.php
@@ -14,4 +14,19 @@ class ConferenceControllerTest extends WebTestCase
         $this->assertResponseIsSuccessful();
         $this->assertSelectorTextContains('h2', 'Give your feedback');
     }
+
+    public function testConferencePage()
+    {
+        $client = static::createClient();
+        $crawler = $client->request('GET', '/');
+
+        $this->assertCount(2, $crawler->filter('h4'));
+
+        $client->clickLink('View');
+
+        $this->assertPageTitleContains('Amsterdam');
+        $this->assertResponseIsSuccessful();
+        $this->assertSelectorTextContains('h2', 'Amsterdam 2019');
+        $this->assertSelectorExists('div:contains("There are 1 comments")');
+    }
 }
```

Давайте опишем, какие действия должны происходить в тесте на

простом английском языке:

- Как и в первом тесте, сначала переходим на главную страницу;
- Метод `request()` возвращает экземпляр `Crawler`, с помощью которого можно найти нужные элементы на странице (например, ссылки, формы и всё остальное, что можно получить через селекторы CSS или XPath);
- Благодаря CSS-селектору проверяем, что на главной странице есть две конференции;
- Далее кликаем по ссылке «View» (из-за того, что один вызов щёлкает только по одной ссылке, Symfony автоматически выберет первую найденную в разметке);
- Проверяем заголовок страницы, ответ и тег `<h2>` для того, чтобы знать наверняка, что мы находимся на нужной странице (дополнительно можно было проверить на совпадение маршрут);
- И последнее: проверяем, что на странице есть 1 комментарий. В Symfony кое-какие некорректные в CSS селекторы позаимствованы из jQuery. Как раз один из таких селекторов мы используем — `div:contains()`.

Опять же при помощи CSS-селектора вместо нажатия на текст (`View`), выбираем нужную ссылку:

```
$client->click($crawler->filter('h4 + p a')->link());
```

Проследите, чтобы новый тест выполнился успешно:

```
$ symfony run bin/phpunit tests/Controller/ConferenceControllerTest.php
```

17.6 Работа с тестовой базой данных

По умолчанию тесты выполняются в Symfony-окружении `test`, как это определено в файле `phpunit.xml.dist`:

phpunit.xml.dist

```
<phpunit>
    <php>
        <server name="APP_ENV" value="test" force="true" />
    </php>
</phpunit>
```

Если вам нужно выполнять тесты в другой базе данных, переопределите переменную окружения `DATABASE_URL` в файле `.env.test`:

```
--- a/.env.test
+++ b/.env.test
@@ -1,4 +1,5 @@
 # define your env variables for the test env here
+DATABASE_URL=postgres://main:main@127.0.0.1:32773/
test?sslmode=disable&charset=utf8
 KERNEL_CLASS='App\Kernel'
 APP_SECRET='$ecretf0rt3st'
 SYMFONY_DEPRECATIONS_HELPER=999999
```

Загрузите фикстуры для окружения `test`:

```
$ APP_ENV=test symfony console doctrine:fixtures:load
```

Далее в этом шаге мы не будем изменять переменную окружения `DATABASE_URL`. Использование одной и той же тестовой базы данных также и в окружении `dev` даёт массу преимуществ, которые мы рассмотрим в следующем разделе.

17.7 Отправка формы в функциональном тесте

Хотите пойти дальше и ещё лучше протестировать приложение? Давайте напишем тест, который имитирует отправку форму, добавляя новый комментарий с фотографией к конференции. Непростая задача, не правда? Как бы не так: только посмотрите на используемый для решения код — это не сложнее того, что мы уже делали ранее:

```
--- a/tests/Controller/ConferenceControllerTest.php
```

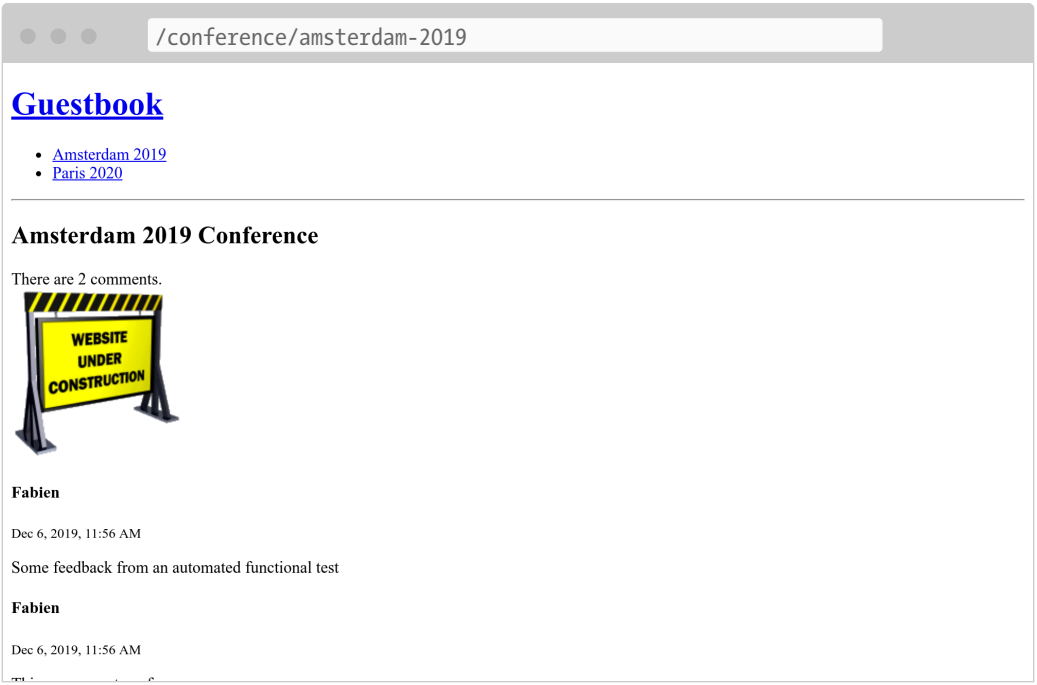
```
+++ b/tests/Controller/ConferenceControllerTest.php
@@ -29,4 +29,19 @@ class ConferenceControllerTest extends WebTestCase
         $this->assertSelectorTextContains('h2', 'Amsterdam 2019');
         $this->assertSelectorExists('div:contains("There are 1 comments")');
     }
+
+    public function testCommentSubmission()
+    {
+        $client = static::createClient();
+        $client->request('GET', '/conference/amsterdam-2019');
+        $client->submitForm('Submit', [
+            'comment_form[author]' => 'Fabien',
+            'comment_form[text]' => 'Some feedback from an automated
functional test',
+            'comment_form[email]' => 'me@automat.ed',
+            'comment_form[photo]' => dirname(__DIR__, 2).'/public/images/under-
construction.gif',
+        ]);
+        $this->assertResponseRedirects();
+        $client->followRedirect();
+        $this->assertSelectorExists('div:contains("There are 2 comments")');
+    }
 }
```

Для отправки формы через метод `submitForm()` для начала нужно узнать имена элементов с помощью инструментов разработки в браузере или через панель Form в профилировщике Symfony. Обратите внимание на разумное повторное использование изображения-заглушки!

Снова запустите тесты, чтобы убедиться, что они все прошли:

```
$ symfony run bin/phpunit tests/Controller/ConferenceControllerTest.php
```

Теперь мы видим один из плюсов использования одинаковой базы данных в окружении разработки и тестирования — полученный результат можно посмотреть в браузере:



17.8 Выгрузка фикстур

Если запустить тесты ещё раз, то они больше не пройдут. А всё потому, что в базе данных уже есть комментарии, и при повторном выполнении тестов они снова добавятся. Вот поэтому проверка количества комментариев не даст положительного результата. Таким образом, перед каждым выполнением тестов нужно очищать базу данных и заново загружать фикстуры:

```
$ symfony console doctrine:fixtures:load
$ symfony run bin/phpunit tests/Controller/ConferenceControllerTest.php
```

17.9 Автоматизация рабочего процесса с помощью Make-файлов

Помнить и набирать длинную последовательность команд, чтобы выполнить тесты — досадно и неприятно. Как минимум это всё нужно указать в документации, но оставим это на крайний случай. А что, если автоматизировать такие рутинные задачи? Вдобавок это дало своего рода документацию, помогло остальным разработчикам, и просто

ускорило разработку.

Один из способов автоматизации выполнения команд — воспользоваться `Makefile`:

Makefile
```
SHELL := /bin/bash

tests:
    symfony console doctrine:fixtures:load -n
    symfony run bin/phpunit
.PHONY: tests
```

Обратите внимание на флаг `-n` в команде Doctrine; это глобальный флаг для Symfony-команд, который выполняет их в обычном (неинтерактивном) режиме.

Теперь, если вам нужно выполнить тесты, то используйте команду `make tests`:

```
$ make tests
```

17.10 Очистка базы данных после каждого теста

Чистить базу данных после каждого запуска тестов, конечно, очень хорошо, но иметь действительно независимые друг от друга тесты — гораздо лучше. Крайне нежелательно, чтобы один тест опирался на результаты от предыдущих. Помимо этого, изменение порядка выполнения тестов также не должно влиять на результат. Как мы сейчас убедимся, наши тесты сейчас не соответствуют описанному выше условию.

Переместите тест `testConferencePage` после `testCommentSubmission`:

```
--- a/tests/Controller/ConferenceControllerTest.php
+++ b/tests/Controller/ConferenceControllerTest.php
@@ -15,21 +15,6 @@ class ConferenceControllerTest extends WebTestCase
         $this->assertSelectorTextContains('h2', 'Give your feedback');
     }
```

```
-    public function testConferencePage()
-    {
-        $client = static::createClient();
-        $crawler = $client->request('GET', '/');
-
-        $this->assertCount(2, $crawler->filter('h4'));
-
-        $client->clickLink('View');
-
-        $this->assertPageTitleContains('Amsterdam');
-        $this->assertResponseIsSuccessful();
-        $this->assertSelectorTextContains('h2', 'Amsterdam 2019');
-        $this->assertSelectorExists('div:contains("There are 1 comments")');
-    }
-
     public function testCommentSubmission()
     {
         $client = static::createClient();
@@ -44,4 +29,19 @@ class ConferenceControllerTest extends WebTestCase
         $crawler = $client->followRedirect();
         $this->assertSelectorExists('div:contains("There are 2 comments")');
     }
+
+    public function testConferencePage()
+    {
+        $client = static::createClient();
+        $crawler = $client->request('GET', '/');
+
+        $this->assertCount(2, $crawler->filter('h4'));
+
+        $client->clickLink('View');
+
+        $this->assertPageTitleContains('Amsterdam');
+        $this->assertResponseIsSuccessful();
+        $this->assertSelectorTextContains('h2', 'Amsterdam 2019');
+        $this->assertSelectorExists('div:contains("There are 1 comments")');
+    }
 }
```

Теперь тесты завершились неудачно.

Для очистки базы данных между прогонами тестов установите DoctrineTestBundle:

```
$ symfony composer require dama/doctrine-test-bundle --dev
```

Чтобы установить этот бандл потребуется вручную в терминале подтвердить выполнение его рецепта (так как он не поддерживается

«официально»):

```
Symfony operations: 1 recipe (d7f110145ba9f62430d1ad64d57ab069)
  -  WARNING  dama/doctrine-test-bundle (>=4.0): From github.com/symfony/
recipes-contrib:master
    The recipe for this package comes from the "contrib" repository, which is
open to community contributions.
    Review the recipe at https://github.com/symfony/recipes-contrib/tree/master/
dama/doctrine-test-bundle/4.0

    Do you want to execute this recipe?
    [y] Yes
    [n] No
    [a] Yes for all packages, only for the current installation session
    [p] Yes permanently, never ask again for this project
    (defaults to n): p
```

Активируйте обработчик в PHPUnit:

```
--- a/phpunit.xml.dist
+++ b/phpunit.xml.dist
@@ -27,6 +27,10 @@
         </whitelist>
     </filter>

+    <extensions>
+        <extension class="DAMA\DoctrineTestBundle\PHPUnit\PHPUnitExtension" />
+    </extensions>
+
     <listeners>
         <listener class="Symfony\Bridge\PhpUnit\SymfonyTestsListener" />
     </listeners>
```

Вот и всё, теперь любые изменения данных в тестах будут автоматически отменены после их выполнения.

Тесты снова заработали:

```
$ make tests
```

17.11 Использование настоящего браузера для функциональных

тестов

Только что мы рассмотрели использование специального браузера для функциональных тестов, который напрямую вызывает код Symfony. Однако вместо этого можно задействовать настоящий браузер, отправляя реальные HTTP-запросы. Встречайте, Symfony Panther:

На момент написания этого параграфа невозможно было установить Panther в проект на Symfony 5 из-за того, что одна из зависимостей этого бандла не была совместима с новой версией фреймворка.

```
$ symfony composer req panther --dev
```

После установки можно написать тесты, в которых используется настоящий браузер Google Chrome, как показано ниже:

```
--- a/tests/Controller/ConferenceControllerTest.php
+++ b/tests/Controller/ConferenceControllerTest.php
@@ -2,13 +2,13 @@

 namespace App\Tests\Controller;

-use Symfony\Bundle\FrameworkBundle\Test\WebTestCase;
+use Symfony\Component\Panther\PantherTestCase;

-class ConferenceControllerTest extends WebTestCase
+class ConferenceControllerTest extends PantherTestCase
 {
     public function testIndex()
     {
-        $client = static::createClient();
+        $client = static::createPantherClient(['external_base_uri' =>
$_SERVER['SYMFONY_DEFAULT_ROUTE_URL']]);
         $client->request('GET', '/');

         $this->assertResponseIsSuccessful();
```

Переменная окружения `SYMFONY_DEFAULT_ROUTE_URL` содержит URL-адрес локального веб-сервера.

17.12 Выполнение

функциональных тестов по принципу чёрного ящика с помощью Blackfire

Приложение *Blackfire player* — ещё один способ запустить функциональные тесты. Помимо функционального тестирования, этот инструмент также может проводить тестирование производительности.

Для более подробного изучения этой темы посмотрите шаг «Производительность».

Двигаемся дальше

- *Список проверок в Symfony* для функциональных тестов;
- *Документация по PHPUnit*;
- *Библиотека Faker* для генерации реалистичных данных в фикстурах;
- *Документация по компоненту CssSelector*;
- *Symfony-библиотека Panther* для тестирования в браузере и парсинга сайтов в приложениях Symfony;
- *Документация по Make/Makefile*.

Шаг 18

Переход к асинхронности

Во время проверки формы на наличие спама могут возникнуть проблемы. В случае, если ответ от Akismet API будет долгим, наша страница с формой также станет работать медленнее. Худший сценарий — мы можем совсем потерять комментарии из-за достижения тайм-аута или недоступности Akismet API.

В идеале нам нужно немедленно возвращать ответ на отправленную форму, а приложение должно сохранять данные без попытки их сразу же опубликовать. Проверить на спам можно и позже.

18.1 Установка статусов комментариев

Реализуем статусы (`state`) для комментариев: `submitted` (отправлен), `spam` (спам) и `published` (опубликован).

Добавим свойство `state` в класс `Comment`:

```
$ symfony console make:entity Comment
```

Создайте миграцию базы данных:

```
$ symfony console make:migration
```

Измените миграцию, чтобы все существующие комментарии были по умолчанию со статусом published:

```
--- a/src/Migrations/Version00000000000000.php
+++ b/src/Migrations/Version00000000000000.php
@@ -22,7 +22,9 @@ final class Version00000000000000 extends AbstractMigration
         // this up() migration is auto-generated, please modify it to your
needs
         $this->abortIf($this->connection->getDatabasePlatform()->getName() !==
'postgresql', 'Migration can only be executed safely on \'postgresql\'.');

-        $this->addSql('ALTER TABLE comment ADD state VARCHAR(255) NOT NULL');
+        $this->addSql('ALTER TABLE comment ADD state VARCHAR(255)');
+        $this->addSql("UPDATE comment SET state='published'");
+        $this->addSql('ALTER TABLE comment ALTER COLUMN state SET NOT NULL');
     }

     public function down(Schema $schema) : void
```

Выполните миграцию базы данных:

```
$ symfony console doctrine:migrations:migrate
```

Убедимся, что для свойства state по умолчанию задан статус submitted.

```
--- a/src/Entity/Comment.php
+++ b/src/Entity/Comment.php
@@ -49,9 +49,9 @@ class Comment
     private $photoFilename;

     /**
-     * @ORM\Column(type="string", length=255)
+     * @ORM\Column(type="string", length=255, options={"default": "submitted"})
      */
-    private $state;
+    private $state = 'submitted';
```

```
    public function __toString(): string
    {
```

Обновите конфигурацию EasyAdmin, чтобы отображать статус комментария в административной панели:

```
--- a/config/packages/easy_admin.yaml
+++ b/config/packages/easy_admin.yaml
@@ -18,6 +18,7 @@ easy_admin:
                     - author
                     - { property: 'email', type: 'email' }
                     - { property: 'photoFilename', type: 'image', 'base_path':
"/uploads/photos", label: 'Photo' }
+                    - state
                     - { property: 'createdAt', type: 'datetime' }
                 sort: ['createdAt', 'ASC']
                 filters: ['conference']
@@ -26,5 +27,6 @@ easy_admin:
                     - { property: 'conference' }
                     - { property: 'createdAt', type: datetime, type_options: {
attr: { readonly: true } } }
                     - 'author'
+                    - { property: 'state' }
                     - { property: 'email', type: 'email' }
                     - text
```

Не забудьте также обновить тесты, установив новое свойство state в фикстурах:

```
--- a/src/DataFixtures/AppFixtures.php
+++ b/src/DataFixtures/AppFixtures.php
@@ -37,8 +37,16 @@ class AppFixtures extends Fixture
         $comment1->setAuthor('Fabien');
         $comment1->setEmail('fabien@example.com');
         $comment1->setText('This was a great conference.');
+        $comment1->setState('published');
         $manager->persist($comment1);

+        $comment2 = new Comment();
+        $comment2->setConference($amsterdam);
+        $comment2->setAuthor('Lucas');
+        $comment2->setEmail('lucas@example.com');
+        $comment2->setText('I think this one is going to be moderated.');
+        $manager->persist($comment2);
+
         $admin = new Admin();
```

```
        $admin->setRoles(['ROLE_ADMIN']);
        $admin->setUsername('admin');
```

Для тестирования контроллера смоделируйте проверку комментария:

```
--- a/tests/Controller/ConferenceControllerTest.php
+++ b/tests/Controller/ConferenceControllerTest.php
@@ -2,6 +2,8 @@

 namespace App\Tests\Controller;

+use App\Repository\CommentRepository;
+use Doctrine\ORM\EntityManagerInterface;
 use Symfony\Bundle\FrameworkBundle\Test\WebTestCase;

 class ConferenceControllerTest extends WebTestCase
@@ -22,11 +24,17 @@ class ConferenceControllerTest extends WebTestCase
         $client->submitForm('Submit', [
             'comment_form[author]' => 'Fabien',
             'comment_form[text]' => 'Some feedback from an automated
functional test',
-            'comment_form[email]' => 'me@automat.ed',
+            'comment_form[email]' => $email = 'me@automat.ed',
             'comment_form[photo]' => dirname(__DIR__, 2).'/public/images/under-
construction.gif',
         ]);
         $this->assertResponseRedirects();
+
+        // simulate comment validation
+        $comment = self::$container->get(CommentRepository::class)-
>findOneByEmail($email);
+        $comment->setState('published');
+        self::$container->get(EntityManagerInterface::class)->flush();
+
         $client->followRedirect();
         $this->assertSelectorExists('div:contains("There are 2 comments")');
     }
```

В тесте PHPUnit вы можете получить любой сервис из контейнера с помощью метода `self::$container->get()`, который также даёт доступ к непубличным сервисам.

18.2 Внедрение компонента Messenger

Компонент Messenger управляет асинхронным кодом в Symfony:

```
$ symfony composer req messenger
```

Для выполнения асинхронных задач, отправьте *сообщение* на *шину обмена сообщениями*. Шина сохраняет сообщение в *очередь* и немедленно возвращает ответ, чтобы исключить любые задержки в ваших приложениях.

Получатель работает постоянно в фоновом режиме, читая новые сообщения из очереди и выполняя соответствующие задачи. Получатель может работать как на том же сервере, где находится само приложение, так и на отдельном сервере.

Это напоминает обработку HTTP-запросов, за исключением того, что здесь нет ответов.

18.3 Создание обработчика сообщений

Сообщение — это класс с данными, в котором не должно быть никакой логики. Он будет сериализован для хранения в очереди, поэтому держите в нем только «простые» сериализуемые данные.

Создайте класс `CommentMessage`:

src/Message/CommentMessage.php

```
namespace App\Message;

class CommentMessage
{
    private $id;
    private $context;

    public function __construct(int $id, array $context = [])
    {
        $this->id = $id;
```

```
        $this->context = $context;
    }

    public function getId(): int
    {
        return $this->id;
    }

    public function getContext(): array
    {
        return $this->context;
    }
}
```

В мире Messenger нет контроллеров, есть только обработчики сообщений.

Создайте класс `CommentMessageHandler` в новом пространстве имён `App\MessageHandler`, который знает, как обрабатывать сообщения `CommentMessage`:

src/MessageHandler/CommentMessageHandler.php

```
namespace App\MessageHandler;

use App\Message\CommentMessage;
use App\Repository\CommentRepository;
use App\SpamChecker;
use Doctrine\ORM\EntityManagerInterface;
use Symfony\Component\Messenger\Handler\MessageHandlerInterface;

class CommentMessageHandler implements MessageHandlerInterface
{
    private $spamChecker;
    private $entityManager;
    private $commentRepository;

    public function __construct(EntityManagerInterface $entityManager,
SpamChecker $spamChecker, CommentRepository $commentRepository)
    {
        $this->entityManager = $entityManager;
        $this->spamChecker = $spamChecker;
        $this->commentRepository = $commentRepository;
    }

    public function __invoke(CommentMessage $message)
    {
        $comment = $this->commentRepository->find($message->getId());
```

```
        if (!$comment) {
            return;
        }

        if (2 === $this->spamChecker->getSpamScore($comment,
$message->getContext())) {
            $comment->setState('spam');
        } else {
            $comment->setState('published');
        }

        $this->entityManager->flush();
    }
}
```

`MessageHandlerInterface` — это *маркирующий* интерфейс. Его наличие помогает Symfony понять, что класс нужно автоматически зарегистрировать и настроить в качестве обработчика Messenger. По соглашению логика обработчика находится в методе `__invoke()`. Подсказка об используемом типе `CommentMessage` для единственного аргумента этого метода сообщает Messenger, какой класс он будет обрабатывать.

Обновите контроллер, чтобы использовать новую систему:

```
--- a/src/Controller/ConferenceController.php
+++ b/src/Controller/ConferenceController.php
@@ -5,14 +5,15 @@ namespace App\Controller;
 use App\Entity\Comment;
 use App\Entity\Conference;
 use App\Form\CommentFormType;
+use App\Message\CommentMessage;
 use App\Repository\CommentRepository;
 use App\Repository\ConferenceRepository;
-use App\SpamChecker;
 use Doctrine\ORM\EntityManagerInterface;
 use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
 use Symfony\Component\HttpFoundation\File\Exception\FileException;
 use Symfony\Component\HttpFoundation\Request;
 use Symfony\Component\HttpFoundation\Response;
+use Symfony\Component\Messenger\MessageBusInterface;
 use Symfony\Component\Routing\Annotation\Route;
 use Twig\Environment;

@@ -20,11 +21,13 @@ class ConferenceController extends AbstractController
 {
     private $twig;
```

```
     private $entityManager;
+    private $bus;

-    public function __construct(Environment $twig, EntityManagerInterface
$entityManager)
+    public function __construct(Environment $twig, EntityManagerInterface
$entityManager, MessageBusInterface $bus)
     {
         $this->twig = $twig;
         $this->entityManager = $entityManager;
+        $this->bus = $bus;
     }

     /**
@@ -40,7 +43,7 @@ class ConferenceController extends AbstractController
      /**
       * @Route("/conference/{slug}", name="conference")
       */
-    public function show(Request $request, Conference $conference,
CommentRepository $commentRepository, SpamChecker $spamChecker, string
$photoDir)
+    public function show(Request $request, Conference $conference,
CommentRepository $commentRepository, string $photoDir)
     {
         $comment = new Comment();
         $form = $this->createForm(CommentFormType::class, $comment);
@@ -59,6 +62,7 @@ class ConferenceController extends AbstractController
             }

             $this->entityManager->persist($comment);
+            $this->entityManager->flush();

             $context = [
                 'user_ip' => $request->getClientIp(),
@@ -66,11 +70,8 @@ class ConferenceController extends AbstractController
                 'referrer' => $request->headers->get('referer'),
                 'permalink' => $request->getUri(),
             ];
-            if (2 === $spamChecker->getSpamScore($comment, $context)) {
-                throw new \RuntimeException('Blatant spam, go away!');
-            }

-            $this->entityManager->flush();
+            $this->bus->dispatch(new CommentMessage($comment->getId(),
$context));

             return $this->redirectToRoute('conference', ['slug' => $conference-
>getSlug()]);
         }
```

Теперь вместо того, чтобы зависеть от скорости работы антиспам-сервиса, мы отправляем сообщение на шину. Затем обработчик решает, что делать с этим сообщением.

Произошло что-то неожиданное: мы отделили наш контроллер от антиспам-сервиса и перенесли логику в новый класс — обработчик. Это идеальный вариант для использования шины. Проверьте код — всё работает. Операции всё ещё выполняются синхронно, но код, вероятно, уже стал «лучше».

18.4 Сокращение количества отображаемых комментариев

Обновите логику отображения, чтобы на странице показывались только опубликованные комментарии:

```
--- a/src/Repository/CommentRepository.php
+++ b/src/Repository/CommentRepository.php
@@ -25,7 +25,9 @@ class CommentRepository extends ServiceEntityRepository
     {
         return $this->createQueryBuilder('c')
             ->andWhere('c.conference = :conference')
+            ->andWhere('c.state = :state')
             ->setParameter('conference', $conference)
+            ->setParameter('state', 'published')
             ->orderBy('c.createdAt', 'DESC')
             ->setMaxResults($limit)
             ->setFirstResult($offset)
```

18.5 Переходим к асинхронности по-настоящему

По умолчанию обработчики вызываются синхронно. Для асинхронного вызова необходимо указать, какую очередь использовать для каждого обработчика в конфигурационном файле `config/packages/messenger.yaml`:

```
--- a/config/packages/messenger.yaml
+++ b/config/packages/messenger.yaml
@@ -5,10 +5,10 @@ framework:

         transports:
             # https://symfony.com/doc/current/messenger.html#transport-
configuration
-            # async: '%env(MESSENGER_TRANSPORT_DSN)%'
+            async: '%env(RABBITMQ_DSN)%'
             # failed: 'doctrine://default?queue_name=failed'
             # sync: 'sync://'

         routing:
             # Route your messages to the transports
-            # 'App\Message\YourMessage': async
+            App\Message\CommentMessage: async
```

Шина настроена на отсылку сообщений типа `App\Message\CommentMessage` в очередь `async`, которая определена DSN, хранящейся в переменной окружения `RABBITMQ_DSN`.

18.6 Добавление RabbitMQ в Docker

Как вы могли догадаться, мы будем использовать RabbitMQ:

```
--- a/docker-compose.yaml
+++ b/docker-compose.yaml
@@ -12,3 +12,7 @@ services:
     redis:
         image: redis:5-alpine
         ports: [6379]
+
+    rabbitmq:
+        image: rabbitmq:3.7-management
+        ports: [5672, 15672]
```

18.7 Перезапуск сервисов Docker

Чтобы Docker Compose запустил контейнер с RabbitMQ, остановите и

перезапустите контейнеры:

```
$ docker-compose stop
$ docker-compose up -d
```

Dumping and Restoring Database Data

Выполнение команды `docker-compose down` приведёт к потере данных, поэтому перед этим стоит сделать резервную копию базы данных с помощью `pg_dump`:

```
$ symfony run pg_dump --data-only > dump.sql
```

А затем, чтобы восстановить данные, воспользуйтесь следующей командой:

```
$ symfony run psql < dump.sql
```

18.8 Получение сообщений

При отправке нового комментария, проверка на спам больше не сработает. Чтобы убедиться в этом, добавьте вызов функции“error_log()“ в метод `getSpamScore()`. Видно, что сообщение находится в очереди RabbitMQ и ждёт обработки.

Как вы понимаете, у Symfony есть команда для получения сообщений. Давайте выполним её сейчас:

```
$ symfony console messenger:consume async -vv
```

Команда немедленно получит сообщение, отправленное для комментария:

```
 [OK] Consuming messages from transports "async".

 // The worker will automatically exit once it has received a stop signal via
the messenger:stop-workers command.

 // Quit the worker with CONTROL-C.
```

```
11:30:20 INFO      [messenger] Received message App\Message\CommentMessage
["message" => App\Message\CommentMessage^ { …},"class" => "App\Message\
CommentMessage"]
11:30:20 INFO      [http_client] Request: "POST
https://80cea32be1f6.rest.akismet.com/1.1/comment-check"
11:30:20 INFO      [http_client] Response: "200
https://80cea32be1f6.rest.akismet.com/1.1/comment-check"
11:30:20 INFO      [messenger] Message App\Message\CommentMessage handled by
App\MessageHandler\CommentMessageHandler::__invoke ["message" => App\Message\
CommentMessage^ { …},"class" => "App\Message\CommentMessage","handler" => "App\
MessageHandler\CommentMessageHandler::__invoke"]
11:30:20 INFO      [messenger] App\Message\CommentMessage was handled
successfully (acknowledging to transport). ["message" => App\Message\
CommentMessage^ { …},"class" => "App\Message\CommentMessage"]
```

Все действия получателя сообщений пишутся в логи, но эти же действия можно увидеть в консоли в реальном времени, если к команде добавить флаг `-vv`. Вы даже сможете увидеть вызов к Akismet API.

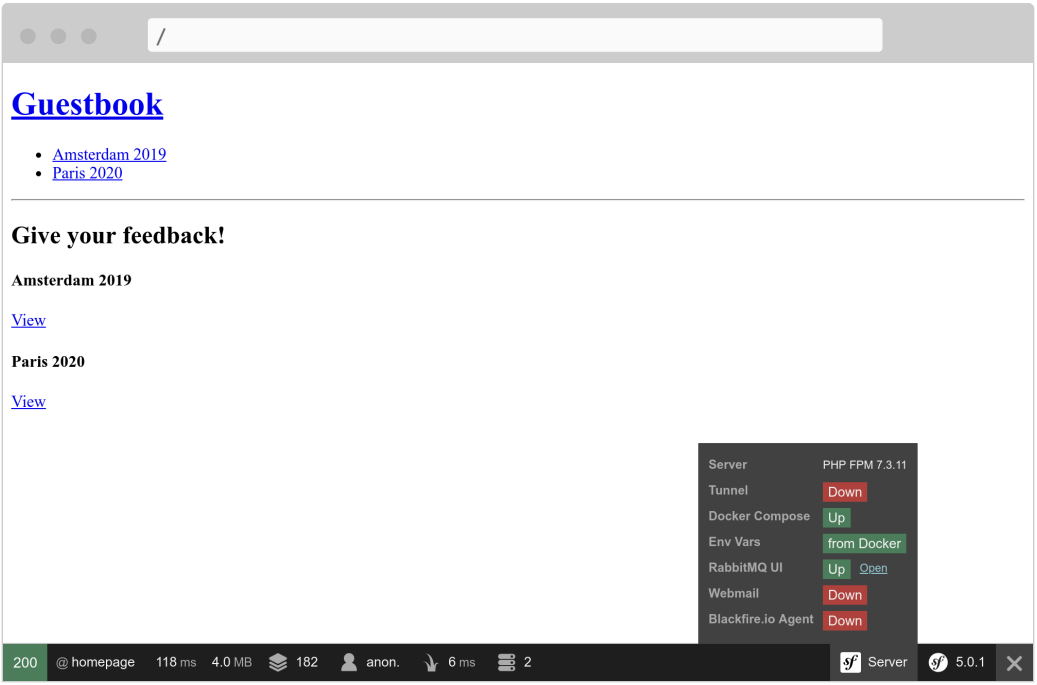
Для остановки получателя, нажмите `Ctrl+C`.

18.9 Знакомство с панелью управления RabbitMQ

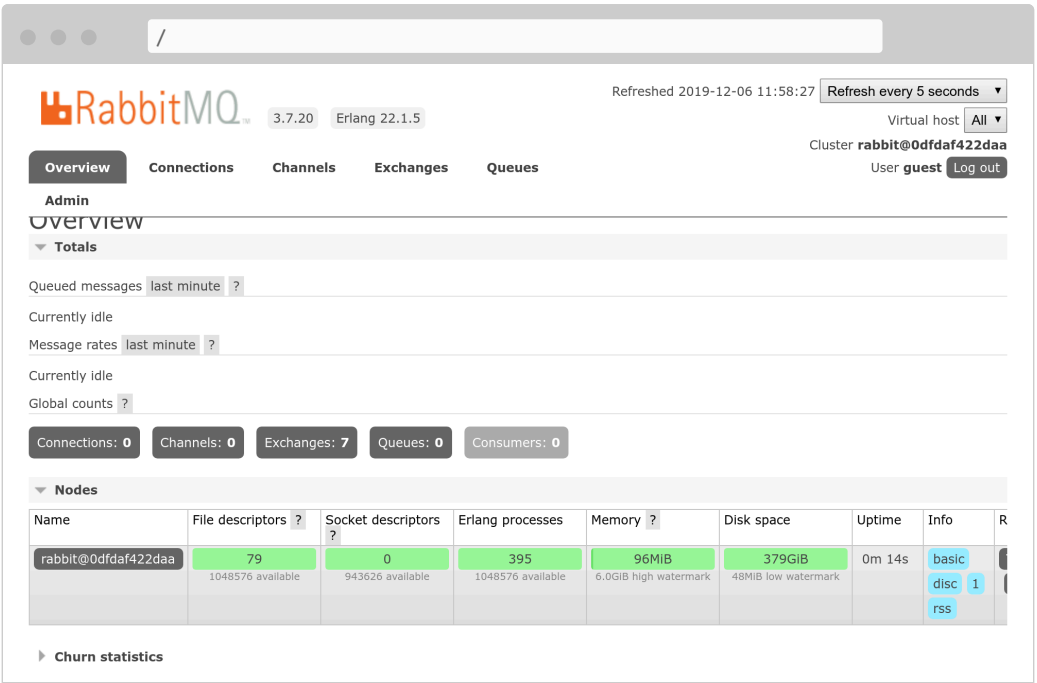
Для просмотра сообщений и очередей в RabbitMQ, откройте веб-интерфейс для его управления:

```
$ symfony open:local:rabbitmq
```

Или запустите из панели отладки:



Используйте `guest/guest` для входа в RabbitMQ:



18.10 Запуск воркеров в фоновом режиме

Вместо того, чтобы запускать получателя каждый раз после отправки

комментария и тут же его останавливать, хотелось бы, чтобы он работал непрерывно, и при этом не нужно было бы открывать слишком много окон или вкладок терминала.

Symfony CLI может управлять такими фоновыми командами или воркерами, если добавить флаг (`-d`) к команде `run`.

Запустите получателя сообщений ещё раз, но теперь уже в фоновом режиме:

```
$ symfony run -d --watch=config,src,templates,vendor symfony console
messenger:consume async
```

При использовании параметра `--watch` Symfony перезапускает команду каждый раз при изменении файловой системы в директориях `config/`, `src/`, `templates/` или `vendor/`.

Не используйте флаг `-vv`, иначе получите дублирующие сообщения в логах команды `server:log` (логированные и консольные сообщения).

Если получатель перестаёт работать по какой-либо причине (недостаточно памяти, баг и т.д.), он будет перезапущен автоматически. Но Symfony CLI не перезапустит получателя в том случае, если получатель выдает ошибку сразу после запуска.

Логи можно просматривать через команду `symfony server:log` вместе с остальными логами, приходящими от PHP, веб-сервера и приложения:

```
$ symfony server:log
```

Команда `server:status` выведет все фоновые воркеры для текущего проекта:

```
$ symfony server:status

Web server listening on https://127.0.0.1:8000
  Command symfony console messenger:consume async running with PID 15774
(watching config/, src/, templates/)
```

Чтобы остановить воркер, нужно либо остановить веб-сервер, либо принудительно завершить процесс по его PID после запуска команды `server:status`:

```
$ kill 15774
```

18.11 Повторная обработка сообщений в случае ошибки

Что, если Akismet будет недоступен во время получения сообщения? Это не отразится на пользователях, оправляющих комментарии, но грозит потерей сообщения и, как следствие, отсутствием проверки на спам.

У компонента Messenger есть механизм повторных попыток в случае возникновения исключений во время обработки сообщений. Давайте его настроим:

```
--- a/config/packages/messenger.yaml
+++ b/config/packages/messenger.yaml
@@ -5,10 +5,17 @@ framework:

         transports:
             # https://symfony.com/doc/current/messenger.html#transport-
configuration
-            async: '%env(RABBITMQ_DSN)%'
-            # failed: 'doctrine://default?queue_name=failed'
+            async:
+                dsn: '%env(RABBITMQ_DSN)%'
+                retry_strategy:
+                    max_retries: 3
+                    multiplier: 2
+
+            failed: 'doctrine://default?queue_name=failed'
             # sync: 'sync://'

+        failure_transport: failed
+
         routing:
             # Route your messages to the transports
             App\Message\CommentMessage: async
```

Если при обработке сообщения возникнет ошибка, получатель повторит попытку ещё 3 раза, а затем остановится. Но вместо удаления получатель перенесёт сообщение в постоянное хранилище — очередь `failed`, которая использует базу данных Doctrine.

Просматривать сообщения, давшие сбой и повторить попытку их обработки можно с помощью следующих команд:

```
$ symfony console messenger:failed:show

$ symfony console messenger:failed:retry
```

18.12 Развёртывание RabbitMQ

Добавим RabbitMQ на продакшен-серверы, включив его в список сервисов:

```
--- a/.symfony/services.yaml
+++ b/.symfony/services.yaml
@@ -5,3 +5,8 @@ db:

 rediscache:
     type: redis:5.0
+
+queue:
+    type: rabbitmq:3.7
+    disk: 1024
+    size: S
```

Также укажите его в конфигурации главного веб-контейнера и включите PHP-модуль `amqp`:

```
--- a/.symfony.cloud.yaml
+++ b/.symfony.cloud.yaml
@@ -4,6 +4,7 @@ type: php:7.3

 runtime:
     extensions:
+        - amqp
         - redis
         - pdo_pgsql
         - apcu
@@ -17,6 +18,7 @@ build:
 relationships:
     database: "db:postgresql"
     redis: "rediscache:redis"
+    rabbitmq: "queue:rabbitmq"
```

```
 web:
     locations:
```

После установки сервиса RabbitMQ на проекте, можно получить доступ к его веб-интерфейсу через открытие туннеля:

```
$ symfony tunnel:open
$ symfony open:remote:rabbitmq

# when done
$ symfony tunnel:close
```

18.13 Выполнение воркеров на SymfonyCloud

Чтобы получать сообщения от RabbitMQ, нам нужно постоянно запускать команду `messenger:consume`. В SymfonyCloud для этой задачи отведена специальная роль — *воркер*:

```
--- a/.symfony.cloud.yaml
+++ b/.symfony.cloud.yaml
@@ -46,3 +46,12 @@ hooks:
         set -x -e

         (>&2 symfony-deploy)
+
+workers:
+    messages:
+        commands:
+            start: |
+                set -x -e
+
+                (>&2 symfony-deploy)
+                php bin/console messenger:consume async -vv --time-limit 3600
--memory-limit=128M
```

Подобно Symfony CLI, SymfonyCloud позволяет управлять перезагрузками и логами.

Чтобы посмотреть логи воркера, воспользуйтесь командой ниже:

```
$ symfony logs --worker=messages all
```

Двигаемся дальше

- *Видеокурс по Messenger на SymfonyCasts*;
- *Архитектура сервисной шины предприятия* и *шаблон CQRS*;
- *Документация по Symfony Messenger*;
- *Документация по RabbitMQ*.

Шаг 19

Управление состоянием с помощью Workflow

Наличие какого-либо состояния у модели — довольно обычное явление. Состояние комментария определяет только антиспам-сервис. Но что, если в будущем у вас появятся больше факторов для изменения состояния?

Возможно, вы захотите дать администратору сайта возможность модерировать все комментарии после того, как они будут проверены антиспам-сервисом. Вот как будет выглядеть этот процесс:

- Начинаем с состояния `submitted`, когда пользователь отправляет комментарий;
- Делегируем антиспам-сервису проанализировать комментарий и переключим его в зависимости от результата в одно из состояний: `potential_spam`, `ham` или `rejected`;
- Если комментарий не был отклонён (то есть он не спам), ожидаем,

пока администратор не решит, достаточно ли комментарий хорош, изменив его состояние на `published` или `rejected`.

Реализация данной логики — не слишком сложная задача. Однако добавление дополнительных правил значительно усложнит эту задачу. Воспользуемся Symfony-компонентом Workflow, чтобы не писать самим логику с нуля:

```
$ symfony composer req workflow
```

19.1 Определение бизнес-процессов

Бизнес-процесс комментария можно описать в конфигурационном файле `config/packages/workflow.yaml`:

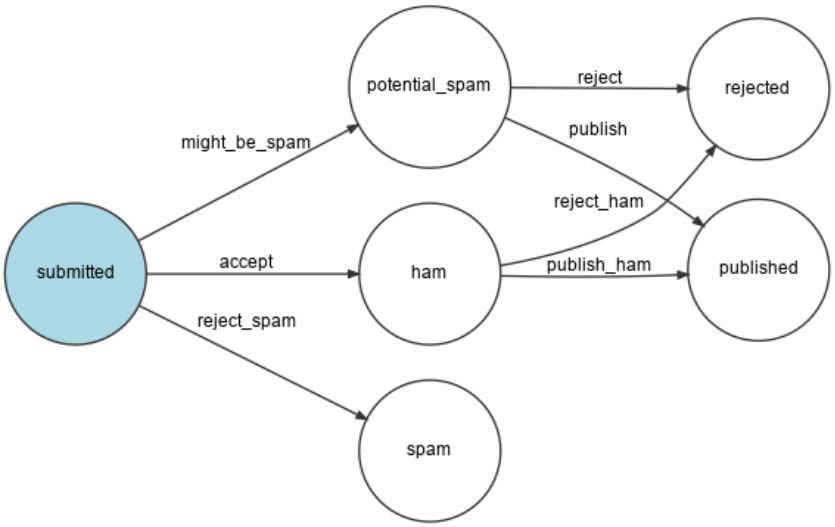
config/packages/workflow.yaml

```
framework:
    workflows:
        comment:
            type: state_machine
            audit_trail:
                enabled: "%kernel.debug%"
            marking_store:
                type: 'method'
                property: 'state'
            supports:
                - App\Entity\Comment
            initial_marking: submitted
            places:
                - submitted
                - ham
                - potential_spam
                - spam
                - rejected
                - published
            transitions:
                accept:
                    from: submitted
                    to:   ham
                might_be_spam:
                    from: submitted
```

```
        to:   potential_spam
    reject_spam:
        from: submitted
        to:   spam
    publish:
        from: potential_spam
        to:   published
    reject:
        from: potential_spam
        to:   rejected
    publish_ham:
        from: ham
        to:   published
    reject_ham:
        from: ham
        to:   rejected
```

Чтобы убедиться в правильности построения этого бизнес-процесса, давайте отобразим его визуально:

```
$ symfony console workflow:dump comment | dot -Tpng -o workflow.png
```



Команда dot является частью утилиты *Graphviz*.

19.2 Использование бизнес-

процессов

Замените текущую логику в обработчике сообщений на новую с использованием определенного ранее бизнес-процесса:

```
--- a/src/MessageHandler/CommentMessageHandler.php
+++ b/src/MessageHandler/CommentMessageHandler.php
@@ -6,19 +6,28 @@ use App\Message\CommentMessage;
 use App\Repository\CommentRepository;
 use App\SpamChecker;
 use Doctrine\ORM\EntityManagerInterface;
+use Psr\Log\LoggerInterface;
 use Symfony\Component\Messenger\Handler\MessageHandlerInterface;
+use Symfony\Component\Messenger\MessageBusInterface;
+use Symfony\Component\Workflow\WorkflowInterface;

 class CommentMessageHandler implements MessageHandlerInterface
 {
     private $spamChecker;
     private $entityManager;
     private $commentRepository;
+    private $bus;
+    private $workflow;
+    private $logger;

-    public function __construct(EntityManagerInterface $entityManager,
SpamChecker $spamChecker, CommentRepository $commentRepository)
+    public function __construct(EntityManagerInterface $entityManager,
SpamChecker $spamChecker, CommentRepository $commentRepository,
MessageBusInterface $bus, WorkflowInterface $commentStateMachine,
LoggerInterface $logger = null)
     {
         $this->entityManager = $entityManager;
         $this->spamChecker = $spamChecker;
         $this->commentRepository = $commentRepository;
+        $this->bus = $bus;
+        $this->workflow = $commentStateMachine;
+        $this->logger = $logger;
     }

     public function __invoke(CommentMessage $message)
@@ -28,12 +37,21 @@ class CommentMessageHandler implements
MessageHandlerInterface
             return;
         }

-        if (2 === $this->spamChecker->getSpamScore($comment, $message-
>getContext())) {
```

```
-            $comment->setState('spam');
-        } else {
-            $comment->setState('published');
-        }
+
+        if ($this->workflow->can($comment, 'accept')) {
+            $score = $this->spamChecker->getSpamScore($comment, $message-
>getContext());
+            $transition = 'accept';
+            if (2 === $score) {
+                $transition = 'reject_spam';
+            } elseif (1 === $score) {
+                $transition = 'might_be_spam';
+            }
+            $this->workflow->apply($comment, $transition);
+            $this->entityManager->flush();

-        $this->entityManager->flush();
+            $this->bus->dispatch($message);
+        } elseif ($this->logger) {
+            $this->logger->debug('Dropping comment message', ['comment' =>
$comment->getId(), 'state' => $comment->getState()]);
+        }
     }
 }
```

Новая логика выглядит следующим образом:

- Если комментарий может перейти в состояние `accept`, значит проверяем сообщение на спам;
- В зависимости от результата проверки, нужно выбрать подходящий переход;
- Вызываем метод `apply()`, чтобы обновить состояние для объекта Comment, который в свою очередь вызывает в этом объекте метод `setState()`;
- Сохраняем данные в базе данных, используя метод `flush()`;
- Повторно отправляем сообщение на шину, чтобы ещё раз запустить бизнес-процесс комментария для определения следующего перехода.

Так как ещё не реализована возможность проверки сообщения администратором, при следующий обработке сообщения в лог запишется следующее: «Dropping comment message».

Перед тем как начать следующую главу, давайте добавим автоматическую проверку:

```
--- a/src/MessageHandler/CommentMessageHandler.php
+++ b/src/MessageHandler/CommentMessageHandler.php
@@ -47,6 +47,9 @@ class CommentMessageHandler implements MessageHandlerInterface
             $this->entityManager->flush();

             $this->bus->dispatch($message);
+        } elseif ($this->workflow->can($comment, 'publish') || $this->workflow-
>can($comment, 'publish_ham')) {
+            $this->workflow->apply($comment, $this->workflow->can($comment,
'publish') ? 'publish' : 'publish_ham');
+            $this->entityManager->flush();
         } elseif ($this->logger) {
             $this->logger->debug('Dropping comment message', ['comment' =>
$comment->getId(), 'state' => $comment->getState()]);
         }
```

Выполните команду `symfony server:log` и добавьте комментарий к любой конференции, чтобы увидеть в терминале, как один за другим происходят переходы состояний.

Двигаемся дальше

- *Бизнес-процессы и конечные автоматы* и что выбрать из них;
- *Документация по Symfony Workflow*.

Шаг 20

Отправка электронной почты администраторам

Для качественной обратной связи, необходимо модерировать все комментарии. Если комментарий находится в состоянии `ham` или `potential_spam`, администратору следует отправить *электронное письмо* с двумя ссылками: для одобрения и для отклонения комментария.

Для начала установите компонент Symfony Mailer:

```
$ symfony composer req mailer
```

20.1 Установка адреса электронной почты для администратора

Для хранения электронной почты администратора используйте параметр контейнера. В демонстрационных целях поместим её в переменную окружения (хотя в реальной проекте обычно это вряд ли понадобится). Чтобы облегчить вставку адреса электронной почты администратора в сервисы, которым он нужен, определите параметр контейнера `bind`:

```
--- a/config/services.yaml
+++ b/config/services.yaml
@@ -4,6 +4,7 @@
 # Put parameters here that don't need to change on each machine where the app
is deployed
 # https://symfony.com/doc/current/best_practices/
configuration.html#application-related-configuration
 parameters:
+    default_admin_email: admin@example.com

 services:
     # default configuration for services in *this* file
@@ -13,6 +14,7 @@ services:
         bind:
             $photoDir: "%kernel.project_dir%/public/uploads/photos"
             $akismetKey: "%env(AKISMET_KEY)%"
+            $adminEmail:
"%env(string:default:default_admin_email:ADMIN_EMAIL)%"

     # makes classes in src/ available to be used as services
     # this creates a service per class whose id is the fully-qualified class
name
```

Переменная окружения может быть «обработана» перед использованием. Здесь мы используем процессор `default`, чтобы получить значение параметра `default_admin_email`, если переменная окружения `ADMIN_EMAIL` не существует.

20.2 Отправка уведомления по электронной почте

Чтобы отправить электронное письмо, вы можете выбирать между несколькими абстракциями класса `Email`: от `Message` (самый низкий уровень) до `NotificationEmail` (самый высокий уровень). Чаще всего, конечно, вы будете использовать класс `Email`, но для внутренних писем предпочтительнее использовать класс `NotificationEmail`.

В обработчике сообщений давайте заменим логику автоматической проверки:

```
--- a/src/MessageHandler/CommentMessageHandler.php
+++ b/src/MessageHandler/CommentMessageHandler.php
@@ -7,6 +7,8 @@ use App\Repository\CommentRepository;
 use App\SpamChecker;
 use Doctrine\ORM\EntityManagerInterface;
 use Psr\Log\LoggerInterface;
+use Symfony\Bridge\Twig\Mime\NotificationEmail;
+use Symfony\Component\Mailer\MailerInterface;
 use Symfony\Component\Messenger\Handler\MessageHandlerInterface;
 use Symfony\Component\Messenger\MessageBusInterface;
 use Symfony\Component\Workflow\WorkflowInterface;
@@ -18,15 +20,19 @@ class CommentMessageHandler implements
MessageHandlerInterface
     private $commentRepository;
     private $bus;
     private $workflow;
+    private $mailer;
+    private $adminEmail;
     private $logger;

-    public function __construct(EntityManagerInterface $entityManager,
SpamChecker $spamChecker, CommentRepository $commentRepository,
MessageBusInterface $bus, WorkflowInterface $commentStateMachine,
LoggerInterface $logger = null)
+    public function __construct(EntityManagerInterface $entityManager,
SpamChecker $spamChecker, CommentRepository $commentRepository,
MessageBusInterface $bus, WorkflowInterface $commentStateMachine,
MailerInterface $mailer, string $adminEmail, LoggerInterface $logger = null)
     {
         $this->entityManager = $entityManager;
         $this->spamChecker = $spamChecker;
         $this->commentRepository = $commentRepository;
         $this->bus = $bus;
         $this->workflow = $commentStateMachine;
```

```
+        $this->mailer = $mailer;
+        $this->adminEmail = $adminEmail;
         $this->logger = $logger;
     }

@@ -51,8 +57,13 @@ class CommentMessageHandler implements
MessageHandlerInterface

             $this->bus->dispatch($message);
         } elseif ($this->workflow->can($comment, 'publish') || $this->workflow-
>can($comment, 'publish_ham')) {
-            $this->workflow->apply($comment, $this->workflow->can($comment,
'publish') ? 'publish' : 'publish_ham');
-            $this->entityManager->flush();
+            $this->mailer->send((new NotificationEmail())
+                ->subject('New comment posted')
+                ->htmlTemplate('emails/comment_notification.html.twig')
+                ->from($this->adminEmail)
+                ->to($this->adminEmail)
+                ->context(['comment' => $comment])
+            );
         } elseif ($this->logger) {
             $this->logger->debug('Dropping comment message', ['comment' =>
$comment->getId(), 'state' => $comment->getState()]);
         }
```

`MailerInterface` является основной точкой входа и позволяет отправлять электронную почту с помощью метода `send()`.

Чтобы отправить электронное письмо, нам нужен отправитель (заголовок `From`/`Sender`). Вместо того, чтобы устанавливать его явно в экземпляре Email, определите его глобально:

```
--- a/config/packages/mailer.yaml
+++ b/config/packages/mailer.yaml
@@ -1,3 +1,5 @@
 framework:
     mailer:
         dsn: '%env(MAILER_DSN)%'
+        envelope:
+            sender: "%env(string:default:default_admin_email:ADMIN_EMAIL)%"
```

20.3 Расширение шаблона

электронной почты для уведомлений

Шаблон электронной почты для уведомлений наследуется от стандартного шаблона уведомлений в Symfony:

templates/emails/comment_notification.html.twig

```
{% extends '@email/default/notification/body.html.twig' %}

{% block content %}
    Author: {{ comment.author }}<br />
    Email: {{ comment.email }}<br />
    State: {{ comment.state }}<br />

    <p>
        {{ comment.text }}
    </p>
{% endblock %}

{% block action %}
    <spacer size="16"></spacer>
    <button href="{{ url('review_comment', { id: comment.id })
}}">Accept</button>
    <button href="{{ url('review_comment', { id: comment.id, reject: true })
}}">Reject</button>
{% endblock %}
```

Шаблон переопределяет несколько блоков, чтобы изменить текст письма и добавить ссылки для одобрения или отклонения комментария. Любой некорректный параметр для маршрута добавляется в качестве параметра строки запроса (адрес для отклонения комментария будет выглядеть так: `/admin/comment/review/42?reject=true`).

Шаблон по умолчанию `NotificationEmail` использует *Inky* вместо HTML для описания разметки электронных писем. Этот шаблонизатор помогает создавать адаптивные электронные письма, совместимые со всеми популярными почтовыми клиентами.

Для максимальной совместимости с программами для чтения электронной почты, базовый макет уведомлений уже по умолчанию использует встроенные стили (с помощью пакета CSS inliner).

Эти две возможности являются частью дополнительных расширений

Twig, которые необходимо установить:

```
$ symfony composer req twig/cssinliner-extra twig/inky-extra
```

20.4 Создание абсолютных адресов с помощью команды

В электронных письмах абсолютные адреса (со схемой и хостом) создавайте с помощью `url()` вместо `path()`.

Электронное письмо отправляется из обработчика сообщений в контексте консоли. Создание абсолютных адресов из браузера проще, поскольку мы знаем схему и домен текущей страницы. Это не относится к консоли.

Явно определите доменное имя и схему:

```
--- a/config/services.yaml
+++ b/config/services.yaml
@@ -5,6 +5,11 @@
 # https://symfony.com/doc/current/best_practices/
configuration.html#application-related-configuration
 parameters:
     default_admin_email: admin@example.com
+    default_domain: '127.0.0.1'
+    default_scheme: 'http'
+
+    router.request_context.host:
'%env(default:default_domain:SYMFONY_DEFAULT_ROUTE_HOST)%'
+    router.request_context.scheme:
'%env(default:default_scheme:SYMFONY_DEFAULT_ROUTE_SCHEME)%'

 services:
     # default configuration for services in *this* file
```

Переменные окружения `SYMFONY_DEFAULT_ROUTE_HOST` и `SYMFONY_DEFAULT_ROUTE_PORT` автоматически устанавливаются локально при использовании CLI-команды `symfony` и определяются на основе конфигурации в SymfonyCloud.

20.5 Подключение маршрута к контроллеру

Маршрут `review_comment` пока не существует, давайте создадим административный контроллер для его обработки:

src/Controller/AdminController.php

```
namespace App\Controller;

use App\Entity\Comment;
use App\Message\CommentMessage;
use Doctrine\ORM\EntityManagerInterface;
use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Request;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Messenger\MessageBusInterface;
use Symfony\Component\Routing\Annotation\Route;
use Symfony\Component\Workflow\Registry;
use Twig\Environment;

class AdminController extends AbstractController
{
    private $twig;
    private $entityManager;
    private $bus;

    public function __construct(Environment $twig, EntityManagerInterface
$entityManager, MessageBusInterface $bus)
    {
        $this->twig = $twig;
        $this->entityManager = $entityManager;
        $this->bus = $bus;
    }

    /**
     * @Route("/admin/comment/review/{id}", name="review_comment")
     */
    public function reviewComment(Request $request, Comment $comment, Registry
$registry)
    {
        $accepted = !$request->query->get('reject');

        $machine = $registry->get($comment);
        if ($machine->can($comment, 'publish')) {
            $transition = $accepted ? 'publish' : 'reject';
        } elseif ($machine->can($comment, 'publish_ham')) {
```

```
            $transition = $accepted ? 'publish_ham' : 'reject_ham';
        } else {
            return new Response('Comment already reviewed or not in the right
state.');
        }

        $machine->apply($comment, $transition);
        $this->entityManager->flush();

        if ($accepted) {
            $this->bus->dispatch(new CommentMessage($comment->getId()));
        }

        return $this->render('admin/review.html.twig', [
            'transition' => $transition,
            'comment' => $comment,
        ]);
    }
}
```

Адрес проверки комментария начинается с `/admin/`, чтобы защитить его с помощью файрвола, определённого на предыдущем шаге. Администратор должен пройти аутентификацию для доступа к этому ресурсу.

Вместо создания экземпляра `Response` мы использовали короткий метод `render()` из базового класса контроллера `AbstractController`.

Когда проверка комментария проведена, короткое сообщение поблагодарит администратора за хорошую работу:

templates/admin/review.html.twig

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

{% block body %}
    <h2>Comment reviewed, thank you!</h2>

    <p>Applied transition: <strong>{{ transition }}</strong></p>
    <p>New state: <strong>{{ comment.state }}</strong></p>
{% endblock %}
```

20.6 Использование перехватчика почты

Вместо того, чтобы использовать «настоящий» SMTP-сервер или сторонний провайдер для отправки электронной почты, давайте применим перехватчик почты. Он представляет собой SMTP-сервер, который не занимается доставкой электронной почты, а показывает её в веб-интерфейсе:

```
--- a/docker-compose.yaml
+++ b/docker-compose.yaml
@@ -16,3 +16,7 @@ services:
     rabbitmq:
         image: rabbitmq:3.7-management
         ports: [5672, 15672]
+
+    mailcatcher:
+        image: schickling/mailcatcher
+        ports: [1025, 1080]
```

Остановите и перезапустите контейнеры, чтобы добавить перехватчик почты:

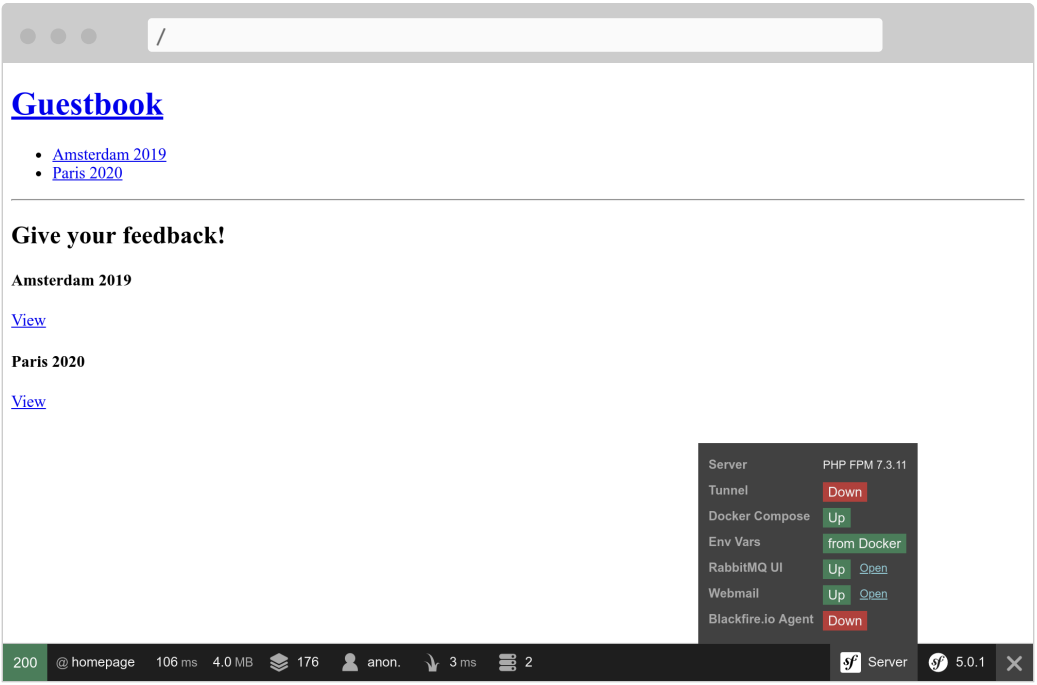
```
$ docker-compose stop
$ docker-compose up -d
```

20.7 Доступ к почтовому веб-сервису

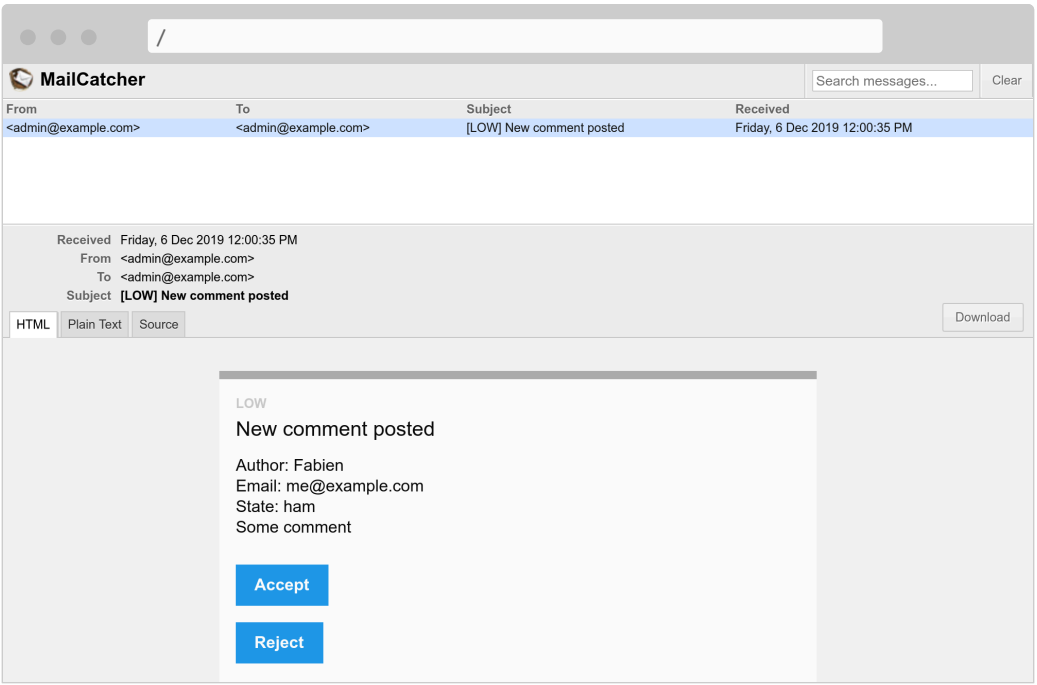
Вы можете открыть почтовый веб-сервис из терминала:

```
$ symfony open:local:webmail
```

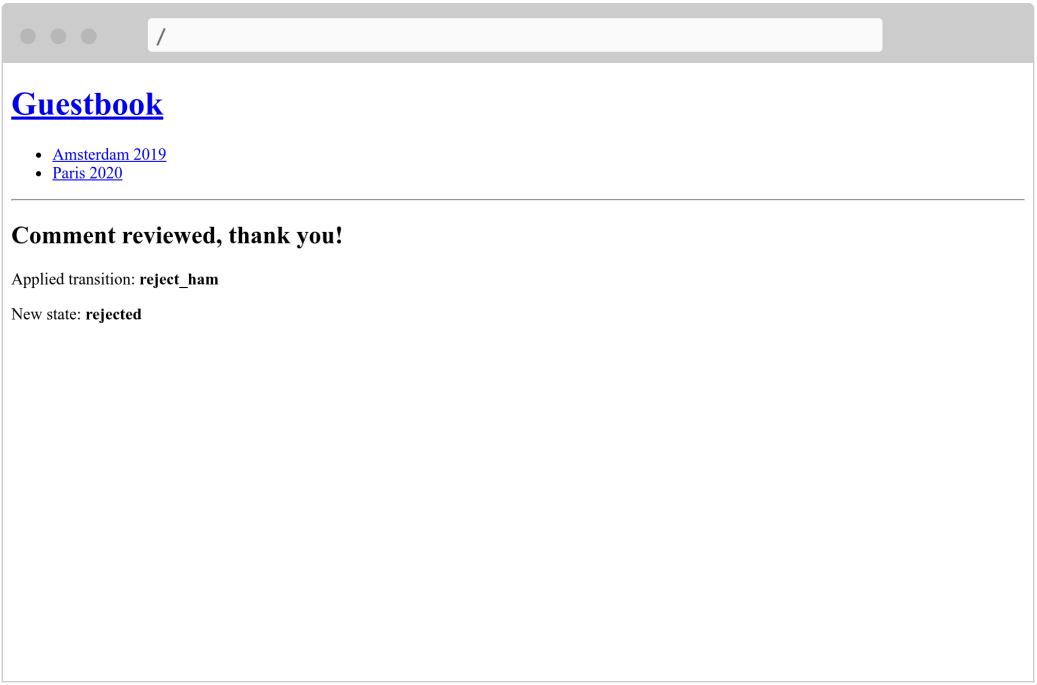
Или сделать это из панели отладки:



Оставив комментарий, вы сможете посмотреть электронное письмо в почтовом приложении:



Нажмите на заголовок электронного письма в почтовом клиенте и одобрите или отклоните комментарий по своему усмотрению:



Проверьте логи с помощью команды `server:log`, если нет видимого результата.

20.8 Управление долго выполняющимися скриптами

Для начала нужно пояснить особенности работы долго выполняющихся скриптов. В отличие от модели PHP, используемой для HTTP, где каждый запрос начинается с чистого состояния, потребитель сообщений работает непрерывно в фоновом режиме. Каждая обработка сообщения наследует текущее состояние, включая кеш памяти. Чтобы избежать каких-либо проблем с Doctrine, его менеджеры сущностей автоматически очищаются после обработки каждого сообщения. Учитывайте это при разработке собственных сервисов.

20.9 Асинхронная отправка электронной почты

Отправка электронной почты в обработчике сообщений может занять

некоторое время. Может даже выбросить исключение. В этом случае сообщение будет отправлено повторно. Но вместо того, чтобы повторять обработку сообщения комментария снова, лучше попробовать ещё раз только отправить электронное письмо.

Мы уже знаем, как это сделать: отправить сообщение электронной почты на шину.

Экземпляр `MailerInterface` выполняет тяжелую работу: если шина определена, он посылает на неё сообщения электронной почты, а не отправляет их. Никакие изменения в вашем коде не требуются.

Поскольку мы не настроили очередь, шина отправляет сообщения электронной почты синхронно. Давайте снова используем RabbitMQ:

```
--- a/config/packages/messenger.yaml
+++ b/config/packages/messenger.yaml
@@ -19,3 +19,4 @@ framework:
         routing:
             # Route your messages to the transports
             App\Message\CommentMessage: async
+            Symfony\Component\Mailer\Messenger\SendEmailMessage: async
```

Даже если мы используем один и тот же брокер (RabbitMQ) для доставки сообщений комментариев и электронной почты, это со временем может измениться. Например, вы можете решить использовать другую систему обмена сообщениями, чтобы управлять сообщениями с различными приоритетами. Использование других брокеров также позволяет вам использовать разные рабочие машины, обрабатывающие разные виды сообщений. Функциональность компонента Messenger очень гибкая и настраиваемая в соответствии с вашими потребностями.

20.10 Тестирование электронной почты

Есть множество способов протестировать электронную почту.

Вы можете написать модульные тесты, если создадите класс для каждого электронного письма (например, путем наследования `Email` или `TemplatedEmail`).

Однако чаще всего вам предстоит писать функциональные тесты, которые проверяют, что определённые действия запускают отправку электронных писем, и, возможно, проверяют само содержимое динамических писем.

В Symfony есть проверки, которые облегчают написание подобных тестов:

```
public function testMailerAssertions()
{
    $client = static::createClient();
    $client->request('GET', '/');

    $this->assertEmailCount(1);
    $event = $this->getMailerEvent(0);
    $this->assertEmailIsQueued($event);

    $email = $this->getMailerMessage(0);
    $this->assertEmailHeaderSame($email, 'To', 'fabien@example.com');
    $this->assertEmailTextBodyContains($email, 'Bar');
    $this->assertEmailAttachmentCount($email, 1);
}
```

Эти проверки работают, когда электронная почта отправляется синхронно или асинхронно.

20.11 Отправка электронной почты в SymfonyCloud

Для SymfonyCloud нет специальной конфигурации. Все учётные записи имеют аккаунт на сервисе Sendgrid, который автоматически используется для отправки электронных писем.

Вам всё ещё необходимо обновить конфигурацию SymfonyCloud, чтобы включить PHP-модуль `xsl`, необходимый для Inky:

```
--- a/.symfony.cloud.yaml
+++ b/.symfony.cloud.yaml
@@ -4,6 +4,7 @@ type: php:7.3

 runtime:
     extensions:
+        - xsl
```

```
- amqp
- redis
- pdo_pgsql
```

Из соображений безопасности электронные письма по умолчанию отправляются *только* из ветки `master`. Включите SMTP явно, если вы знаете, что делаете:

```
$ symfony env:setting:set email on
```

Двигаемся дальше

- *Обучающий видеокурс по Mailer на SymfonyCasts*;
- *Документация по шаблонизатору Inky*;
- *Процессоры переменных окружения*;
- *Документация по Mailer на сайте Symfony*;
- *Документация по настройке электронной почты в SymfonyCloud*.

Шаг 21

Повышение производительности с помощью кеширования

По мере роста популярности могут возникнуть проблемы с производительностью. Вот только пара типичных примеров: отсутствие индексов в базе данных или огромное количество SQL-запросов на страницу. Вам нечего боятся при пустой базе данных, но с ростом трафика и объема данных могут начаться проблемы.

21.1 Добавление заголовков кеширования HTTP

HTTP-кеширование — отличный способ увеличить производительность для пользователей с минимальными усилиями. Для большей производительности в продакшене используйте кеширующий обратный прокси-сервер и пограничный *CDN*.

Давайте закешируем главную страницу на один час:

```
--- a/src/Controller/ConferenceController.php
+++ b/src/Controller/ConferenceController.php
@@ -37,9 +37,12 @@ class ConferenceController extends AbstractController
      */
     public function index(ConferenceRepository $conferenceRepository)
     {
-        return new Response($this->twig->render('conference/index.html.twig', [
+        $response = new Response($this->twig->render('conference/
index.html.twig', [
             'conferences' => $conferenceRepository->findAll(),
         ]));
+        $response->setSharedMaxAge(3600);
+
+        return $response;
     }

     /**
```

Метод `setSharedMaxAge()` устанавливает срок действия кеша для обратных прокси-серверов. Используйте метод `setMaxAge()`, чтобы управлять кешем браузера. Время указывается в секундах (1 час = 60 минут = 3600 секунд).

Кеширование страницы конференции является менее тривиальной задачей, поскольку данные на ней более динамичны. В любой момент любой желающий может добавить комментарий, и никто не хочет ждать целый час, чтобы его увидеть на сайте. В таких случаях используйте *валидацию кеширования HTTP*.

21.2 Активация ядра HTTP-кеширования в Symfony

Для тестирования стратегии HTTP-кеширования используйте обратный прокси-сервер Symfony:

```
--- a/public/index.php
+++ b/public/index.php
@@ -1,6 +1,7 @@
 <?php

 use App\Kernel;
+use Symfony\Bundle\FrameworkBundle\HttpCache\HttpCache;
 use Symfony\Component\ErrorHandler\Debug;
 use Symfony\Component\HttpFoundation\Request;

@@ -21,6 +22,11 @@ if ($trustedHosts = $_SERVER['TRUSTED_HOSTS'] ??
$_ENV['TRUSTED_HOSTS'] ?? false
 }

 $kernel = new Kernel($_SERVER['APP_ENV'], (bool) $_SERVER['APP_DEBUG']);
+
+if ('dev' === $kernel->getEnvironment()) {
+    $kernel = new HttpCache($kernel);
+}
+
 $request = Request::createFromGlobals();
 $response = $kernel->handle($request);
 $response->send();
```

Являясь полноценным обратным прокси-сервером HTTP, он дополнительно (с помощью класса `HttpCache`) добавляет полезную отладочную информацию в виде HTTP-заголовков. Она может сильно помочь в проверке заголовков кеширования, которые мы установили.

Проверим, как работает кеширование на главной странице:

```
$ curl -s -I -X GET https://127.0.0.1:8000/


HTTP/2 200
age: 0
cache-control: public, s-maxage=3600
content-type: text/html; charset=UTF-8
date: Mon, 28 Oct 2019 08:11:57 GMT
```

```
x-content-digest:
en63cef7045fe418859d73668c2703fb1324fcc0d35b21d95369a9ed1aca48e73e
x-debug-token: 9eb25a
x-debug-token-link: https://127.0.0.1:8000/_profiler/9eb25a
x-robots-tag: noindex
x-symfony-cache: GET /: miss, store
content-length: 50978
```

Для самого первого запроса кеширующий сервер говорит нам, что ответ на запрос не был закеширован (кеш-промах, или `miss`) и что он сохранил полученный ответ (`store`) для будущих запросов. Посмотрите на заголовок `cache-control`, чтобы увидеть настроенную стратегию кеширования.

Для последующих запросов ответ будет закеширован (при этом `age` также обновился):

```
HTTP/2 200
age: 143
cache-control: public, s-maxage=3600
content-type: text/html; charset=UTF-8
date: Mon, 28 Oct 2019 08:11:57 GMT
x-content-digest:
en63cef7045fe418859d73668c2703fb1324fcc0d35b21d95369a9ed1aca48e73e
x-debug-token: 9eb25a
x-debug-token-link: https://127.0.0.1:8000/_profiler/9eb25a
x-robots-tag: noindex
x-symfony-cache: GET /: fresh
content-length: 50978
```

21.3 Кеширование SQL-запросов при помощи ESI

Обработчик `TwigEventSubscriber` внедряет глобальную переменную в Twig для всех объектов конференции. Это происходит для каждой отдельной страницы сайта. Вероятно, это отличное место для оптимизации.

Вы не добавляете новые конференции каждый день, однако код запрашивает одни и те же данные из базы снова и снова.

Возможно, нам стоит закешировать названия конференций и слаги с

помощью Symfony Cache, но там, где это возможно, я предпочёл бы использовать HTTP-кеширование.

Когда вам нужно закешировать фрагмент страницы, не загружайте его в текущем HTTP-запросе, а вместо этого создайте *подзапрос* с ним. *ESI* идеально подходит для решения такой задачи. ESI — это способ вставить результат одного HTTP-запроса в другой.

Создайте контроллер, возвращающий только HTML-фрагмент с конференциями:

```
--- a/src/Controller/ConferenceController.php
+++ b/src/Controller/ConferenceController.php
@@ -45,6 +45,16 @@ class ConferenceController extends AbstractController
         return $response;
     }

+    /**
+     * @Route("/conference_header", name="conference_header")
+     */
+    public function conferenceHeader(ConferenceRepository
$conferenceRepository)
+    {
+        return new Response($this->twig->render('conference/header.html.twig',
[
+            'conferences' => $conferenceRepository->findAll(),
+        ]));
+    }
+
     /**
      * @Route("/conference/{slug}", name="conference")
      */
```

Создайте соответствующий шаблон:

templates/conference/header.html.twig

```
<ul>
    {% for conference in conferences %}
        <li><a href="{{ path('conference', { slug: conference.slug }) }}">{{
conference }}</a></li>
    {% endfor %}
</ul>
```

Посетите `/conference/header`, чтобы проверить, что всё работает.

Раскроем тайну фокуса! Обновите шаблон Twig, чтобы вызывать только что созданный контроллер:

```
--- a/templates/base.html.twig
+++ b/templates/base.html.twig
@@ -8,11 +8,7 @@
     <body>
         <header>
             <h1><a href="{{ path('homepage') }}">Guestbook</a></h1>
-            <ul>
-            {% for conference in conferences %}
-                <li><a href="{{ path('conference', { slug: conference.slug })
}}">{{ conference }}</a></li>
-            {% endfor %}
-            </ul>
+            {{ render(path('conference_header')) }}
             <hr />
         </header>
         {% block body %}{% endblock %}
```

И *вуаля*. Перезагрузите страницу в браузере — на сайте по-прежнему будет показано то же самое.

Воспользуйтесь панелью «Request / Response» в профилировщике Symfony, чтобы узнать больше об основном запросе и его подзапросах.

Теперь каждый раз, когда вы открываете страницу в браузере, выполняются два HTTP-запроса: один с конференциями для шапки сайта, другой — для главной страницы. Вы только что ухудшили производительность. Поздравляю!

HTTP-запрос для получения списка конференций сейчас выполняется изнутри в Symfony, поэтому HTTP-вызова нет. Помимо всего, это означает, что мы не сможем извлечь выгоду от заголовков HTTP-кеширования.

Поэтому нужно превратить этот запрос в «настоящий» HTTP с помощью ESI.

Во-первых, включите поддержку ESI:

```
--- a/config/packages/framework.yaml
+++ b/config/packages/framework.yaml
@@ -10,7 +10,7 @@ framework:
         cookie_secure: auto
         cookie_samesite: lax
```

```
-    #esi: true
+    esi: true
     #fragments: true
     php_errors:
         log: true
```

Далее замените `render` на `render_esi`:

```
--- a/templates/base.html.twig
+++ b/templates/base.html.twig
@@ -8,7 +8,7 @@
     <body>
         <header>
             <h1><a href="{{ path('homepage') }}">Guestbook</a></h1>
-            {{ render(path('conference_header')) }}
+            {{ render_esi(path('conference_header')) }}
             <hr />
         </header>
         {% block body %}{% endblock %}
```

Symfony автоматически активирует поддержку ESI, если обратный прокси-сервер умеет работать с ESI (в противном случае переходит к синхронной обработке подзапросов).

Поскольку обратный прокси-сервер Symfony поддерживает ESI, давайте проверим его логи (сначала очистите кеш, как описано в следующем разделе):

```
$ curl -s -I -X GET https://127.0.0.1:8000/


HTTP/2 200
age: 0
cache-control: must-revalidate, no-cache, private
content-type: text/html; charset=UTF-8
date: Mon, 28 Oct 2019 08:20:05 GMT
expires: Mon, 28 Oct 2019 08:20:05 GMT
x-content-digest:
en4dd846a34dcd757eb9fd277f43220effd28c00e4117bed41af7f85700eb07f2c
x-debug-token: 719a83
x-debug-token-link: https://127.0.0.1:8000/_profiler/719a83
x-robots-tag: noindex
x-symfony-cache: GET /: miss, store; GET /conference_header: miss
content-length: 50978
```

Обновите страницу несколько раз: страница по адресу `/` кешируется,

а по /conference_header — нет. Неплохой результат: вся страница кешируется, при этом динамическая часть всё ещё присутствует.

Однако это не то, что нам нужно. Давайте закешируем страницу с конференциями на один час, независимо от всего остального:

```
--- a/src/Controller/ConferenceController.php
+++ b/src/Controller/ConferenceController.php
@@ -50,9 +50,12 @@ class ConferenceController extends AbstractController
      */
     public function conferenceHeader(ConferenceRepository
$conferenceRepository)
     {
-        return new Response($this->twig->render('conference/header.html.twig',
[
+        $response = new Response($this->twig->render('conference/
header.html.twig', [
             'conferences' => $conferenceRepository->findAll(),
         ]));
+        $response->setSharedMaxAge(3600);
+
+        return $response;
     }

     /**
```

Кеширование теперь включено для обоих запросов:

```
$ curl -s -I -X GET https://127.0.0.1:8000/


HTTP/2 200
age: 613
cache-control: public, s-maxage=3600
content-type: text/html; charset=UTF-8
date: Mon, 28 Oct 2019 07:31:24 GMT
x-content-digest:
en15216b0803c7851d3d07071473c9f6a3a3360c6a83ccb0e550b35d5bc484bbd2
x-debug-token: cfb0e9
x-debug-token-link: https://127.0.0.1:8000/_profiler/cfb0e9
x-robots-tag: noindex
x-symfony-cache: GET /: fresh; GET /conference_header: fresh
content-length: 50978
```

Заголовок x-symfony-cache содержит два элемента: основной запрос на / и подзапрос (conference_header через ESI). Ответы на оба запроса находятся в кеше (fresh).

Стратегия кеширования может различаться между главной страницей и теми, которые загружаются через ESI. Допустим, у нас есть редко обновляемая страница «О нас», то мы можем закешировать её на неделю, и при этом по-прежнему обновлять страницу с конференциями каждый час.

Удалите обработчик, так как он нам больше не нужен:

```
$ rm src/EventSubscriber/TwigEventSubscriber.php
```

21.4 Очистка HTTP-кеша для тестирования

Тестирование сайта в браузере или с помощью автотестов становится немного сложнее при наличии слоя кеширования.

Вы можете вручную очистить весь HTTP-кеш, если удалите директорию `var/cache/dev/http_cache/`:

```
$ rm -rf var/cache/dev/http_cache/
```

Такой подход неудобен и не эффективен, если вам нужно инвалидировать кеш только определённых URL-адресов или интегрировать инвалидацию кеша в функциональные тесты. Давайте добавим HTTP-маршрут, который будет доступен только для администратора и при помощи которого можно будет сбросить кеш для некоторых URL-адресов:

```
--- a/src/Controller/AdminController.php
+++ b/src/Controller/AdminController.php
@@ -6,8 +6,10 @@ use App\Entity\Comment;
 use App\Message\CommentMessage;
 use Doctrine\ORM\EntityManagerInterface;
 use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
+use Symfony\Bundle\FrameworkBundle\HttpCache\HttpCache;
 use Symfony\Component\HttpFoundation\Request;
 use Symfony\Component\HttpFoundation\Response;
+use Symfony\Component\HttpKernel\KernelInterface;
 use Symfony\Component\Messenger\MessageBusInterface;
 use Symfony\Component\Routing\Annotation\Route;
 use Symfony\Component\Workflow\Registry;
```

```
@@ -54,4 +56,19 @@ class AdminController extends AbstractController
             'comment' => $comment,
         ]);
     }
+
+    /**
+     * @Route("/admin/http-cache/{uri<.*>}", methods={"PURGE"})
+     */
+    public function purgeHttpCache(KernelInterface $kernel, Request $request,
string $uri)
+    {
+        if ('prod' === $kernel->getEnvironment()) {
+            return new Response('KO', 400);
+        }
+
+        $store = (new class($kernel) extends HttpCache {})->getStore();
+        $store->purge($request->getSchemeAndHttpHost().'/'.$uri);
+
+        return new Response('Done');
+    }
 }
```

Новый контроллер обрабатывает только HTTP-метод `PURGE`. Этот метод не входит в стандарт HTTP, но широко используется для инвалидации кеша.

По умолчанию параметры маршрута не могут содержать сами символы `/`, так как они разделяют сегменты URL-адреса. Но вы можете обойти это ограничение для последнего параметра маршрута, как это было сделано в `uri`, если зададите для него нужное вам выражение (`.*`).

Получение объекта `HttpCache` выглядит немного странным — мы используем анонимный класс, так как получить доступ к «настоящему» невозможно. Объект `HttpCache` оборачивает ядро фреймворка, которое ничего не знает о слое кеширования, но так и должно быть.

Инвалидируйте главную страницу и страницу с конференциями, выполнив следующие cURL-команды:

```
$ curl -I -X PURGE -u admin:admin `symfony var:export
SYMFONY_DEFAULT_ROUTE_URL`/admin/http-cache/
$ curl -I -X PURGE -u admin:admin `symfony var:export
SYMFONY_DEFAULT_ROUTE_URL`/admin/http-cache/conference_header
```

Подкоманда `symfony var:export SYMFONY_DEFAULT_ROUTE_URL` возвращает

текущий URL-адрес локального веб-сервера.

Контроллер не имеет имени маршрута, так как он никогда не будет использован в коде.

21.5 Группировка схожих маршрутов по префиксу

Два маршрута в административном контроллере имеют одинаковый префикс `/admin`. Вместо того, чтобы повторять его для всех маршрутов, улучшим маршруты таким образом, чтобы префикс был определён в самом классе:

```
--- a/src/Controller/AdminController.php
+++ b/src/Controller/AdminController.php
@@ -15,6 +15,9 @@ use Symfony\Component\Routing\Annotation\Route;
 use Symfony\Component\Workflow\Registry;
 use Twig\Environment;

+/**
+ * @Route("/admin")
+ */
 class AdminController extends AbstractController
 {
     private $twig;
@@ -29,7 +32,7 @@ class AdminController extends AbstractController
     }

     /**
-     * @Route("/admin/comment/review/{id}", name="review_comment")
+     * @Route("/comment/review/{id}", name="review_comment")
      */
     public function reviewComment(Request $request, Comment $comment, Registry
$registry)
     {
@@ -58,7 +61,7 @@ class AdminController extends AbstractController
     }

     /**
-     * @Route("/admin/http-cache/{uri<.*>}", methods={"PURGE"})
+     * @Route("/http-cache/{uri<.*>}", methods={"PURGE"})
      */
     public function flushHttpCache(KernelInterface $kernel, Request $request,
```

```
string $uri)
    {
```

21.6 Кеширование операций с большим потреблением ресурсов ЦПУ/памяти

У нас на сайте нет алгоритмов, интенсивно использующих ЦПУ или память. Чтобы рассмотреть *локальное кеширование*, давайте создадим команду, которая будет отображать шаг, над которым мы сейчас работаем (если точнее, то мы выведем имя Git-тега, прикреплённого к текущему коммиту).

Компонент Symfony Process позволяет выполнить команду и получить результат (стандартный вывод и вывод ошибок). Установите его:

```
$ symfony composer req process
```

Реализуйте команду:

src/Command/StepInfoCommand.php

```
namespace App\Command;

use Symfony\Component\Console\Command\Command;
use Symfony\Component\Console\Input\InputInterface;
use Symfony\Component\Console\Output\OutputInterface;
use Symfony\Component\Process\Process;

class StepInfoCommand extends Command
{
    protected static $defaultName = 'app:step:info';

    protected function execute(InputInterface $input, OutputInterface $output):
int
    {
        $process = new Process(['git', 'tag', '-l', '--points-at', 'HEAD']);
        $process->mustRun();
        $output->write($process->getOutput());

        return 0;
```

```
    }
}
```

Вы можете выполнить `make:command` для создания команды:

```
$ symfony console make:command app:step:info
```

Как насчёт того, чтобы закешировать результат на несколько минут? Установите Symfony-компонент Cache:

```
$ symfony composer req cache
```

А затем оберните код логикой кеширования:

```
--- a/src/Command/StepInfoCommand.php
+++ b/src/Command/StepInfoCommand.php
@@ -6,16 +6,31 @@ use Symfony\Component\Console\Command\Command;
 use Symfony\Component\Console\Input\InputInterface;
 use Symfony\Component\Console\Output\OutputInterface;
 use Symfony\Component\Process\Process;
+use Symfony\Contracts\Cache\CacheInterface;

 class StepInfoCommand extends Command
 {
     protected static $defaultName = 'app:step:info';

+    private $cache;
+
+    public function __construct(CacheInterface $cache)
+    {
+        $this->cache = $cache;
+
+        parent::__construct();
+    }
+
     protected function execute(InputInterface $input, OutputInterface
$output): int
     {
-        $process = new Process(['git', 'tag', '-l', '--points-at', 'HEAD']);
-        $process->mustRun();
-        $output->write($process->getOutput());
+        $step = $this->cache->get('app.current_step', function ($item) {
+            $process = new Process(['git', 'tag', '-l', '--points-at',
'HEAD']);
+            $process->mustRun();
```

```
+            $item->expiresAfter(30);
+
+            return $process->getOutput();
+        });
+        $output->writeln($step);

         return 0;
     }
```

Теперь процесс вызывается только в том случае, если элемент `app.current_step` отсутствует в кеше.

21.7 Профилирование и сравнение производительности

Никогда не используйте кеширование вслепую. Имейте в виду, что использование кеширования добавляет ещё один уровень сложности. Сложно предсказать, что будет работать быстро, а что — медленно. В итоге вы можете оказаться в ситуации, когда кеширование замедлит работу вашего приложения.

Всегда измеряйте результат от использования кеширования с помощью инструментов профилировщика, например, *Blackfire*.

Обратитесь к шагу про производительность, чтобы узнать больше о том, как использовать Blackfire для тестирования перед развёртыванием.

21.8 Настройка кеширующего обратного прокси-сервера в продакшене

Не используйте обратный прокси-сервер Symfony в продакшене. Всегда отдавайте приоритет таким обратным прокси-серверам, как Varnish, либо используйте коммерческий CDN.

Добавьте Varnish в сервисы SymfonyCloud:

```
--- a/.symfony/services.yaml
+++ b/.symfony/services.yaml
@@ -7,3 +7,12 @@ queue:
     type: rabbitmq:3.5
     disk: 1024
     size: S
+
+varnish:
+    type: varnish:6.0
+    relationships:
+        application: 'app:http'
+    configuration:
+        vcl: !include
+            type: string
+            path: config.vcl
```

Используйте Varnish в качестве основной точки входа в маршруты:

```
--- a/.symfony/routes.yaml
+++ b/.symfony/routes.yaml
@@ -1,2 +1,2 @@
-"https://{all}/": { type: upstream, upstream: "app:http" }
+"https://{all}/": { type: upstream, upstream: "varnish:http", cache: {
enabled: false } }
 "http://{all}/": { type: redirect, to: "https://{all}/" }
```

Наконец, создайте файл `config.vcl` для конфигурации Varnish:

.symfony/config.vcl

```
sub vcl_recv {
    set req.backend_hint = application.backend();
}
```

21.9 Включение поддержки ESI в Varnish

Поддержка ESI в Varnish должна быть включена явно для каждого запроса. Чтобы сделать это для всех запросов сразу, Symfony использует стандартные заголовки `Surrogate-Capability` и `Surrogate-Control`:

.symfony/config.vcl

```
sub vcl_recv {
    set req.backend_hint = application.backend();
    set req.http.Surrogate-Capability = "abc=ESI/1.0";
}

sub vcl_backend_response {
    if (beresp.http.Surrogate-Control ~ "ESI/1.0") {
        unset beresp.http.Surrogate-Control;
        set beresp.do_esi = true;
    }
}
```

21.10 Инвалидация кеша Varnish

Инвалидация кеша в продакшене, скорее всего, никогда не понадобится, кроме экстренных случаев и, возможно, только в ветках, отличных от `master`. Если вам приходится часто чистить кеш, то обычно это означает, что стратегия кеширования должна быть скорректирована (путём уменьшения TTL или с помощью использования стратегии валидации кеша вместо истечения срока действия).

В любом случае, давайте посмотрим, как сконфигурировать Varnish для инвалидации кеша:

```
--- a/.symfony/config.vcl
+++ b/.symfony/config.vcl
@@ -1,6 +1,13 @@
 sub vcl_recv {
     set req.backend_hint = application.backend();
     set req.http.Surrogate-Capability = "abc=ESI/1.0";
+
+    if (req.method == "PURGE") {
+        if (req.http.x-purge-token != "PURGE_NOW") {
+            return(synth(405));
+        }
+        return (purge);
+    }
 }

 sub vcl_backend_response {
```

Скорее всего, в реальности вы разрешите очистку кеша только с

определённых IP-адресов, как об этом описано в *документации Varnish*.

Теперь очистите кеш для некоторых URL-адресов:

```
$ curl -X PURGE -H 'x-purge-token PURGE_NOW' `symfony env:urls --first`
$ curl -X PURGE -H 'x-purge-token PURGE_NOW' `symfony env:urls --
first`conference_header
```

URL-адреса выглядит немного необычно, потому что команда `env:urls` возвращает адреса, которые уже заканчиваются на символ `/`.

Двигаемся дальше

- Глобальная облачная платформа *Cloudflare*;
- *Документация по Varnish*;
- *Спецификация ESI* и *ресурсы по ESI*;
- *Модель валидации HTTP-кеширования*;
- *HTTP-кеширование в SymfonyCloud*.

Шаг 22
Стилизация интерфейса с помощью Webpack

До сих пор мы совсем не занимались дизайном пользовательского интерфейса. Для оформления интерфейса, как подобает настоящему профессионалу, воспользуемся современным стеком технологий при помощи *Webpack*. А чтобы быть ближе к Symfony и как можно проще интегрировать Webpack в наше приложение, установим *Webpack Encore*:

```
$ symfony composer req encore
```

Теперь у нас есть полноценная среда для работы Webpack: сгенерированы файлы `package.json` и `webpack.config.js` с уже рабочими настройкам по умолчанию. Откройте файл `webpack.config.js`, который использует Encore, чтобы настроить Webpack.

В файле `package.json` есть несколько полезных команд, которые мы

будем использовать на протяжении всей книги.

Директория `assets` содержит главные точки входа ресурсов в проекте: `css/app.css` и `js/app.js`.

22.1 Использование Sass

Вместо обычного CSS давайте воспользуемся *Saas*:

```
$ mv assets/css/app.css assets/css/app.scss
```

```
--- a/assets/js/app.js
+++ b/assets/js/app.js
@@ -6,7 +6,7 @@
  */

 // any CSS you require will output into a single css file (app.css in this
case)
-require('../css/app.css');
+import '../css/app.scss';

 // Need jQuery? Install it with "yarn add jquery", then uncomment to require
it.
 // const $ = require('jquery');
```

Установите загрузчик Saas:

```
$ yarn add node-sass "sass-loader@^7.0.1" --dev
```

А затем включите Sass в конфигурации webpack:

```
--- a/webpack.config.js
+++ b/webpack.config.js
@@ -54,7 +54,7 @@ Encore
     })

     // enables Sass/SCSS support
-    //.enableSassLoader()
+    .enableSassLoader()

     // uncomment if you use TypeScript
     //.enableTypeScriptLoader()
```

Как я узнал, какие пакеты нужно установить? При попытке собрать

ресурсы без необходимых пакетов, Encore вернул бы нам сообщение об ошибке и предложил бы выполнить команду `yarn add`, чтобы установить зависимости для обработки файлов с расширением“.scss“.

22.2 Эффективное использование Bootstrap

В создании адаптивного сайта, особенно в самом начале разработки, такой CSS-фреймворк как *Bootstrap,* может значительно помочь. Установите его как пакет:

```
$ yarn add bootstrap jquery popper.js bs-custom-file-input --dev
```

Теперь подключите Bootstrap в следующем CSS-файле (предварительно очистив его):

```
--- a/assets/css/app.scss
+++ b/assets/css/app.scss
@@ -1,3 +1 @@
-body {
-    background-color: lightgray;
-}
+@import '~bootstrap/scss/bootstrap';
```

То же самое нужно сделать в JS-файле:

```
--- a/assets/js/app.js
+++ b/assets/js/app.js
@@ -7,8 +7,7 @@

 // any CSS you require will output into a single css file (app.css in this
case)
 import '../css/app.scss';
+import 'bootstrap';
+import bsCustomFileInput from 'bs-custom-file-input';

-// Need jQuery? Install it with "yarn add jquery", then uncomment to require
it.
-// const $ = require('jquery');
-
-console.log('Hello Webpack Encore! Edit me in assets/js/app.js');
+bsCustomFileInput.init();
```

У форм в Symfony есть встроенная поддержка Bootstrap со специальной темой, включите её:

config/packages/twig.yaml

```
twig:
    form_themes: ['bootstrap_4_layout.html.twig']
```

22.3 Стилизация HTML-шаблона

Теперь всё готово, чтобы непосредственно перейти к оформлению внешнего вида приложения. Скачайте и распакуйте архив в корневой директории проекта:

```
$ php -r "copy('https://symfony.com/uploads/assets/guestbook.zip',
'guestbook.zip');"
$ unzip -o guestbook.zip
$ rm guestbook.zip
```

Изучите код шаблонов, возможно, вы узнаете кое-что новое о Twig.

22.4 Сборка ресурсов

Один важный момент при использовании Webpack — CSS- и JS-файлы не могут использоваться напрямую в приложении. Перед этим их сначала нужно «скомпилировать».

Собрать ресурсы в процессе разработки можно с помощью команды `encore dev`:

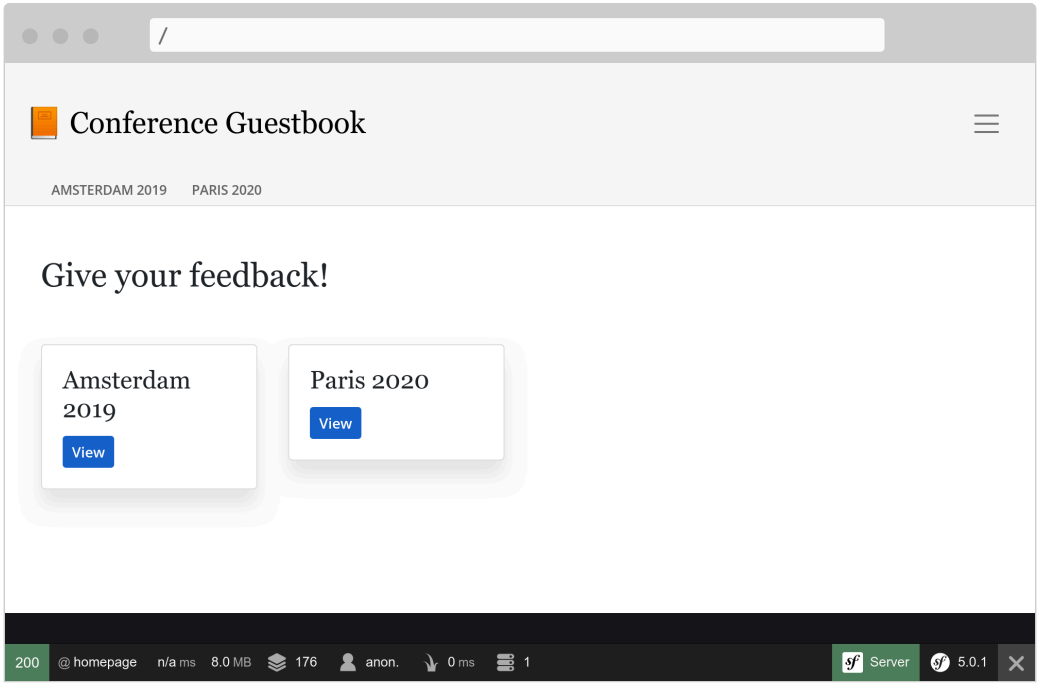
```
$ symfony run yarn encore dev
```

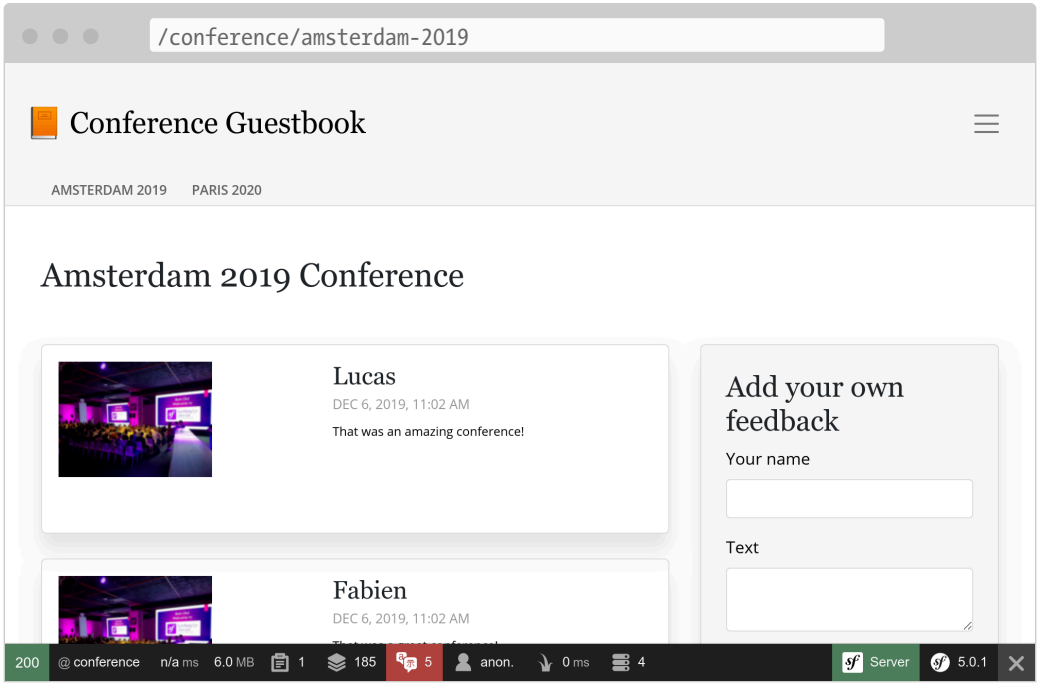
Чтобы не выполнять эту команду каждый раз после внесения изменений, запустите её в фоновом режиме и оставьте наблюдать за изменениями JS и CSS:

```
$ symfony run -d yarn encore dev --watch
```

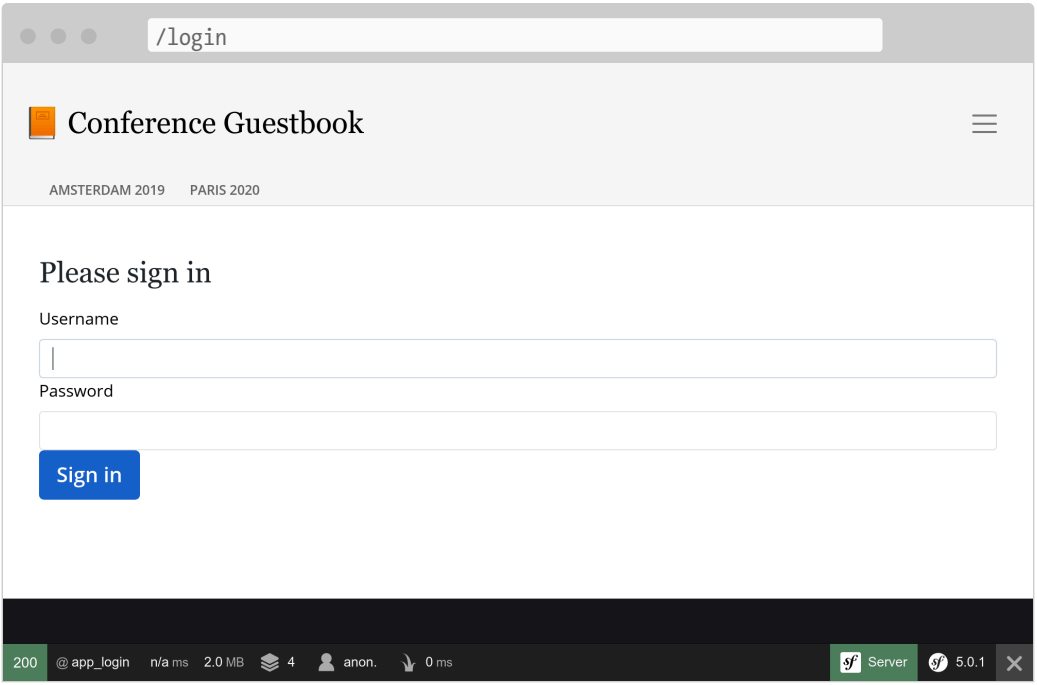
Остановитесь на минутку и изучите изменения внешнего вида.

Посмотрите на новый дизайн в браузере.





Теперь ранее сгенерированная форма входа имеет оформление, потому что бандл Maker по умолчанию использует CSS-классы из Bootstrap:



В продакшен-окружении SymfonyCloud автоматически определит, что в проекте используется Encore и поэтому автоматически во время сборки приложения скомпилирует все его ресурсы.

Двигаемся дальше

- *Документация по Webpack*;
- *Документация по Symfony Webpack Encore*;
- *Учебный видеоролик по Webpack Encore на SymfonyCast*.

Шаг 23

Изменение размера изображений

По дизайну страницы конференции фотографии должны быть размером не более 200x150 пикселей. Может тогда нам стоит оптимизировать и уменьшать изображения, в случае если загруженные фотографии превышают указанный максимальный размер?

Идеальным решением будет включить такую операцию в бизнес-процесс комментария: после проверки и перед публикацией комментария.

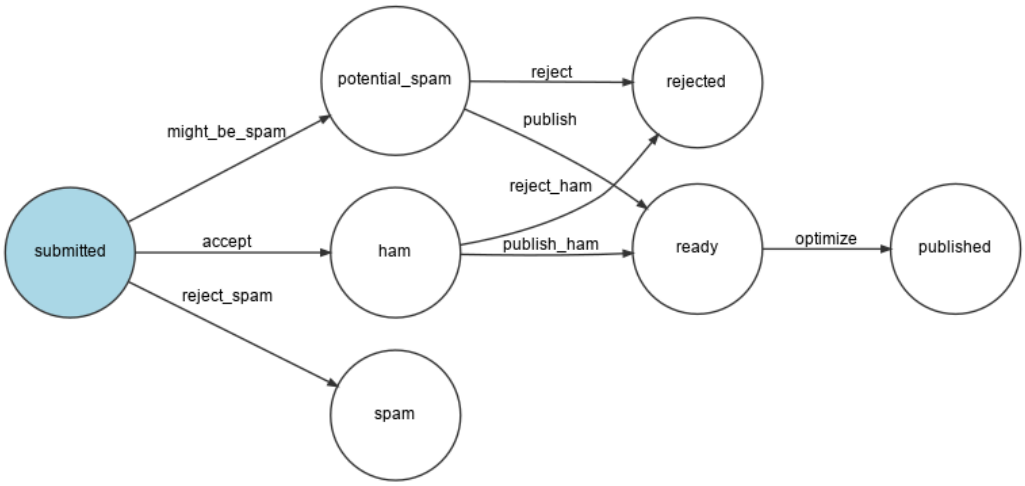
Давайте добавим новое состояние `ready` и переход `optimize`:

```
--- a/config/packages/workflow.yaml
+++ b/config/packages/workflow.yaml
@@ -16,6 +16,7 @@ framework:
                 - potential_spam
                 - spam
                 - rejected
+                - ready
                 - published
             transitions:
```

```
                accept:
@@ -29,13 +30,16 @@ framework:
                    to:   spam
                publish:
                    from: potential_spam
-                   to:   published
+                   to:   ready
                reject:
                    from: potential_spam
                    to:   rejected
                publish_ham:
                    from: ham
-                   to:   published
+                   to:   ready
                reject_ham:
                    from: ham
                    to:   rejected
+               optimize:
+                   from: ready
+                   to:   published
```

Посмотрим, как визуально выглядит измененный бизнес-процесс, чтобы убедиться, что он корректно составлен:

```
$ symfony console workflow:dump comment | dot -Tpng -o workflow.png
```



23.1 Оптимизация изображений с помощью Imagine

Оптимизировать изображения будем через модуль *GD* (должен быть установлен на вашем компьютере) и *Imagine*:

```
$ symfony composer req imagine/imagine
```

Изменение размера изображения будет происходить в отдельном сервисном классе:

src/ImageOptimizer.php

```
namespace App;

use Imagine\Gd\Imagine;
use Imagine\Image\Box;

class ImageOptimizer
{
    private const MAX_WIDTH = 200;
    private const MAX_HEIGHT = 150;

    private $imagine;

    public function __construct()
    {
        $this->imagine = new Imagine();
    }

    public function resize(string $filename): void
    {
        list($iwidth, $iheight) = getimagesize($filename);
        $ratio = $iwidth / $iheight;
        $width = self::MAX_WIDTH;
        $height = self::MAX_HEIGHT;
        if ($width / $height > $ratio) {
            $width = $height * $ratio;
        } else {
            $height = $width / $ratio;
        }

        $photo = $this->imagine->open($filename);
        $photo->resize(new Box($width, $height))->save($filename);
    }
}
```

После оптимизации фотографии заменяем исходный файл на новый. Хотя, возможно, по каким-либо причинам вы решите оставить оригинальное изображение.

23.2 Добавление нового шага в бизнес-процесс

Добавьте в бизнес-процесс обработку нового состояния:

```
--- a/src/MessageHandler/CommentMessageHandler.php
+++ b/src/MessageHandler/CommentMessageHandler.php
@@ -2,6 +2,7 @@

 namespace App\MessageHandler;

+use App\ImageOptimizer;
 use App\Message\CommentMessage;
 use App\Repository\CommentRepository;
 use App\SpamChecker;
@@ -21,10 +22,12 @@ class CommentMessageHandler implements
MessageHandlerInterface
     private $bus;
     private $workflow;
     private $mailer;
+    private $imageOptimizer;
     private $adminEmail;
+    private $photoDir;
     private $logger;

-    public function __construct(EntityManagerInterface $entityManager,
SpamChecker $spamChecker, CommentRepository $commentRepository,
MessageBusInterface $bus, WorkflowInterface $commentStateMachine,
MailerInterface $mailer, string $adminEmail, LoggerInterface $logger = null)
+    public function __construct(EntityManagerInterface $entityManager,
SpamChecker $spamChecker, CommentRepository $commentRepository,
MessageBusInterface $bus, WorkflowInterface $commentStateMachine,
MailerInterface $mailer, ImageOptimizer $imageOptimizer, string $adminEmail,
string $photoDir, LoggerInterface $logger = null)
     {
         $this->entityManager = $entityManager;
         $this->spamChecker = $spamChecker;
@@ -32,7 +35,9 @@ class CommentMessageHandler implements MessageHandlerInterface
         $this->bus = $bus;
         $this->workflow = $commentStateMachine;
         $this->mailer = $mailer;
```

```
+        $this->imageOptimizer = $imageOptimizer;
         $this->adminEmail = $adminEmail;
+        $this->photoDir = $photoDir;
         $this->logger = $logger;
     }

@@ -63,6 +68,12 @@ class CommentMessageHandler implements
MessageHandlerInterface
                 ->to($this->adminEmail)
                 ->context(['comment' => $comment])
             );
+        } elseif ($this->workflow->can($comment, 'optimize')) {
+            if ($comment->getPhotoFilename()) {
+                $this->imageOptimizer->resize($this->photoDir.'/'.$comment-
>getPhotoFilename());
+            }
+            $this->workflow->apply($comment, 'optimize');
+            $this->entityManager->flush();
         } elseif ($this->logger) {
             $this->logger->debug('Dropping comment message', ['comment' =>
$comment->getId(), 'state' => $comment->getState()]);
         }
```

Обратите внимание на автоматически доступную переменную `$photoDir`, так как ранее в одном из предыдущих шагов к этому имени переменной в контейнере мы *привязали* соответствующий параметр:

config/packages/services.yaml

```yaml
services:
    _defaults:
        bind:
            $photoDir: "%kernel.project_dir%/public/uploads/photos"
```

23.3 Хранение загруженных данных в продакшене

В конфигурационном файле `.symfony.cloud.yaml` уже указана специальная директория для чтения и записи загруженных фотографий. Однако она смонтирована только локально. Чтобы к данной директории имели доступ веб-контейнер и воркер для обработки сообщений в очереди, нужно создать *файловый сервис*:

```
--- a/.symfony/services.yaml
+++ b/.symfony/services.yaml
@@ -16,3 +16,7 @@ varnish:
         vcl: !include
             type: string
             path: config.vcl
+
+files:
+    type: network-storage:1.0
+    disk: 256
```

Используйте его для директории загруженных фотографий:

```
--- a/.symfony.cloud.yaml
+++ b/.symfony.cloud.yaml
@@ -29,7 +29,7 @@ disk: 512

 mounts:
     "/var": { source: local, source_path: var }
-    "/public/uploads": { source: local, source_path: uploads }
+    "/public/uploads": { source: service, service: files, source_path: uploads
}

 hooks:
     build: |
```

Этого должно быть достаточно для работы в продакшене.

Шаг 24

Выполнение заданий cron

Задания cron полезны для задач администрирования. В отличие от воркеров, они запускаются на короткое время по расписанию.

24.1 Очистка ненужных комментариев

Комментарии, помеченные как спам или отклонённые администратором, хранятся в базе, чтобы администратор мог просмотреть их позже. Но они, вероятно, в любом случае должны быть удалены через определённое время. Думаю, что хранить такие комментарии в течение недели после создания будет достаточно.

Добавьте несколько вспомогательных методов в репозиторий комментариев для поиска, подсчёта и удаления отклонённых комментариев:

```
--- a/src/Repository/CommentRepository.php
+++ b/src/Repository/CommentRepository.php
@@ -6,6 +6,7 @@ use App\Entity\Comment;
 use App\Entity\Conference;
 use Doctrine\Bundle\DoctrineBundle\Repository\ServiceEntityRepository;
 use Doctrine\Common\Persistence\ManagerRegistry;
+use Doctrine\ORM\QueryBuilder;
 use Doctrine\ORM\Tools\Pagination\Paginator;

 /**
@@ -16,12 +17,37 @@ use Doctrine\ORM\Tools\Pagination\Paginator;
  */
 class CommentRepository extends ServiceEntityRepository
 {
+    private const DAYS_BEFORE_REJECTED_REMOVAL = 7;
+
     public const PAGINATOR_PER_PAGE = 2;

     public function __construct(ManagerRegistry $registry)
     {
         parent::__construct($registry, Comment::class);
     }
+
+    public function countOldRejected(): int
+    {
+        return $this->getOldRejectedQueryBuilder()->select('COUNT(c.id)')-
>getQuery()->getSingleScalarResult();
+    }
+
+    public function deleteOldRejected(): int
+    {
+        return $this->getOldRejectedQueryBuilder()->delete()->getQuery()-
>execute();
+    }
+
+    private function getOldRejectedQueryBuilder(): QueryBuilder
+    {
+        return $this->createQueryBuilder('c')
+            ->andWhere('c.state = :state_rejected or c.state = :state_spam')
+            ->andWhere('c.createdAt < :date')
+            ->setParameters([
+                'state_rejected' => 'rejected',
+                'state_spam' => 'spam',
+                'date' => new \DateTime(-self::DAYS_BEFORE_REJECTED_REMOVAL.'
days'),
+            ])
+        ;
+    }
```

```
    public function getCommentPaginator(Conference $conference, int $offset):
Paginator
    {
```

Для более сложных запросов иногда полезно посмотреть сгенерированные SQL-запросы (которые можно найти в логах и в профилировщике веб-запросов).

24.2 Использование констант класса, параметров контейнера и переменных среды окружения

Почему именно 7 дней? Мы могли бы выбрать другое число, может быть 10 или 20. Это число потом может поменяться. Поэтому лучше всего хранить такие данные в константе класса, что мы и сделали, хотя это значение можно было поместить в параметр контейнера или даже определить соответствующую переменную окружения.

Несколько основных правил, по которым можно определить, какую абстракцию использовать:

- Если значение является конфиденциальной информацией (пароли, токены API и т.д.), используйте *секретное хранилище* Symfony или Vault;
- Если значение динамическое, которое должно изменяться *без* повторного развёртывания, используйте *переменные окружения*;
- Если значение может различаться в разных окружениях, используйте *параметры контейнера*;
- Во всех остальных случаях храните значение в коде, например, в *константах класса*.

24.3 Создание CLI-команды

Удаление старых комментариев — идеальное задание для cron. Оно

должно выполняться регулярно и небольшие задержки в выполнении не играют существенной роли.

Создайте CLI-команду с названием `app:comment:cleanup` в файле `src/Command/CommentCleanupCommand.php`:

src/Command/CommentCleanupCommand.php

```
namespace App\Command;

use App\Repository\CommentRepository;
use Symfony\Component\Console\Command\Command;
use Symfony\Component\Console\Input\InputInterface;
use Symfony\Component\Console\Input\InputOption;
use Symfony\Component\Console\Output\OutputInterface;
use Symfony\Component\Console\Style\SymfonyStyle;

class CommentCleanupCommand extends Command
{
    private $commentRepository;

    protected static $defaultName = 'app:comment:cleanup';

    public function __construct(CommentRepository $commentRepository)
    {
        $this->commentRepository = $commentRepository;

        parent::__construct();
    }

    protected function configure()
    {
        $this
            ->setDescription('Deletes rejected and spam comments from the
database')
            ->addOption('dry-run', null, InputOption::VALUE_NONE, 'Dry run')
        ;
    }

    protected function execute(InputInterface $input, OutputInterface $output):
int
    {
        $io = new SymfonyStyle($input, $output);

        if ($input->getOption('dry-run')) {
            $io->note('Dry mode enabled');

            $count = $this->commentRepository->countOldRejected();
        } else {
            $count = $this->commentRepository->deleteOldRejected();
```

```
        }

        $io->success(sprintf('Deleted "%d" old rejected/spam comments.',
$count));

        return 0;
    }
}
```

Все команды приложения регистрируются вместе со встроенными командами Symfony, и все они доступны через `symfony console`. Поскольку количество доступных команд может быть большим, необходимо группировать их по пространствам имён. По соглашению команды приложения должны храниться в пространстве `app`. Можно добавлять любое количество подпространств, разделяя их двоеточием (`:`).

Команда принимает *input* (аргументы и параметры, переданные команде), а также *output*, который вы можете использовать для вывода данных в консоль.

Очистите базу данных, выполнив команду:

```
$ symfony console app:comment:cleanup
```

24.4 Настройка cron в SymfonyCloud

Одной из удобств SymfonyCloud является то, что большая часть конфигурации хранится в одном файле: `.symfony.cloud.yaml`. Веб-контейнер, воркеры и задания cron описаны в одном месте для простоты администрирования:

```
--- a/.symfony.cloud.yaml
+++ b/.symfony.cloud.yaml
@@ -43,6 +43,15 @@ hooks:

         (>&2 symfony-deploy)

+crons:
```

```
+    comment_cleanup:
+        # Cleanup every night at 11.50 pm (UTC).
+        spec: '50 23 * * *'
+        cmd: |
+            if [ "$SYMFONY_BRANCH" = "master" ]; then
+                croncape symfony console app:comment:cleanup
+            fi
+
 workers:
     messages:
         commands:
```

Раздел `crons` описывает все задания cron. Каждое задание выполняется в соответствии с расписанием, указанным в свойстве `spec`.

Утилита `croncape` следит за выполнением задания и посылает электронное письмо на адреса, указанные в переменной окружения `MAILTO`, если команда возвращает код завершения, отличный от `0`.

Настройте переменную окружения `MAILTO`:

```
$ symfony var:set MAILTO=ops@example.com
```

Вы можете принудительно выполнить задания cron с вашей локальной машины:

```
$ symfony cron comment_cleanup
```

Обратите внимание, что задания cron настроены на все ветки SymfonyCloud. Если вы не хотите запускать какие-то задания вне продакшена, проверьте переменную окружения `$SYMFONY_BRANCH`:

```
if [ "$SYMFONY_BRANCH" = "master" ]; then
    croncape symfony app:invoices:send
fi
```

Двигаемся дальше

- *Синтаксис cron/crontab*;
- *Репозиторий Croncape*;
- *Команды Symfony Console*;
- *Шпаргалка по Symfony Console*.

Шаг 25
Уведомление различными способами

Приложение гостевой книги собирает отзывы о конференциях. Мы хотим улучить обратную связь с нашими пользователями.

Пользователи, вероятно, не понимают, что комментарий находится на проверке, поэтому не публикуется мгновенно. По этой причине они могут повторно его отправить, думая, что произошла техническая ошибка. Было бы здорово уведомить их после отправки комментария.

Кроме того, хорошо бы им сообщить, когда комментарий будет опубликован. Мы просим пользователей указать электронную почту, давайте использовать её.

Существует много способов уведомить пользователей. Электронная почта — это первое, что приходит в голову, хотя мы также можем сделать это и на самом сайте. Отправка SMS-сообщений или уведомлений в Slack или Telegram — вы можете выбрать любой из вариантов.

Компонент Symfony Notifier предлагает множество стратегий уведомления:

```
$ symfony composer req notifier
```

25.1 Отправка уведомлений в браузере

Для начала давайте уведомим пользователей непосредственно в браузере, что их комментарии проверяются после отправки:

```
--- a/src/Controller/ConferenceController.php
+++ b/src/Controller/ConferenceController.php
@@ -14,6 +14,8 @@ use Symfony\Component\HttpFoundation\File\Exception\
FileException;
 use Symfony\Component\HttpFoundation\Request;
 use Symfony\Component\HttpFoundation\Response;
 use Symfony\Component\Messenger\MessageBusInterface;
+use Symfony\Component\Notifier\Notification\Notification;
+use Symfony\Component\Notifier\NotifierInterface;
 use Symfony\Component\Routing\Annotation\Route;
 use Symfony\Component\Workflow\Registry;
 use Twig\Environment;
@@ -60,7 +62,7 @@ class ConferenceController extends AbstractController
     /**
      * @Route("/conference/{slug}", name="conference")
      */
-    public function show(Request $request, Conference $conference,
CommentRepository $commentRepository, string $photoDir)
+    public function show(Request $request, Conference $conference,
CommentRepository $commentRepository, NotifierInterface $notifier, string
$photoDir)
     {
         $comment = new Comment();
         $form = $this->createForm(CommentFormType::class, $comment);
@@ -90,9 +92,15 @@ class ConferenceController extends AbstractController

             $this->bus->dispatch(new CommentMessage($comment->getId(),
$context));

+            $notifier->send(new Notification('Thank you for the feedback; your
comment will be posted after moderation.', ['browser']));
+
             return $this->redirectToRoute('conference', ['slug' => $conference-
```

```
>getSlug()]);
        }

+       if ($form->isSubmitted()) {
+           $notifier->send(new Notification('Can you check your submission?
There are some problems with it.', ['browser']));
+       }
+
        $offset = max(0, $request->query->getInt('offset', 0));
        $paginator = $commentRepository->getCommentPaginator($conference,
$offset);
```

Уведомитель *отправляет уведомление получателям* по *каналу*.

Уведомление состоит из темы, необязательного содержания и важности.

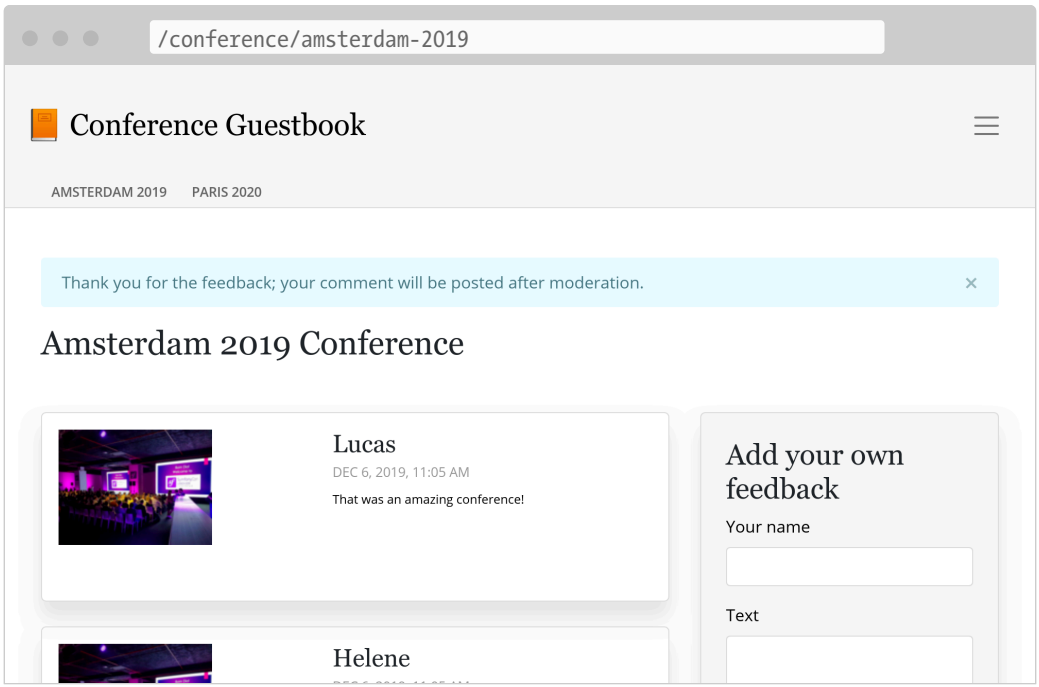
Уведомление отправляется по одному или нескольким каналам в зависимости от важности. Например, вы можете отправлять срочные уведомления по СМС, а обычные — по электронной почте.

У браузерных уведомлений нет получателей.

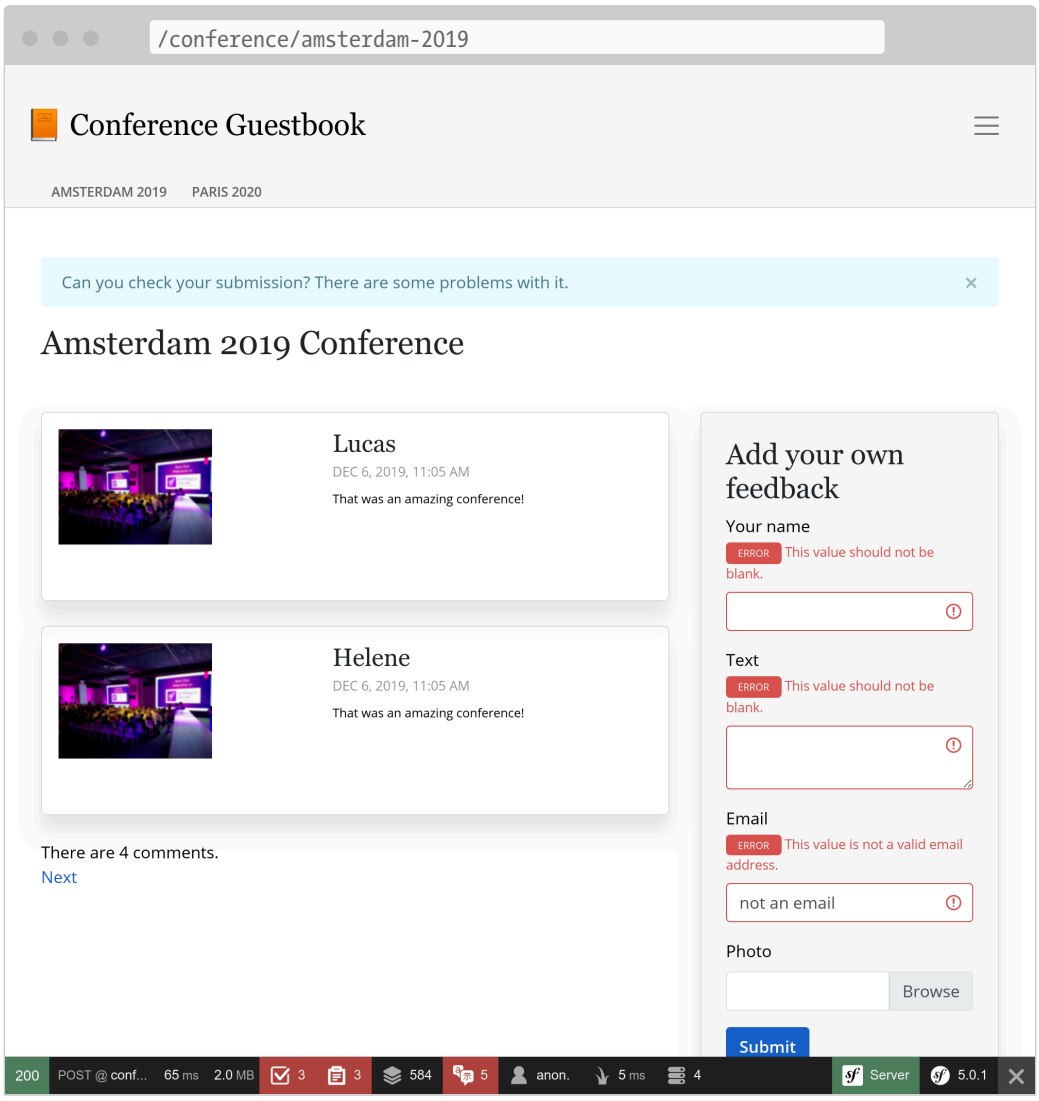
Для уведомлений в браузере используются *мгновенные сообщения* с типом *notification*. Чтобы вывести их, нам нужно обновить шаблон конференции:

```
--- a/templates/conference/show.html.twig
+++ b/templates/conference/show.html.twig
@@ -3,6 +3,13 @@
 {% block title %}Conference Guestbook - {{ conference }}{% endblock %}

 {% block body %}
+    {% for message in app.flashes('notification') %}
+        <div class="alert alert-info alert-dismissible fade show">
+            {{ message }}
+            <button type="button" class="close" data-dismiss="alert" aria-
label="Close"><span aria-hidden="true">&times;</span></button>
+        </div>
+    {% endfor %}
+
     <h2 class="mb-5">
         {{ conference }} Conference
     </h2>
```

Теперь пользователи увидят, что их комментарий проверяется:



А в качестве приятного дополнения в верхней части сайта появится красивое уведомление, если возникнет ошибка при заполнении формы:



Мгновенные сообщения хранятся в *HTTP-сессии*. Именно поэтому HTTP-запросы, использующие сессию, не кешируются по умолчанию, так как сессия должна быть запущена, чтобы проверить новые сообщения.

Вот почему, чтобы кешировать главную страницу, мы добавили вывод мгновенных сообщений не базовом шаблоне, а в `show.html.twig`.

25.2 Уведомление администраторов по электронной почте

Вместо отправки администратору электронного письма о новом комментарии с помощью `MailerInterface`, воспользуемся компонентом Notifier в обработчике сообщений:

```
--- a/src/MessageHandler/CommentMessageHandler.php
+++ b/src/MessageHandler/CommentMessageHandler.php
@@ -4,14 +4,14 @@ namespace App\MessageHandler;

 use App\ImageOptimizer;
 use App\Message\CommentMessage;
+use App\Notification\CommentReviewNotification;
 use App\Repository\CommentRepository;
 use App\SpamChecker;
 use Doctrine\ORM\EntityManagerInterface;
 use Psr\Log\LoggerInterface;
-use Symfony\Bridge\Twig\Mime\NotificationEmail;
-use Symfony\Component\Mailer\MailerInterface;
 use Symfony\Component\Messenger\Handler\MessageHandlerInterface;
 use Symfony\Component\Messenger\MessageBusInterface;
+use Symfony\Component\Notifier\NotifierInterface;
 use Symfony\Component\Workflow\WorkflowInterface;

 class CommentMessageHandler implements MessageHandlerInterface
@@ -21,22 +21,20 @@ class CommentMessageHandler implements
MessageHandlerInterface
     private $commentRepository;
     private $bus;
     private $workflow;
-    private $mailer;
+    private $notifier;
     private $imageOptimizer;
-    private $adminEmail;
     private $photoDir;
     private $logger;

-    public function __construct(EntityManagerInterface $entityManager,
SpamChecker $spamChecker, CommentRepository $commentRepository,
MessageBusInterface $bus, WorkflowInterface $commentStateMachine,
MailerInterface $mailer, ImageOptimizer $imageOptimizer, string $adminEmail,
string $photoDir, LoggerInterface $logger = null)
+    public function __construct(EntityManagerInterface $entityManager,
```

```
SpamChecker $spamChecker, CommentRepository $commentRepository,
MessageBusInterface $bus, WorkflowInterface $commentStateMachine,
NotifierInterface $notifier, ImageOptimizer $imageOptimizer, string $photoDir,
LoggerInterface $logger = null)
     {
         $this->entityManager = $entityManager;
         $this->spamChecker = $spamChecker;
         $this->commentRepository = $commentRepository;
         $this->bus = $bus;
         $this->workflow = $commentStateMachine;
-        $this->mailer = $mailer;
+        $this->notifier = $notifier;
         $this->imageOptimizer = $imageOptimizer;
-        $this->adminEmail = $adminEmail;
         $this->photoDir = $photoDir;
         $this->logger = $logger;
     }
@@ -62,13 +60,7 @@ class CommentMessageHandler implements
MessageHandlerInterface

             $this->bus->dispatch($message);
         } elseif ($this->workflow->can($comment, 'publish') || $this->workflow-
>can($comment, 'publish_ham')) {
-            $this->mailer->send((new NotificationEmail())
-                ->subject('New comment posted')
-                ->htmlTemplate('emails/comment_notification.html.twig')
-                ->from($this->adminEmail)
-                ->to($this->adminEmail)
-                ->context(['comment' => $comment])
-            );
+            $this->notifier->send(new CommentReviewNotification($comment),
...$this->notifier->getAdminRecipients());
         } elseif ($this->workflow->can($comment, 'optimize')) {
             if ($comment->getPhotoFilename()) {
                 $this->imageOptimizer->resize($this->photoDir.'/'.$comment-
>getPhotoFilename());
```

Метод `getAdminRecipients()` возвращает список администраторов, которых необходимо уведомить; добавьте в него свою электронную почту:

```
--- a/config/packages/notifier.yaml
+++ b/config/packages/notifier.yaml
@@ -13,4 +13,4 @@ framework:
             medium: ['email']
             low: ['email']
         admin_recipients:
```

```
-            - { email: admin@example.com }
+            - { email: "%env(string:default:default_admin_email:ADMIN_EMAIL)%"
}
```

Теперь создайте класс `CommentReviewNotification`:

src/Notification/CommentReviewNotification.php

```
namespace App\Notification;

use App\Entity\Comment;
use Symfony\Component\Notifier\Message\EmailMessage;
use Symfony\Component\Notifier\Notification\EmailNotificationInterface;
use Symfony\Component\Notifier\Notification\Notification;
use Symfony\Component\Notifier\Recipient\Recipient;

class CommentReviewNotification extends Notification implements
EmailNotificationInterface
{
    private $comment;

    public function __construct(Comment $comment)
    {
        $this->comment = $comment;

        parent::__construct('New comment posted');
    }

    public function asEmailMessage(Recipient $recipient, string $transport =
null): ?EmailMessage
    {
        $message = EmailMessage::fromNotification($this, $recipient,
$transport);
        $message->getMessage()
            ->htmlTemplate('emails/comment_notification.html.twig')
            ->context(['comment' => $this->comment])
        ;

        return $message;
    }
}
```

Необязательный метод `asEmailMessage()` интерфейса `EmailNotificationInterface` позволяет изменить сообщение электронной почты.

Одним из преимуществ использования Notifier вместо соответствующего компонента для отправки почты напрямую состоит

в том, что он отделяет уведомление от выбранного «канала». То есть, как вы видите, нет явного указания, что уведомление должно быть отправлено по электронной почте.

Вместо этого канал настраивается в файле `config/packages/notifier.yaml` и выбирается в зависимости от *важности* уведомления (`low` по умолчанию):

config/packages/notifier.yaml

```
framework:
notifier:
    channel_policy:
        # use chat/slack, chat/telegram, sms/twilio or sms/nexmo
        urgent: ['email']
        high: ['email']
        medium: ['email']
        low: ['email']
```

Мы рассмотрели каналы `browser` и `email`. Давайте посмотрим на другие, более интересные и сложные.

25.3 Отправка уведомлений в чаты для администраторов

Давайте будем честны: мы все ожидаем положительных или, по крайней мере, конструктивных отзывов. Если кто-то напишет комментарий со словами «great» или «awesome», мы, скорее всего, одобрим его быстрее.

Такие сообщения хочется получать не только по электронной почте, но и в чатах, например, в Slack или Telegram.

Добавьте поддержку Slack для Symfony Notifier:

```
$ symfony composer req slack-notifier
```

Для начала сформируйте строку подключения DSN для Slack с токеном доступа и идентификатором канала Slack, куда вы хотите отправлять сообщения: `slack://ACCESS_TOKEN@default?channel=CHANNEL`.

Поскольку токен доступа относится к конфиденциальной информации, то сохраните DSN-строку Slack в соответствующем

хранилище:

```
$ symfony console secrets:set SLACK_DSN
```

Проделайте то же самое для продакшена:

```
$ APP_ENV=prod symfony console secrets:set SLACK_DSN
```

Включите поддержку Chatter в Slack:

```
--- a/config/packages/notifier.yaml
+++ b/config/packages/notifier.yaml
@@ -1,7 +1,7 @@
 framework:
     notifier:
-        #chatter_transports:
-        #    slack: '%env(SLACK_DSN)%'
+        chatter_transports:
+            slack: '%env(SLACK_DSN)%'
         #    telegram: '%env(TELEGRAM_DSN)%'
         #texter_transports:
         #    twilio: '%env(TWILIO_DSN)%'
```

Обновите класс уведомления, чтобы отправлять сообщения в нужные каналы, в зависимости от содержания комментария (с этим справится простое регулярное выражение):

```
--- a/src/Notification/CommentReviewNotification.php
+++ b/src/Notification/CommentReviewNotification.php
@@ -27,4 +27,15 @@ class CommentReviewNotification extends Notification
implements EmailNotificationInterface
             ->context(['comment' => $this->comment])
         );
     }
+
+    public function getChannels(Recipient $recipient): array
+    {
+        if (preg_match('{\b(great|awesome)\b}i', $this->comment->getText())) {
+            return ['email', 'chat/slack'];
+        }
+
+        $this->importance(Notification::IMPORTANCE_LOW);
+
+        return ['email'];
+    }
 }
```

Мы также изменили важность «обычных» комментариев, так как они слегка изменяют дизайн письма.

Готово! Отправьте комментарий, содержащий слово «awesome» и в чате Slack вы увидите этот комментарий.

По аналогии с электронным письмом, вы также можете изменить стандартное оформление сообщения в Slack, если реализуете интерфейс `ChatNotificationInterface`:

```
--- a/src/Notification/CommentReviewNotification.php
+++ b/src/Notification/CommentReviewNotification.php
@@ -3,12 +3,17 @@
 namespace App\Notification;

 use App\Entity\Comment;
+use Symfony\Component\Notifier\Bridge\Slack\Block\SlackDividerBlock;
+use Symfony\Component\Notifier\Bridge\Slack\Block\SlackSectionBlock;
+use Symfony\Component\Notifier\Bridge\Slack\SlackOptions;
+use Symfony\Component\Notifier\Message\ChatMessage;
 use Symfony\Component\Notifier\Message\EmailMessage;
+use Symfony\Component\Notifier\Notification\ChatNotificationInterface;
 use Symfony\Component\Notifier\Notification\EmailNotificationInterface;
 use Symfony\Component\Notifier\Notification\Notification;
 use Symfony\Component\Notifier\Recipient\Recipient;

-class CommentReviewNotification extends Notification implements
EmailNotificationInterface
+class CommentReviewNotification extends Notification implements
EmailNotificationInterface, ChatNotificationInterface
 {
     private $comment;

@@ -30,6 +35,28 @@ class CommentReviewNotification extends Notification
implements EmailNotificatio
         return $message;
     }

+    public function asChatMessage(Recipient $recipient, string $transport =
null): ?ChatMessage
+    {
+        if ('slack' !== $transport) {
+            return null;
+        }
+
+        $message = ChatMessage::fromNotification($this, $recipient,
$transport);
+        $message->subject($this->getSubject());
+        $message->options((new SlackOptions())
```

```
+            ->iconEmoji('tada')
+            ->iconUrl('https://guestbook.example.com')
+            ->username('Guestbook')
+            ->block((new SlackSectionBlock())->text($this->getSubject()))
+            ->block(new SlackDividerBlock())
+            ->block((new SlackSectionBlock())
+                ->text(sprintf('%s (%s) says: %s', $this->comment-
>getAuthor(), $this->comment->getEmail(), $this->comment->getText()))
+            )
+        );
+
+        return $message;
+    }
+
     public function getChannels(Recipient $recipient): array
     {
         if (preg_match('{\b(great|awesome)\b}i', $this->comment->getText())) {
```

Так-то лучше, но давайте пойдём ещё дальше. Разве не было бы здорово одобрить или отклонить комментарий непосредственно в Slack?

Измените уведомление, чтобы оно принимало URL-адрес проверки комментария и добавьте две кнопки в сообщение Slack:

```
--- a/src/Notification/CommentReviewNotification.php
+++ b/src/Notification/CommentReviewNotification.php
@@ -3,6 +3,7 @@
 namespace App\Notification;

 use App\Entity\Comment;
+use Symfony\Component\Notifier\Bridge\Slack\Block\SlackActionsBlock;
 use Symfony\Component\Notifier\Bridge\Slack\Block\SlackDividerBlock;
 use Symfony\Component\Notifier\Bridge\Slack\Block\SlackSectionBlock;
 use Symfony\Component\Notifier\Bridge\Slack\SlackOptions;
@@ -16,10 +17,12 @@ use Symfony\Component\Notifier\Recipient\Recipient;
 class CommentReviewNotification extends Notification implements
EmailNotificationInterface, ChatNotificationInterface
 {
     private $comment;
+    private $reviewUrl;

-    public function __construct(Comment $comment)
+    public function __construct(Comment $comment, string $reviewUrl)
     {
         $this->comment = $comment;
+        $this->reviewUrl = $reviewUrl;
```

```
        parent::__construct('New comment posted');
    }
@@ -52,6 +55,10 @@ class CommentReviewNotification extends Notification
implements EmailNotificatio
            ->block((new SlackSectionBlock())
                ->text(sprintf('%s (%s) says: %s', $this->comment-
>getAuthor(), $this->comment->getEmail(), $this->comment->getText()))
            )
+           ->block((new SlackActionsBlock())
+               ->button('Accept', $this->reviewUrl, 'primary')
+               ->button('Reject', $this->reviewUrl.'?reject=1', 'danger')
+           )
        );

    return $message;
```

Теперь мы будем вносить изменения в обратном порядке. Для начала обновите обработчик сообщения и передайте в него URL-адрес проверки комментария:

```
--- a/src/MessageHandler/CommentMessageHandler.php
+++ b/src/MessageHandler/CommentMessageHandler.php
@@ -60,7 +60,8 @@ class CommentMessageHandler implements MessageHandlerInterface

            $this->bus->dispatch($message);
        } elseif ($this->workflow->can($comment, 'publish') || $this->workflow-
>can($comment, 'publish_ham')) {
-           $this->notifier->send(new CommentReviewNotification($comment),
...$this->notifier->getAdminRecipients());
+           $notification = new CommentReviewNotification($comment, $message-
>getReviewUrl());
+           $this->notifier->send($notification, ...$this->notifier-
>getAdminRecipients());
        } elseif ($this->workflow->can($comment, 'optimize')) {
            if ($comment->getPhotoFilename()) {
                $this->imageOptimizer->resize($this->photoDir.'/'.$comment-
>getPhotoFilename());
```

Как видите, проверочный адрес должен быть в самом сообщении, поэтому добавляем его:

```
--- a/src/Message/CommentMessage.php
+++ b/src/Message/CommentMessage.php
@@ -5,14 +5,21 @@ namespace App\Message;
 class CommentMessage
```

```
 {
     private $id;
+    private $reviewUrl;
     private $context;

-    public function __construct(int $id, array $context = [])
+    public function __construct(int $id, string $reviewUrl, array $context =
[])
     {
         $this->id = $id;
+        $this->reviewUrl = $reviewUrl;
         $this->context = $context;
     }

+    public function getReviewUrl(): string
+    {
+        return $this->reviewUrl;
+    }
+
     public function getId(): int
     {
         return $this->id;
```

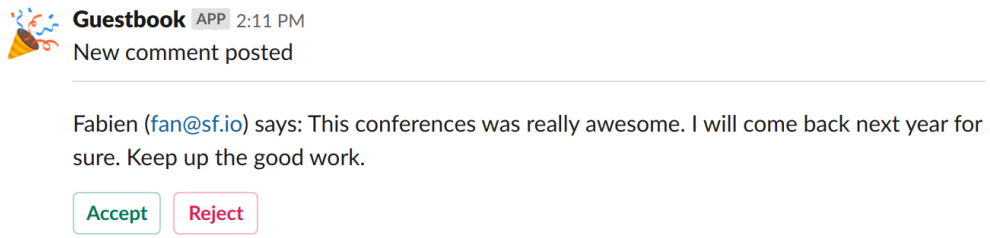
И наконец, сгенерируйте адрес проверки комментария в контроллере, а затем передайте его в конструктор сообщения:

```
--- a/src/Controller/AdminController.php
+++ b/src/Controller/AdminController.php
@@ -12,6 +12,7 @@ use Symfony\Component\HttpFoundation\Response;
 use Symfony\Component\HttpKernel\KernelInterface;
 use Symfony\Component\Messenger\MessageBusInterface;
 use Symfony\Component\Routing\Annotation\Route;
+use Symfony\Component\Routing\Generator\UrlGeneratorInterface;
 use Symfony\Component\Workflow\Registry;
 use Twig\Environment;

@@ -51,7 +52,8 @@ class AdminController extends AbstractController
         $this->entityManager->flush();

         if ($accepted) {
-            $this->bus->dispatch(new CommentMessage($comment->getId()));
+            $reviewUrl = $this->generateUrl('review_comment', ['id' =>
$comment->getId()], UrlGeneratorInterface::ABSOLUTE_URL);
+            $this->bus->dispatch(new CommentMessage($comment->getId(),
$reviewUrl));
         }

         return $this->render('admin/review.html.twig', [
```

```
--- a/src/Controller/ConferenceController.php
+++ b/src/Controller/ConferenceController.php
@@ -17,6 +17,7 @@ use Symfony\Component\Messenger\MessageBusInterface;
 use Symfony\Component\Notifier\Notification\Notification;
 use Symfony\Component\Notifier\NotifierInterface;
 use Symfony\Component\Routing\Annotation\Route;
+use Symfony\Component\Routing\Generator\UrlGeneratorInterface;
 use Twig\Environment;

 class ConferenceController extends AbstractController
@@ -89,7 +90,8 @@ class ConferenceController extends AbstractController
                 'permalink' => $request->getUri(),
             ];

-            $this->bus->dispatch(new CommentMessage($comment->getId(),
$context));
+            $reviewUrl = $this->generateUrl('review_comment', ['id' =>
$comment->getId()], UrlGeneratorInterface::ABSOLUTE_URL);
+            $this->bus->dispatch(new CommentMessage($comment->getId(),
$reviewUrl, $context));

             $notifier->send(new Notification('Thank you for the feedback; your
comment will be posted after moderation.', ['browser']));
```

Декомпозиция кода предполагает изменения в большем количестве мест, но зато она облегчает тестирование, анализ и повторное использование.

Попробуйте ещё раз: сообщение должно быть правильным:



25.4 Включение асинхронного режима для всех каналов

Давайте я объясняю небольшую проблему, которую нам нужно исправить. При каждом добавленном комментарии мы получаем

электронное письмо и сообщение в Slack. Если при отправке Slack-сообщения возникает ошибка (неправильный идентификатор канала, неверный токен и т.п.), то произойдёт повторная отправка сообщения три раза, прежде чем оно будет отклонено. Но поскольку сначала отправляется уведомление по почте, то в итоге у нас будут 3 электронных письма и ни одного сообщения в Slack. Одним из способов решения этой проблемы — это отправка Slack-сообщений асинхронно, как это уже сделано для электронной почтой:

```
--- a/config/packages/messenger.yaml
+++ b/config/packages/messenger.yaml
@@ -20,3 +20,5 @@ framework:
             # Route your messages to the transports
             App\Message\CommentMessage: async
             Symfony\Component\Mailer\Messenger\SendEmailMessage: async
+            Symfony\Component\Notifier\Message\ChatMessage: async
+            Symfony\Component\Notifier\Message\SmsMessage: async
```

После того, как во все каналы сообщения отправляются асинхронно, то сами сообщения перестают зависеть друг от друга. Вдобавок мы также включили асинхронную пересылку SMS-сообщений на случай, если вам понадобится получать уведомления на свой телефон.

25.5 Уведомление пользователей по электронной почте

Последняя задача — уведомить пользователей, когда их комментарий будет одобрен. Как насчёт того, чтобы реализовать это самостоятельно?

Двигаемся дальше

- *Мгновенные сообщения в Symfony*.

Шаг 26

Создание API с помощью API Platform

Мы завершили разработку гостевой книги. Теперь, чтобы использовать данные в полной мере, может быть создадим API? В дальнейшем этот API может использоваться мобильным приложением, в котором будут показываться все конференции и комментарии к ним с возможностью для участников оставить свой отзыв к одной из них.

Сейчас мы разработаем API только для чтения данных.

26.1 Установка API Platform

Конечно, вы можете создать API самостоятельно. Но если вы хотите следовать стандартам, которые применяются при разработке API, лучше всего воспользоваться готовым решением, которое сделает за

вас всю грязную работу. API Platform — как раз одно из таких решений:

```
$ symfony composer req api
```

26.2 Создание API для работы с конференциями

Несколько аннотаций в классе Conference — это всё, что нужно для настройки API:

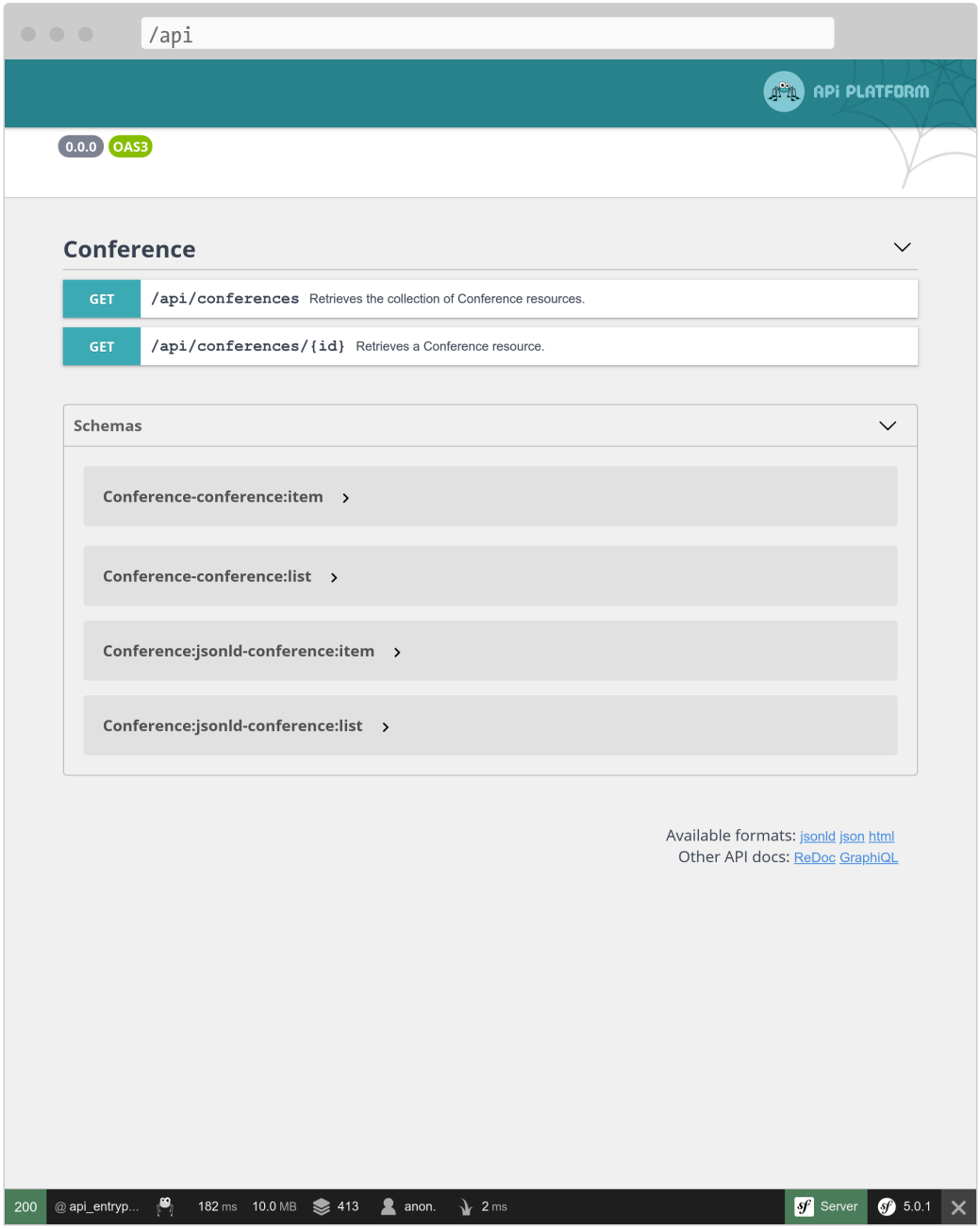
```
--- a/src/Entity/Conference.php
+++ b/src/Entity/Conference.php
@@ -2,15 +2,24 @@

 namespace App\Entity;

+use ApiPlatform\Core\Annotation\ApiResource;
 use Doctrine\Common\Collections\ArrayCollection;
 use Doctrine\Common\Collections\Collection;
 use Doctrine\ORM\Mapping as ORM;
 use Symfony\Bridge\Doctrine\Validator\Constraints\UniqueEntity;
+use Symfony\Component\Serializer\Annotation\Groups;
 use Symfony\Component\String\Slugger\SluggerInterface;

 /**
  * @ORM\Entity(repositoryClass="App\Repository\ConferenceRepository")
  * @UniqueEntity("slug")
+ *
+ * @ApiResource(
+ *
collectionOperations={"get"={"normalization_context"={"groups"="conference:list"}}},
+ *
itemOperations={"get"={"normalization_context"={"groups"="conference:item"}}},
+ *     order={"year"="DESC", "city"="ASC"},
+ *     paginationEnabled=false
+ * )
  */
 class Conference
 {
@@ -18,21 +26,29 @@ class Conference
      * @ORM\Id()
      * @ORM\GeneratedValue()
      * @ORM\Column(type="integer")
```

```
+     *
+     * @Groups({"conference:list", "conference:item"})
      */
     private $id;

     /**
      * @ORM\Column(type="string", length=255)
+     *
+     * @Groups({"conference:list", "conference:item"})
      */
     private $city;

     /**
      * @ORM\Column(type="string", length=4)
+     *
+     * @Groups({"conference:list", "conference:item"})
      */
     private $year;

     /**
      * @ORM\Column(type="boolean")
+     *
+     * @Groups({"conference:list", "conference:item"})
      */
     private $isInternational;

@@ -43,6 +59,8 @@ class Conference

     /**
      * @ORM\Column(type="string", length=255, unique=true)
+     *
+     * @Groups({"conference:list", "conference:item"})
      */
     private $slug;
```

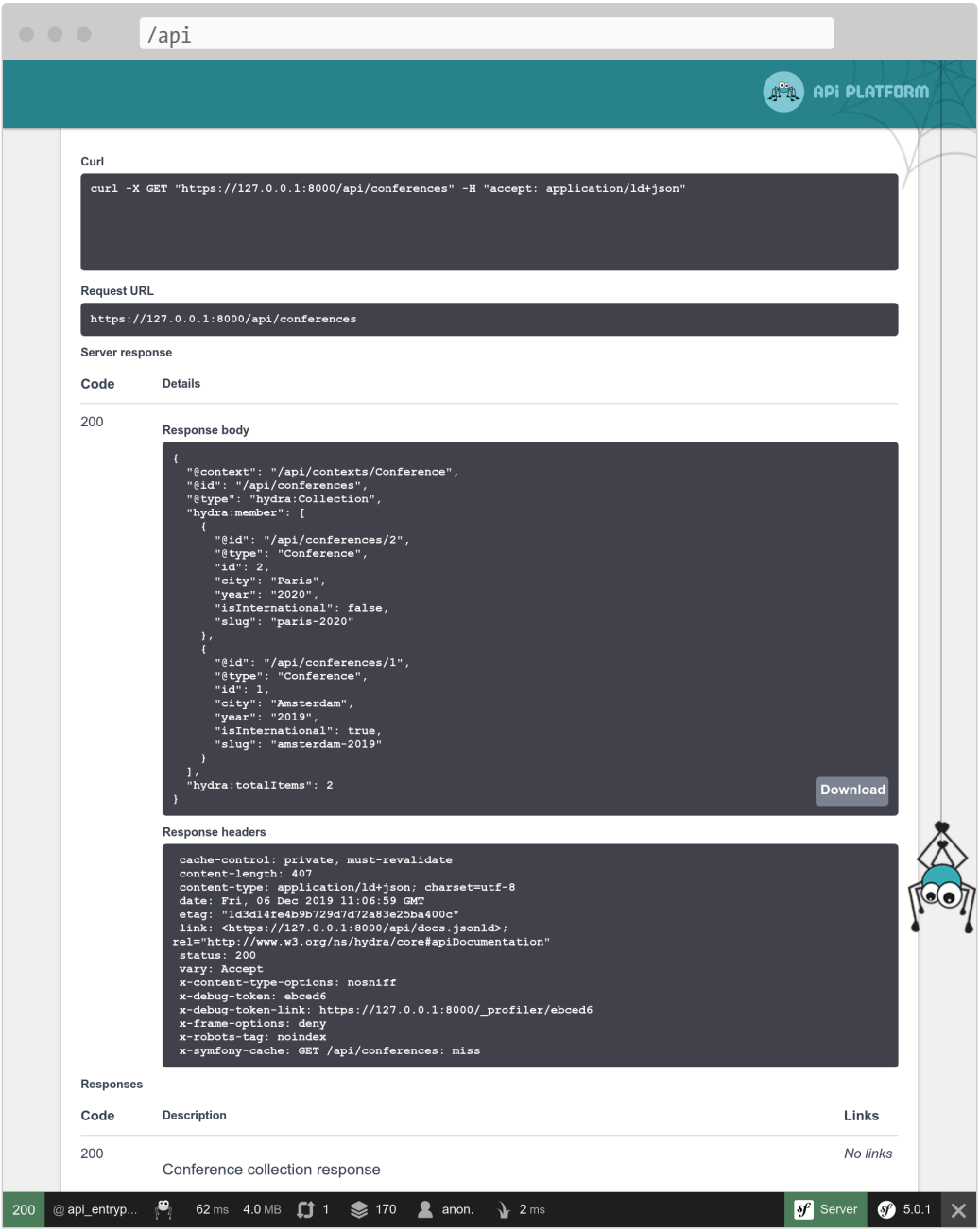
API для конференций настраиваем через основную аннотацию `@ApiResource`. С помощью неё можно ограничить допустимые CRUD-операции только до получения (`get`) и определить другие конфигурационные параметры для конференций: отображаемые поля и их порядок.

По умолчанию API доступен по пути `/api`, который задан в файле `config/routes/api_platform.yaml`. Данный файл с конфигурацией был добавлен рецептом пакета.

Для взаимодействия с API вы можете использовать следующий веб-интерфейс:



Используйте его, чтобы проверить различные возможности API:



Только представьте, сколько времени понадобится для реализации всего этого с нуля!

26.3 Создание API для комментариев

API для получения комментариев сделаем по аналогии с предыдущим:

```
--- a/src/Entity/Comment.php
+++ b/src/Entity/Comment.php
@@ -2,12 +2,25 @@

 namespace App\Entity;

+use ApiPlatform\Core\Annotation\ApiFilter;
+use ApiPlatform\Core\Annotation\ApiResource;
+use ApiPlatform\Core\Bridge\Doctrine\Orm\Filter\SearchFilter;
 use Doctrine\ORM\Mapping as ORM;
+use Symfony\Component\Serializer\Annotation\Groups;
 use Symfony\Component\Validator\Constraints as Assert;

 /**
  * @ORM\Entity(repositoryClass="App\Repository\CommentRepository")
  * @ORM\HasLifecycleCallbacks()
+ *
+ * @ApiResource(
+ *
collectionOperations={"get"={"normalization_context"={"groups"="comment:list"}}},
+ *
itemOperations={"get"={"normalization_context"={"groups"="comment:item"}}},
+ *     order={"createdAt"="DESC"},
+ *     paginationEnabled=false
+ * )
+ *
+ * @ApiFilter(SearchFilter::class, properties={"conference": "exact"})
  */
 class Comment
 {
@@ -15,18 +27,24 @@ class Comment
      * @ORM\Id()
      * @ORM\GeneratedValue()
      * @ORM\Column(type="integer")
+     *
+     * @Groups({"comment:list", "comment:item"})
      */
     private $id;

     /**
      * @ORM\Column(type="string", length=255)
      * @Assert\NotBlank
+     *
+     * @Groups({"comment:list", "comment:item"})
      */
     private $author;

     /**
      * @ORM\Column(type="text")
```

```
      * @Assert\NotBlank
+     *
+     * @Groups({"comment:list", "comment:item"})
      */
     private $text;

@@ -34,22 +52,30 @@ class Comment
      * @ORM\Column(type="string", length=255)
      * @Assert\NotBlank
      * @Assert\Email
+     *
+     * @Groups({"comment:list", "comment:item"})
      */
     private $email;

     /**
      * @ORM\Column(type="datetime")
+     *
+     * @Groups({"comment:list", "comment:item"})
      */
     private $createdAt;

     /**
      * @ORM\ManyToOne(targetEntity="App\Entity\Conference",
inversedBy="comments")
      * @ORM\JoinColumn(nullable=false)
+     *
+     * @Groups({"comment:list", "comment:item"})
      */
     private $conference;

     /**
      * @ORM\Column(type="string", length=255, nullable=true)
+     *
+     * @Groups({"comment:list", "comment:item"})
      */
     private $photoFilename;
```

Такие же аннотации вы можете использовать для настройки класса.

26.4 Ограничение отображения комментариев из API

По умолчанию API Platform возвращает все записи из базы данных. Однако в случае с API для комментариев, нам нужно показывать

только опубликованные среди них.

Для получения через API только определённых элементов, создайте сервис, реализующий интерфейс `QueryCollectionExtensionInterface` для изменения Doctrine-запроса коллекций, и/или интерфейс `QueryItemExtensionInterface` для фильтрации элементов:

src/Api/FilterPublishedCommentQueryExtension.php

```
namespace App\Api;

use ApiPlatform\Core\Bridge\Doctrine\Orm\Extension\
QueryCollectionExtensionInterface;
use ApiPlatform\Core\Bridge\Doctrine\Orm\Extension\QueryItemExtensionInterface;
use ApiPlatform\Core\Bridge\Doctrine\Orm\Util\QueryNameGeneratorInterface;
use App\Entity\Comment;
use Doctrine\ORM\QueryBuilder;

class FilterPublishedCommentQueryExtension implements
QueryCollectionExtensionInterface, QueryItemExtensionInterface
{
    public function applyToCollection(QueryBuilder $qb,
QueryNameGeneratorInterface $queryNameGenerator, string $resourceClass, string
$operationName = null)
    {
        if (Comment::class === $resourceClass) {
            $qb->andWhere(sprintf("%s.state = 'published'",
$qb->getRootAliases()[0]));
        }
    }

    public function applyToItem(QueryBuilder $qb, QueryNameGeneratorInterface
$queryNameGenerator, string $resourceClass, array $identifiers, string
$operationName = null, array $context = [])
    {
        if (Comment::class === $resourceClass) {
            $qb->andWhere(sprintf("%s.state = 'published'",
$qb->getRootAliases()[0]));
        }
    }
}
```

Класс расширения запроса применяется исключительно к ресурсу `Comment`, изменяя построитель запросов Doctrine, чтобы тот вернул только комментарии с состоянием `published`.

26.5 Настройка CORS

Сейчас все современные HTTP-клиенты следуют правилам ограничения домена (same-origin policy), которые запрещают обращаться к API из других доменов. Бандл CORS, устанавливаемый в качестве одной из зависимостей при выполнении команды `composer req api`, отправляет HTTP-заголовки механизма совместного использования ресурсов между разными источниками (Cross-Origin Resource Sharing), в соответствии со значением в переменной окружения `CORS_ALLOW_ORIGIN`.

По умолчанию это значение определено в файле `.env` и разрешает выполнять HTTP-запросы с `localhost` и `127.0.0.1` через любой порт. Этой настройки как раз достаточно для следующего шага, где мы создадим SPA с собственным веб-сервером, который будет взаимодействовать с нашим API.

Двигаемся дальше

- *Обучающий видеокурс по API Platform на SymfonyCasts*;
- Чтобы включить поддержку GraphQL, выполните команду `composer require webonyx/graphql-php`. Затем перейдите по пути `/api/graphql` в браузере.

Шаг 27
Разработка SPA

Большинство комментариев будет отправлено во время конференции, на которую не все принесут с собой ноутбуки. Зато, скорее всего, у них будут смартфоны. Так почему бы не создать мобильное приложение, в котором можно быстро посмотреть комментарии с конференции?

Собрать одностраничное приложение (JavaScript Single Page Application, SPA) — один из способов создать такое мобильное приложение. SPA запускается локально, может использовать локальное хранилище, выполнять HTTP-запросы к сторонним API, а ещё поддерживает сервис-воркеры, которые дают преимущества почти настоящего (нативного) приложения.

27.1 Создание приложения

Для создания мобильного приложения будем использовать *Preact* и **Symfony Encore**. **Preact** — это небольшая и эффективная библиотека, которая хорошо подходит для нашего SPA-приложения гостевой книги.

Чтобы сделать сайт и SPA понятным и предсказуемым, для мобильного приложения мы будем использовать те же таблицы стилей

Sass, что и для сайта.

Создайте SPA-приложение в директории `spa` и скопируйте туда таблицы стилей:

```
$ mkdir -p spa/src spa/public spa/assets/css
$ cp assets/css/*.scss spa/assets/css/
$ cd spa
```

Мы создали директорию `public`, поскольку, как правило, посещать SPA-приложение будем через браузер. Мы бы назвали эту директорию `build` в случае, если нам нужно было только мобильное приложение.

Сгенерируйте файл `package.json` (аналог файла `composer.json` для JavaScript):

```
$ yarn init -y
```

А теперь добавим несколько необходимых зависимостей:

```
$ yarn add @symfony/webpack-encore @babel/core @babel/preset-env babel-preset-
preact preact html-webpack-plugin bootstrap
```

Также не забудем про файл `.gitignore`:

.gitignore

```
/node_modules
/public
/yarn-error.log
# used later by Cordova
/app
```

И последнее — сконфигурируем Webpack Encore:

webpack.config.js

```
const Encore = require('@symfony/webpack-encore');
const HtmlWebpackPlugin = require('html-webpack-plugin');

Encore
    .setOutputPath('public/')
    .setPublicPath('/')
    .cleanupOutputBeforeBuild()
```

```
    .addEntry('app', './src/app.js')
    .enablePreactPreset()
    .enableSingleRuntimeChunk()
    .addPlugin(new HtmlWebpackPlugin({ template: 'src/index.ejs',
alwaysWriteToDisk: true }))
;

module.exports = Encore.getWebpackConfig();
```

27.2 Создание основного шаблона для SPA

Пришло время создать главный шаблон, в котором Preact будет рендерить приложение:

src/index.ejs

```
<!DOCTYPE html>
<html>
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="msapplication-tap-highlight" content="no" />
    <meta name="viewport" content="user-scalable=no, initial-scale=1, maximum-
scale=1, minimum-scale=1, width=device-width" />

    <title>Conference Guestbook application</title>
</head>
<body>
    <div id="app"></div>
</body>
</html>
```

В теге `<div>` с помощью JavaScript будет отрендерено приложение. Первоначальная версия только отобразит на экране надпись «Hello World»:

src/app.js

```
import {h, render} from 'preact';

function App() {
    return (
```

```
        <div>
            Hello world!
        </div>
    )
}

render(<App />, document.getElementById('app'));
```

В последней строке мы указываем функцию `App()` для рендера в элементе `#app` на HTML-странице.

Все готово!

27.3 Запуск SPA в браузере

Поскольку данное приложение работает независимо от основного сайта, нам нужно запустить ещё один веб-сервер:

```
$ symfony server:start -d --passthru=index.html
```

Флаг `--passthru` указывает веб-серверу, что необходимо перенаправлять все HTTP-запросы на файл `public/index.html` (`public/` — корневая директория веб-сервера по умолчанию). Preact инициализирован на этой странице и через API истории браузера он узнает, какую страницу нужно отрендерить.

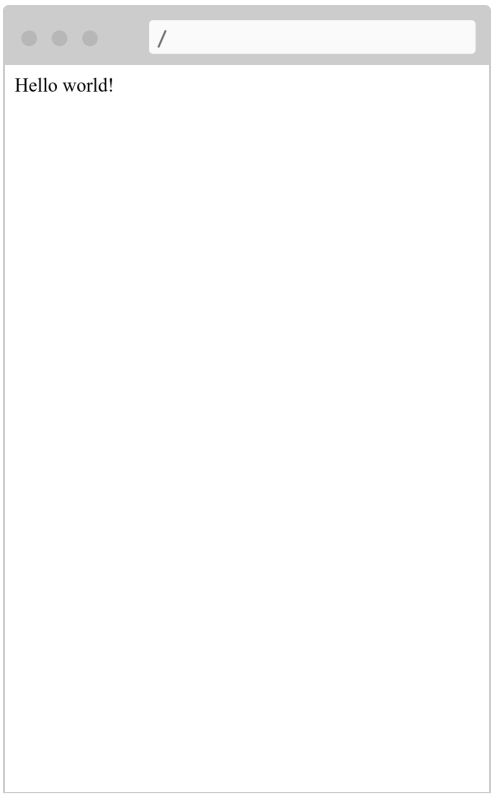
Для сборки CSS- **и JavaScript-файлов** выполните команду `yarn`:

```
$ yarn encore dev
```

Откройте SPA в браузере:

```
$ symfony open:local
```

И посмотрите на надпись «Hello world!», которую вывел SPA:



27.4 Добавление маршрутизатора для обработки состояний

SPA не может обработать несколько страниц. Чтобы добавить их поддержку нам нужен маршрутизатор, по аналогии как в Symfony. Для этого мы будем использовать **preact-router**. Он принимает URL-адрес и сопоставляет его с Preact-компонентом, что его отрендерить страницу.

Установим preact-router:

```
$ yarn add preact-router
```

Создадим главную страницу в виде *компонента Preact*:

src/pages/home.js

```
import {h} from 'preact';

export default function Home() {
    return (
        <div>Home</div>
```

```
    );
};
```

И потом ещё одну страницу для конференций:

src/pages/conference.js

```
import {h} from 'preact';

export default function Conference() {
    return (
        <div>Conference</div>
    );
};
```

Замените элемент `div` с «Hello World» на компонент `Router`:

```
--- a/src/app.js
+++ b/src/app.js
@@ -1,9 +1,22 @@
 import {h, render} from 'preact';
+import {Router, Link} from 'preact-router';
+
+import Home from './pages/home';
+import Conference from './pages/conference';

 function App() {
     return (
         <div>
-            Hello world!
+            <header>
+                <Link href="/">Home</Link>
+                <br />
+                <Link href="/conference/amsterdam2019">Amsterdam 2019</Link>
+            </header>
+
+            <Router>
+                <Home path="/" />
+                <Conference path="/conference/:slug" />
+            </Router>
         </div>
     )
 }
```

Пересоберите приложение:

```
$ yarn encore dev
```

Если вы обновите страницу в браузере, то сможете нажать на ссылки «Home» и конференции. Обратите внимание, что URL-адрес вместе с браузерными кнопками перемещения вперёд и назад работают вполне ожидаемым образом (как и в обычных статичных приложениях).

27.5 Стилизация SPA

Давайте установим загрузчик Sass на сайт:

```
$ yarn add node-sass "sass-loader@^7.0"
```

Включите загрузчик Sass в Webpack, чтобы можно было импортировать таблицу стилей в коде:

```
--- a/src/app.js
+++ b/src/app.js
@@ -1,3 +1,5 @@
+import '../assets/css/app.scss';
+
 import {h, render} from 'preact';
 import {Router, Link} from 'preact-router';

--- a/webpack.config.js
+++ b/webpack.config.js
@@ -7,6 +7,7 @@ Encore
     .cleanupOutputBeforeBuild()
     .addEntry('app', './src/app.js')
     .enablePreactPreset()
+    .enableSassLoader()
     .enableSingleRuntimeChunk()
     .addPlugin(new HtmlWebpackPlugin({ template: 'src/index.ejs',
alwaysWriteToDisk: true }))
 ;
```

Теперь в приложении можно подключить стили:

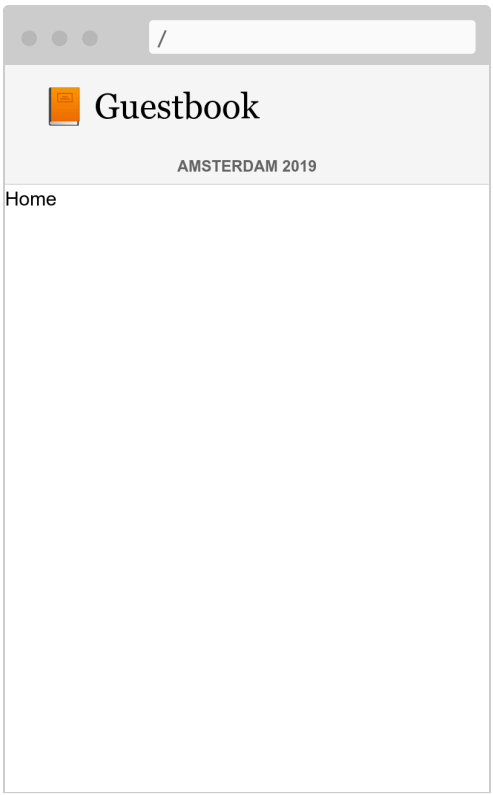
```
--- a/src/app.js
+++ b/src/app.js
@@ -9,10 +9,20 @@ import Conference from './pages/conference';
 function App() {
     return (
         <div>
-            <header>
```

```
-                <Link href="/">Home</Link>
-                <br />
-                <Link href="/conference/amsterdam2019">Amsterdam 2019</Link>
+            <header className="header">
+                <nav className="navbar navbar-light bg-light">
+                    <div className="container">
+                        <Link className="navbar-brand mr-4 pr-2" href="/">
+                            &#128217; Guestbook
+                        </Link>
+                    </div>
+                </nav>
+
+                <nav className="bg-light border-bottom text-center">
+                    <Link className="nav-conference" href="/conference/
amsterdam2019">
+                        Amsterdam 2019
+                    </Link>
+                </nav>
             </header>

             <Router>
```

Пересоберите приложение ещё раз:

```
$ yarn encore dev
```

А сейчас можно насладиться полностью стилизованным SPA:



27.6 Получение данных при помощи API

Итак, структура Preact-приложения закончена: Preact Router управляет отображением страниц, включая обработку динамических URL-адресов каждой конференции. Кроме этого, стили основного приложения используется в SPA.

Чтобы сделать SPA динамическим, получим данные из API, выполнив HTTP-запросы.

С помощью Webpack определим глобальную переменную в приложении с URL-адресом API из соответствующей переменной окружения:

```
--- a/webpack.config.js
+++ b/webpack.config.js
@@ -1,3 +1,4 @@
+const webpack = require('webpack');
 const Encore = require('@symfony/webpack-encore');
 const HtmlWebpackPlugin = require('html-webpack-plugin');
```

```
@@ -10,6 +11,9 @@ Encore
     .enableSassLoader()
     .enableSingleRuntimeChunk()
     .addPlugin(new HtmlWebpackPlugin({ template: 'src/index.ejs',
alwaysWriteToDisk: true }))
+    .addPlugin(new webpack.DefinePlugin({
+        'ENV_API_ENDPOINT': JSON.stringify(process.env.API_ENDPOINT),
+    }))
 ;

 module.exports = Encore.getWebpackConfig();
```

В переменной окружения `API_ENDPOINT` будет храниться адрес точки входа API, который у нас доступен по пути `/api`. Установим её позже, когда начнём выполнять команду `yarn encore`.

Создайте файл `api.js`, в котором будет находится логика получения данных из API:

src/api/api.js

```
function fetchCollection(path) {
    return fetch(ENV_API_ENDPOINT + path).then(resp => resp.json()).then(json
=> json['hydra:member']);
}

export function findConferences() {
    return fetchCollection('api/conferences');
}

export function findComments(conference) {
    return fetchCollection('api/comments?conference='+conference.id);
}
```

Теперь воспользуемся API-методами в корневом компоненте и в компоненте главной страницы:

```
--- a/src/app.js
+++ b/src/app.js
@@ -2,11 +2,23 @@ import '../assets/css/app.scss';

 import {h, render} from 'preact';
 import {Router, Link} from 'preact-router';
+import {useState, useEffect} from 'preact/hooks';

+import {findConferences} from './api/api';
 import Home from './pages/home';
```

```
 import Conference from './pages/conference';

 function App() {
+    const [conferences, setConferences] = useState(null);
+
+    useEffect(() => {
+        findConferences().then((conferences) => setConferences(conferences));
+    }, []);
+
+    if (conferences === null) {
+        return <div className="text-center pt-5">Loading...</div>;
+    }
+
     return (
         <div>
             <header className="header">
@@ -19,15 +31,17 @@ function App() {
                 </nav>

                 <nav className="bg-light border-bottom text-center">
-                    <Link className="nav-conference" href="/conference/
amsterdam2019">
-                        Amsterdam 2019
-                    </Link>
+                    {conferences.map((conference) => (
+                        <Link className="nav-conference"
href={'/conference/'+conference.slug}>
+                            {conference.city} {conference.year}
+                        </Link>
+                    ))}
                 </nav>
             </header>

             <Router>
-                <Home path="/" />
-                <Conference path="/conference/:slug" />
+                <Home path="/" conferences={conferences} />
+                <Conference path="/conference/:slug" conferences={conferences}
/>
             </Router>
         </div>
     )
--- a/src/pages/home.js
+++ b/src/pages/home.js
@@ -1,7 +1,28 @@
 import {h} from 'preact';
+import {Link} from 'preact-router';
+
+export default function Home({conferences}) {
```

```
+    if (!conferences) {
+        return <div className="p-3 text-center">No conferences yet</div>;
+    }

-export default function Home() {
     return (
-        <div>Home</div>
+        <div className="p-3">
+            {conferences.map((conference)=> (
+                <div className="card border shadow-sm lift mb-3">
+                    <div className="card-body">
+                        <div className="card-title">
+                            <h4 className="font-weight-light">
+                                {conference.city} {conference.year}
+                            </h4>
+                        </div>
+
+                        <Link className="btn btn-sm btn-blue stretched-link"
href={'/conference/'+conference.slug}>
+                            View
+                        </Link>
+                    </div>
+                </div>
+            ))}
+        </div>
     );
-};
+}
```

Preact Router передает заполнитель «slug» в качестве свойства компоненту Conference. Используйте его для отображения соответствующей конференции и комментариев к ней через всё тот же API; также изменим компонент конференции, чтобы он использовал данные из API:

```
--- a/src/pages/conference.js
+++ b/src/pages/conference.js
@@ -1,7 +1,48 @@
 import {h} from 'preact';
+import {findComments} from '../api/api';
+import {useState, useEffect} from 'preact/hooks';
+
+function Comment({comments}) {
+    if (comments !== null && comments.length === 0) {
+        return <div className="text-center pt-4">No comments yet</div>;
+    }
+
```

```
+    if (!comments) {
+        return <div className="text-center pt-4">Loading...</div>;
+    }
+
+    return (
+        <div className="pt-4">
+            {comments.map(comment => (
+                <div className="shadow border rounded-lg p-3 mb-4">
+                    <div className="comment-img mr-3">
+                        {!comment.photoFilename ? '' : (
+                            <a href={ENV_API_ENDPOINT+'uploads/
photos/'+comment.photoFilename} target="_blank">
+                                <img src={ENV_API_ENDPOINT+'uploads/
photos/'+comment.photoFilename} />
+                            </a>
+                        )}
+                    </div>
+
+                    <h5 className="font-weight-light mt-3
mb-0">{comment.author}</h5>
+                    <div className="comment-text">{comment.text}</div>
+                </div>
+            ))}
+        </div>
+    );
+}
+
+export default function Conference({conferences, slug}) {
+    const conference = conferences.find(conference => conference.slug ===
slug);
+    const [comments, setComments] = useState(null);
+
+    useEffect(() => {
+        findComments(conference).then(comments => setComments(comments));
+    }, [slug]);

-export default function Conference() {
     return (
-        <div>Conference</div>
+        <div className="p-3">
+            <h4>{conference.city} {conference.year}</h4>
+            <Comment comments={comments} />
+        </div>
     );
-};
+}
```

Теперь нам нужно задать URL-адрес нашего API, присвоив его переменной окружения `API_ENDPOINT`. Используйте для этого URL-

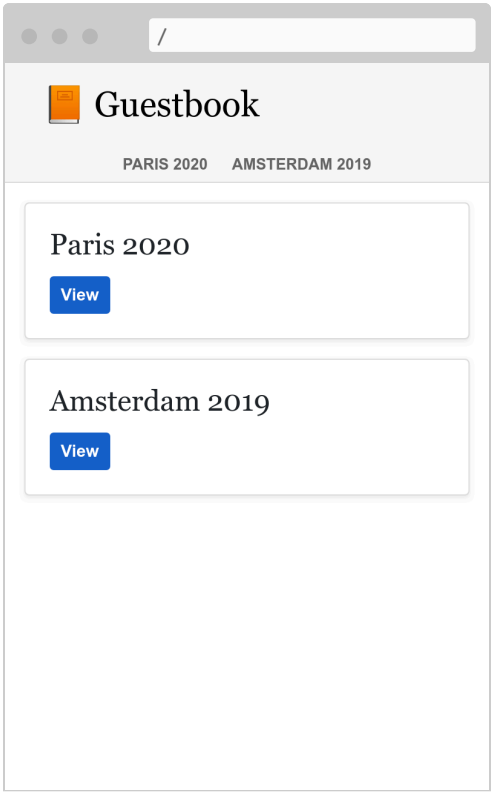
адрес веб-сервера API (запущен в директории `..`):

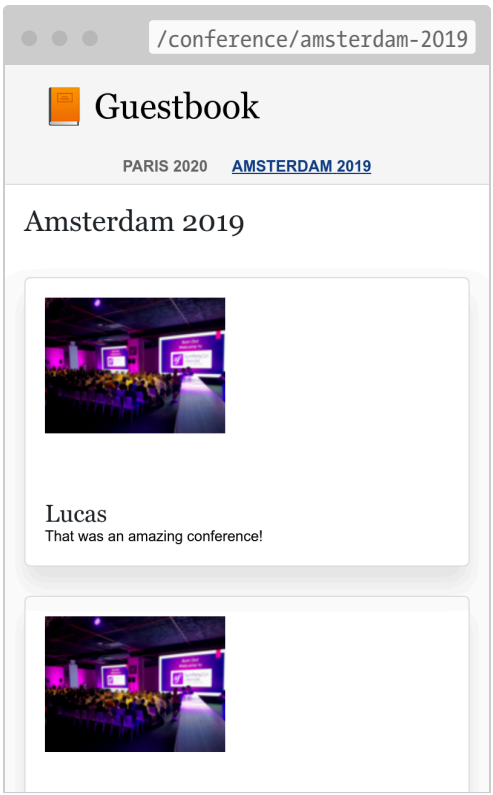
```
$ API_ENDPOINT=`symfony var:export SYMFONY_DEFAULT_ROUTE_URL --dir=..` yarn
encore dev
```

Вы также можете запустить сервер в фоновом режиме:

```
$ API_ENDPOINT=`symfony var:export SYMFONY_DEFAULT_ROUTE_URL --dir=..` symfony
run -d --watch=webpack.config.js yarn encore dev --watch
```

Сейчас приложение в браузере должно работать корректно:





Вау! Теперь у нас есть полностью рабочее SPA-приложение с маршрутизацией и реалистичными данными. Мы можем и дальше улучшать наше приложение на Preact, но оно уже работает отлично.

27.7 Развёртывание SPA в продакшене

SymfonyCloud позволяет развёртывать несколько приложений в рамках одного проекта. Для добавления другого приложения нужен новый файл `.symfony.cloud.yaml` в любой поддиректории. Создайте такой файл в директории `spa/`:

.symfony.cloud.yaml

```
name: spa

type: php:7.3
size: S
disk: 256

build:
```

```
    flavor: none

dependencies:
    nodejs:
        yarn: "*"

web:
    commands:
        start: sleep
    locations:
        "/":
            root: "public"
            index:
                - "index.html"
            scripts: false
            expires: 10m

hooks:
    build: |
        set -x -e

        curl -s https://get.symfony.com/cloud/configurator | (>&2 bash)
        yarn-install
        npm rebuild node-sass
        yarn encore prod
```

Отредактируйте файл `.symfony/routes.yaml` так, чтобы перенаправлять запросы с поддомена `spa.` в приложение `spa`, которое находится в корневой директории проекта:

```
$ cd ../
```

```
--- a/.symfony/routes.yaml
+++ b/.symfony/routes.yaml
@@ -1,2 +1,5 @@
+"https://spa.{all}/": { type: upstream, upstream: "spa:http" }
+"http://spa.{all}/": { type: redirect, to: "https://spa.{all}/" }
+
 "https://{all}/": { type: upstream, upstream: "varnish:http", cache: {
enabled: false } }
 "http://{all}/": { type: redirect, to: "https://{all}/" }
```

27.8 Настройка CORS для SPA

Если попробовать сейчас развернуть приложение, то оно не будет работать, потому что браузер не даст выполнить запрос к API. Чтобы этого не было, нам нужно явно разрешить SPA обращаться к API. Для этого сначала нужно узнать текущий домен, на котором развёрнуто ваше приложение:

```
$ symfony env:urls --first
```

Затем определите переменную окружения `CORS_ALLOW_ORIGIN`, как показано ниже:

```
$ symfony var:set "CORS_ALLOW_ORIGIN=^`symfony env:urls --first | sed 's#/$##'
| sed 's#https://#https://spa.#'`$"
```

К примеру, если у вас домен `https://master-5szvwec-hzhac461b3a6o.eu.s5y.io/`, то после выполнения команды `sed`, он преобразуется в `https://spa.master-5szvwec-hzhac461b3a6o.eu.s5y.io`.

Нам также нужно установить переменную окружения `API_ENDPOINT`:

```
$ symfony var:set API_ENDPOINT=`symfony env:urls --first`
```

Зафиксируйте изменения и разверните:

```
$ git add .
$ git commit -a -m'Add the SPA application'
$ symfony deploy
```

Чтобы автоматически открыть в браузере развёрнутое SPA-приложение, выполните следующую команду:

```
$ symfony open:remote --app=spa
```

27.9 Сборка приложения для смартфона с помощью Cordova

Apache Cordova — это инструмент для создания кроссплатформенных мобильных приложений. Хотя при этом его можно применить с нашим только что созданным SPA.

Давайте установим его:

```
$ cd spa
$ yarn global add cordova
```

Также необходимо установить Android SDK. В этой книге мы добавим поддержку только для Android, хотя Cordova работает со всеми мобильными платформами, включая iOS.

Создайте структуру директорий для приложения:

```
$ cordova create app
```

А теперь сгенерируйте приложение под Android:

```
$ cd app
$ cordova platform add android
$ cd ..
```

Это всё, что нужно. Теперь вы можете собрать файлы приложения и передать в Cordova:

```
$ API_ENDPOINT=`symfony var:export SYMFONY_DEFAULT_ROUTE_URL --dir=..` yarn
encore production
$ rm -rf app/www
$ mkdir -p app/www
$ cp -R public/ app/www
```

Запустите приложение на вашем смартфоне или эмуляторе:

```
$ cordova run android
```

Двигаемся дальше

- *Официальный сайт Preact*;
- *Официальный сайт Cordova <https://cordova.apache.org/>*.

Шаг 28
Локализация приложения

Аудитория Symfony распределена по всему миру и благодаря этому, интернационализация (i18n) и локализация (l10n) уже очень давно доступны из коробки. Локализация приложения заключается не только в переводе интерфейса, но и в обработке форм множественного числа, форматировании даты и валюты, URL-адресов и т. д.

28.1 Интернационализация URL-адресов

Интернационализация URL-адресов — первый шаг к интернационализации сайта. Адрес сайта должен отличаться в зависимости от локали для корректной работы HTTP-кеширования (никогда не используйте одинаковый URL, сохраняя локаль в сессии).

Укажите специальный параметр маршрута `_locale`, чтобы использовать локаль в маршрутах:

```
--- a/src/Controller/ConferenceController.php
+++ b/src/Controller/ConferenceController.php
@@ -34,7 +34,7 @@ class ConferenceController extends AbstractController
     }

     /**
-     * @Route("/", name="homepage")
+     * @Route("/{_locale}/", name="homepage")
      */
     public function index(ConferenceRepository $conferenceRepository)
     {
```

Теперь локаль на главной странице устанавливается в зависимости от URL; например, если вы перейдёте на `/fr/`, вызов `$request->getLocale()` вернёт `fr`.

Как правило, у вас не получится перевести содержимое для всех доступных локалей, поэтому выберите только те, которые вы можете поддерживать:

```
--- a/src/Controller/ConferenceController.php
+++ b/src/Controller/ConferenceController.php
@@ -34,7 +34,7 @@ class ConferenceController extends AbstractController
     }

     /**
-     * @Route("/{_locale}/", name="homepage")
+     * @Route("/{_locale<en|fr>}/", name="homepage")
      */
     public function index(ConferenceRepository $conferenceRepository)
     {
```

Вы можете ограничить любой параметр маршрута, используя регулярное выражение внутри `< >`. Маршрут `homepage` сработает, только если параметр `_locale` будет равен `en` или `fr`. Попробуйте перейти по пути `/es/` и вы увидите ошибку 404, потому что нет подходящего маршрута.

Поскольку мы будем использовать одно и то же ограничение практически во всех маршрутах, давайте переместим его к параметрам контейнера:

```
--- a/config/services.yaml
+++ b/config/services.yaml
```

```
@@ -7,6 +7,7 @@ parameters:
     default_admin_email: admin@example.com
     default_domain: '127.0.0.1'
     default_scheme: 'http'
+    app.supported_locales: 'en|fr'

     router.request_context.host:
'%env(default:default_domain:SYMFONY_DEFAULT_ROUTE_HOST)%'
     router.request_context.scheme:
'%env(default:default_domain:SYMFONY_DEFAULT_ROUTE_SCHEME)%'
--- a/src/Controller/ConferenceController.php
+++ b/src/Controller/ConferenceController.php
@@ -34,7 +34,7 @@ class ConferenceController extends AbstractController
     }

     /**
-     * @Route("/{_locale<en|fr>}/", name="homepage")
+     * @Route("/{_locale<%app.supported_locales%>}/", name="homepage")
      */
     public function index(ConferenceRepository $conferenceRepository)
     {
```

Обновите параметр app.supported_languages, добавив в него новый язык.

Добавьте тот же префикс локали маршрута к другим URL-адресам:

```
--- a/src/Controller/ConferenceController.php
+++ b/src/Controller/ConferenceController.php
@@ -47,7 +47,7 @@ class ConferenceController extends AbstractController
     }

     /**
-     * @Route("/conference_header", name="conference_header")
+     * @Route("/{_locale<%app.supported_locales%>}/conference_header",
name="conference_header")
      */
     public function conferenceHeader(ConferenceRepository
$conferenceRepository)
     {
@@ -60,7 +60,7 @@ class ConferenceController extends AbstractController
     }

     /**
-     * @Route("/conference/{slug}", name="conference")
+     * @Route("/{_locale<%app.supported_locales%>}/conference/{slug}",
name="conference")
      */
```

```
     public function show(Request $request, Conference $conference,
CommentRepository $commentRepository, NotifierInterface $notifier, string
$photoDir)
     {
```

Мы почти закончили. У нас больше нет маршрута, который бы совпадал с `/`. Давайте вернём его и сделаем так, чтобы он перенаправлял на `/en/`:

```
--- a/src/Controller/ConferenceController.php
+++ b/src/Controller/ConferenceController.php
@@ -33,6 +33,14 @@ class ConferenceController extends AbstractController
         $this->bus = $bus;
     }

+    /**
+     * @Route("/")
+     */
+    public function indexNoLocale()
+    {
+        return $this->redirectToRoute('homepage', ['_locale' => 'en']);
+    }
+
     /**
      * @Route("/{_locale<%app.supported_locales%>}/", name="homepage")
      */
```

Теперь, когда все основные маршруты локализованы, обратите внимание, что генерируемые URL-адреса на страницах автоматически включают текущую локаль.

28.2 Добавление переключателя локали

Чтобы пользователи могли переключаться с локали по умолчанию `en` на другую, добавьте переключатель в шапку:

```
--- a/templates/base.html.twig
+++ b/templates/base.html.twig
@@ -34,6 +34,16 @@
                                     Admin
```

```
                                </a>
                            </li>
+<li class="nav-item dropdown">
+    <a class="nav-link dropdown-toggle" href="#" id="dropdown-language"
role="button"
+        data-toggle="dropdown" aria-haspopup="true" aria-expanded="false">
+        English
+    </a>
+    <div class="dropdown-menu dropdown-menu-right" aria-labelledby="dropdown-
language">
+        <a class="dropdown-item" href="{{ path('homepage', {_locale: 'en'})
}}">English</a>
+        <a class="dropdown-item" href="{{ path('homepage', {_locale: 'fr'})
}}">Français</a>
+    </div>
+</li>
                        </ul>
                    </div>
                </div>
```

Для переключения на другую локаль, нам нужно явно передать параметр маршрута `_locale` в функцию `path()`.

Обновите шаблон, чтобы вместо неизменяемой надписи «English» отображалось имя текущей локали:

```
--- a/templates/base.html.twig
+++ b/templates/base.html.twig
@@ -37,7 +37,7 @@
 <li class="nav-item dropdown">
     <a class="nav-link dropdown-toggle" href="#" id="dropdown-language"
role="button"
         data-toggle="dropdown" aria-haspopup="true" aria-expanded="false">
-        English
+        {{ app.request.locale|locale_name(app.request.locale) }}
     </a>
     <div class="dropdown-menu dropdown-menu-right" aria-labelledby="dropdown-
language">
         <a class="dropdown-item" href="{{ path('homepage', {_locale: 'en'})
}}">English</a>
```

`app` — глобальная переменная Twig для доступа к текущему запросу. Чтобы преобразовать локализацию в человеко-читаемую строку, мы используем Twig-фильтр `locale_name`.

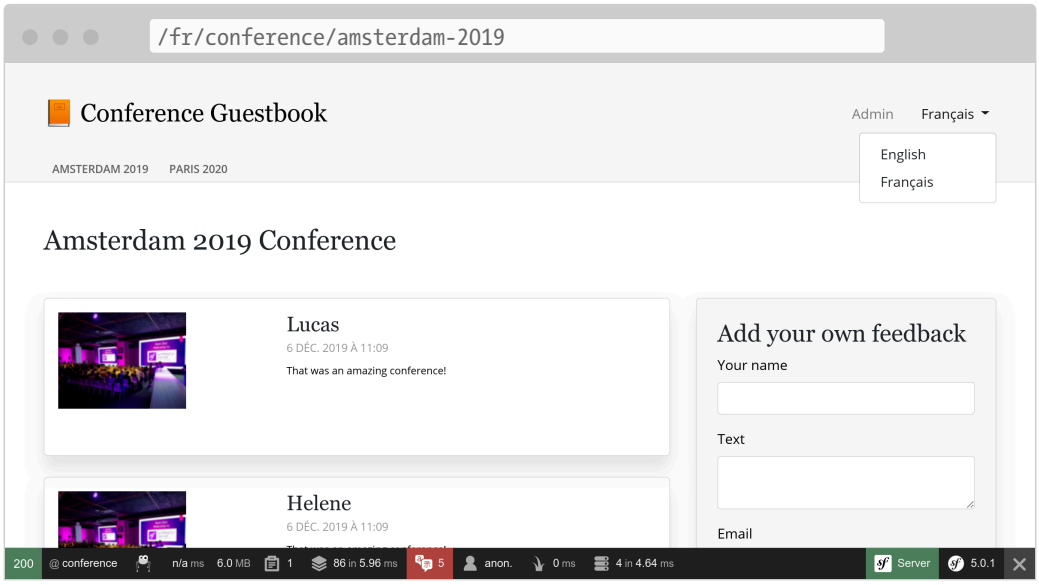
В зависимости от локали, её имя не всегда пишется с большой буквы. Для правильного написания заглавных букв в предложениях нам

нужен фильтр, который поддерживает Юникод. К счастью, такой Twig-фильтр уже реализован, благодаря компоненту Symfony String:

```
$ symfony composer req twig/string-extra
```

```
--- a/templates/base.html.twig
+++ b/templates/base.html.twig
@@ -37,7 +37,7 @@
 <li class="nav-item dropdown">
     <a class="nav-link dropdown-toggle" href="#" id="dropdown-language"
role="button"
         data-toggle="dropdown" aria-haspopup="true" aria-expanded="false">
-        {{ app.request.locale|locale_name(app.request.locale) }}
+        {{ app.request.locale|locale_name(app.request.locale)|u.title }}
     </a>
     <div class="dropdown-menu dropdown-menu-right" aria-labelledby="dropdown-
language">
         <a class="dropdown-item" href="{{ path('homepage', {_locale: 'en'})
}}">English</a>
```

Теперь при помощи переключателя вы можете поменять язык с французского на английский и весь интерфейс подхватит локализацию:



28.3 Перевод интерфейса

Чтобы начать переводить сайт, нужно установить компонент Symfony

Translation:

```
$ symfony composer req translation
```

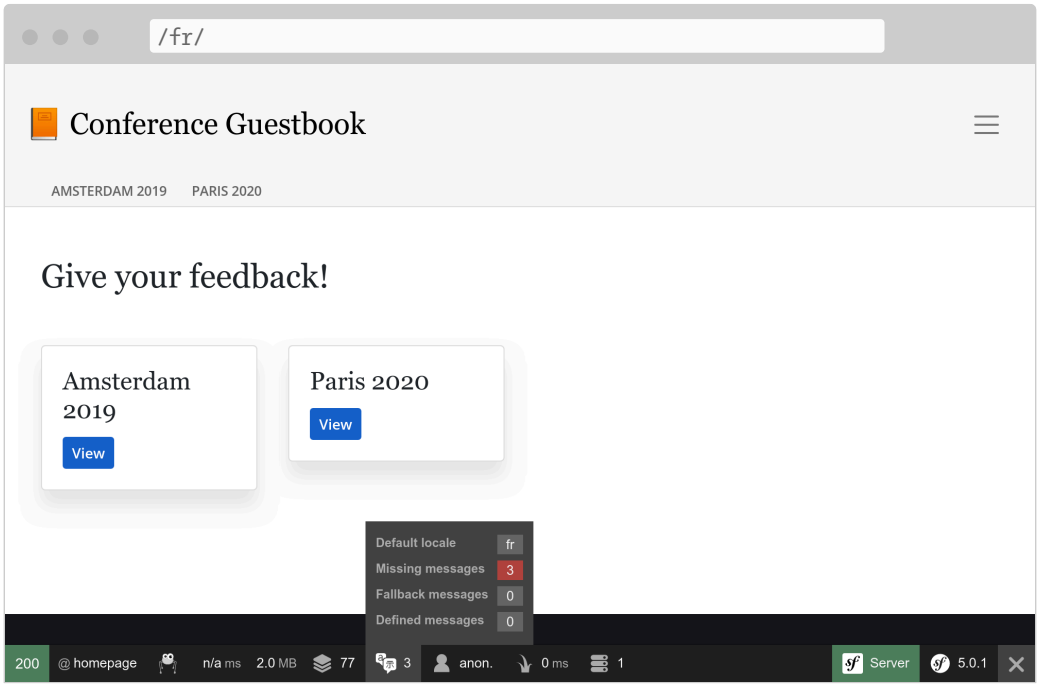
Перевод каждого предложения на большом сайте может быть утомительным, но, к счастью, на нашем сайте всего несколько сообщений. Давайте начнём с предложений на главной странице:

```
--- a/templates/base.html.twig
+++ b/templates/base.html.twig
@@ -20,7 +20,7 @@
             <nav class="navbar navbar-expand-xl navbar-light bg-light">
                 <div class="container mt-4 mb-3">
                     <a class="navbar-brand mr-4 pr-2" href="{{
path('homepage') }}">
-                        &#128217; Conference Guestbook
+                        &#128217; {{ 'Conference Guestbook'|trans }}
                     </a>

                     <button class="navbar-toggler border-0" type="button" data-
toggle="collapse" data-target="#header-menu" aria-
controls="navbarSupportedContent" aria-expanded="false" aria-label="Afficher/
Cacher la navigation">
--- a/templates/conference/index.html.twig
+++ b/templates/conference/index.html.twig
@@ -4,7 +4,7 @@

 {% block body %}
     <h2 class="mb-5">
-        Give your feedback!
+        {{ 'Give your feedback!'|trans }}
     </h2>

     {% for row in conferences|batch(4) %}
@@ -21,7 +21,7 @@

                             <a href="{{ path('conference', { slug:
conference.slug }) }}"
                                class="btn btn-sm btn-blue stretched-link">
-                                View
+                                {{ 'View'|trans }}
                             </a>
                         </div>
                     </div>
```

Twig-фильтр `trans` ищет перевод значения в текущей локализации. Если перевода нет, то возвращается значение из *локали по умолчанию*, указанной в файле `config/packages/translation.yaml`:

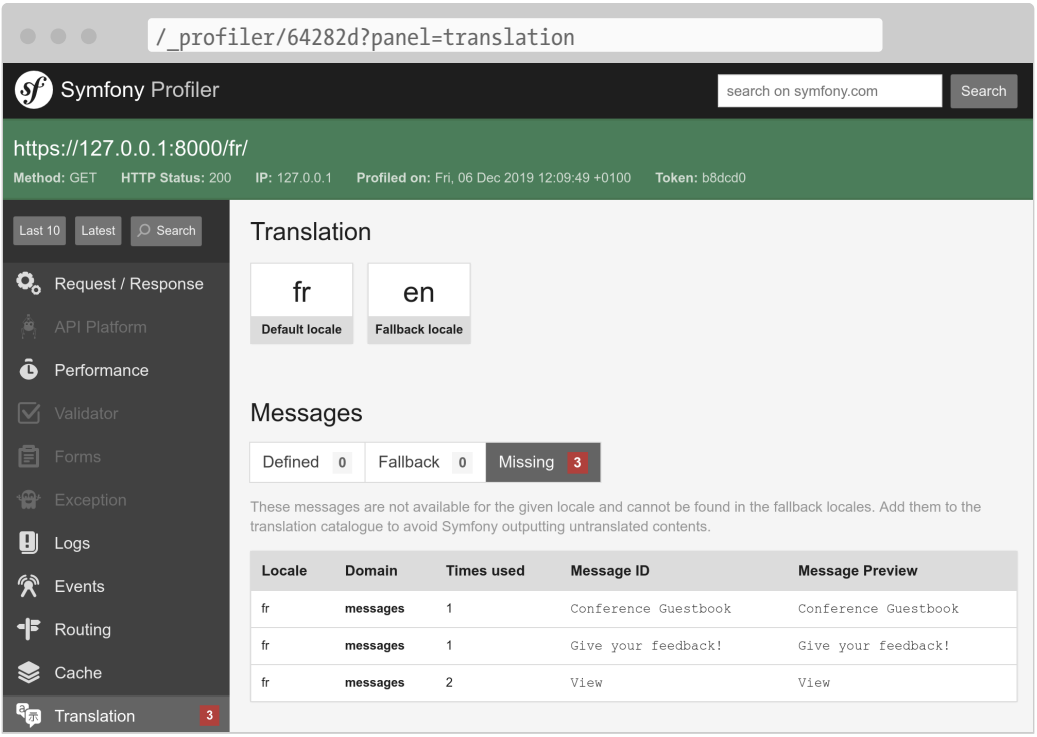
```
framework:
    default_locale: en
    translator:
        default_path: '%kernel.project_dir%/translations'
        fallbacks:
            - en
```

Обратите внимание, что вкладка с переводами на панели отладки стала красной:



Это означает, что ещё не переведены 3 сообщения.

Нажмите на вкладку переводов, чтобы увидеть непереведённые сообщения, которые нашёл Symfony.



28.4 Добавление переводов

Возможно, вы уже заметили в файле `config/packages/translation.yaml`, что переводы хранятся в корневой директории `translations/`, которая была автоматически создана при инициализации проекта.

Чтобы не создавать файлы перевода вручную, воспользуйтесь командой `translation:update`:

```
$ symfony console translation:update fr --force --domain=messages
```

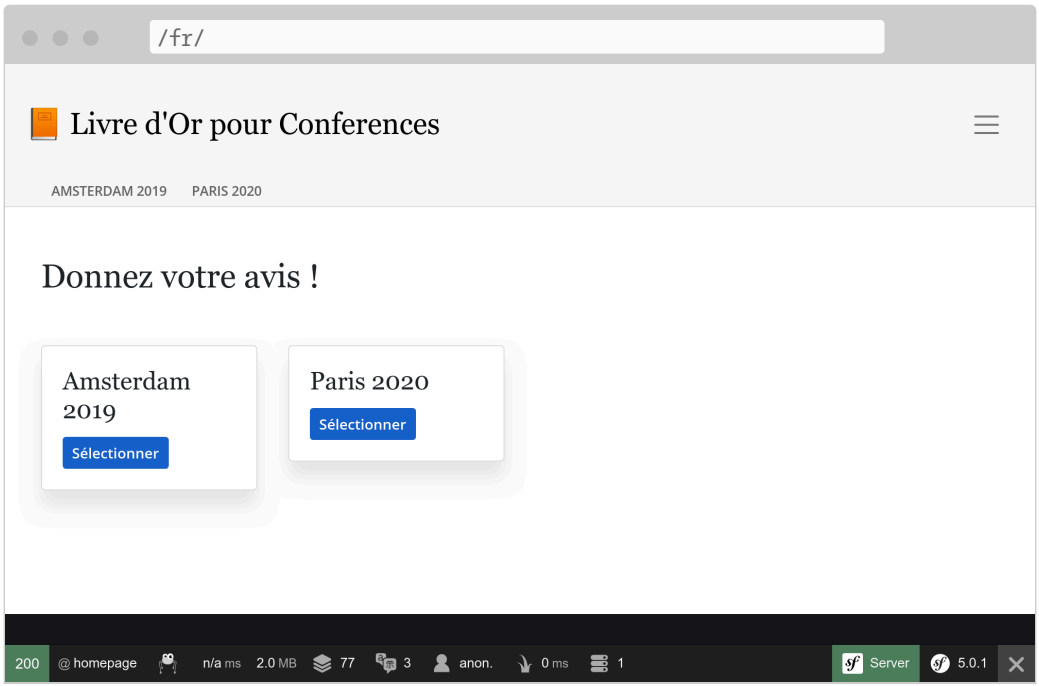
Команда сгенерирует файл перевода (флаг `--force`) для локали `fr` и домена `messages` (содержит все сообщения, характерные для сайта в целом, например, ошибки валидации или безопасности).

В файл `translations/messages+intl-icu.fr.xlf` добавьте перевод на французский. Не говорите по-французски? Давайте я помогу:

```
--- a/translations/messages+intl-icu.fr.xlf
+++ b/translations/messages+intl-icu.fr.xlf
@@ -7,15 +7,15 @@
     <body>
       <trans-unit id="LNAVleg" resname="Give your feedback!">
```

```
        <source>Give your feedback!</source>
-       <target>__Give your feedback!</target>
+       <target>Donnez votre avis !</target>
      </trans-unit>
      <trans-unit id="3Mg5pAF" resname="View">
        <source>View</source>
-       <target>__View</target>
+       <target>Sélectionner</target>
      </trans-unit>
      <trans-unit id="eOy4.6V" resname="Conference Guestbook">
        <source>Conference Guestbook</source>
-       <target>__Conference Guestbook</target>
+       <target>Livre d'Or pour Conferences</target>
      </trans-unit>
    </body>
  </file>
```

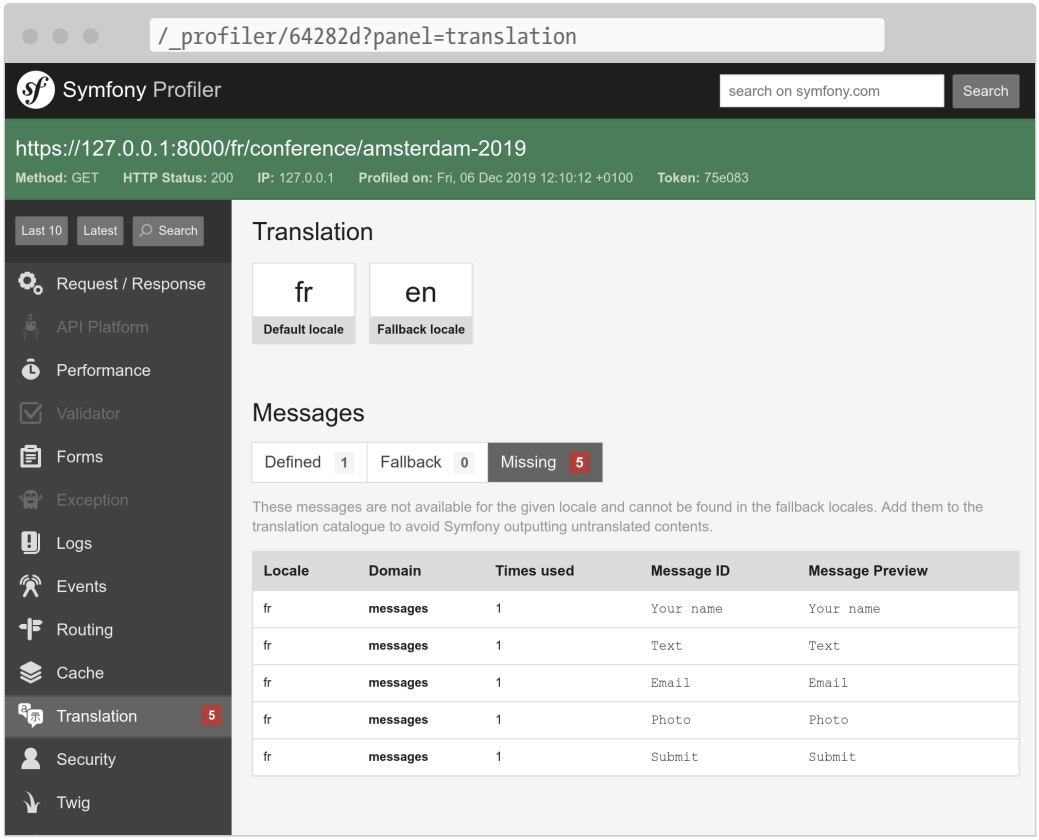
Следует отметить, что мы не будем переводить все шаблоны, но если хотите, сделайте это самостоятельно:



28.5 Перевод форм

Symfony автоматически отображает переведённые метки форм. Перейдите на страницу конференции и нажмите на вкладку с переводами на панели отладки, чтобы увидеть все метки, у которых

отсутствует перевод:



28.6 Локализация дат

If you switch to French and go to a conference webpage that has some comments, you will notice that the comment dates are automatically localized. This works because we used the `format_datetime` Twig filter, which is locale-aware (`{{ comment.createdAt|format_datetime('medium', 'short') }}`).

Локализация поддерживается для дат, времени (`format_time`), валют (`format_currency`) и чисел (`format_number`) в разных форматах (проценты, периоды, прописью и т.д.).

28.7 Перевод сообщений во множественном числе

Управление множественными формами сообщений в переводах — это

одна из распространённых проблем выбора перевода в зависимости от условия.

На странице конференции отображается количество комментариев: `There are 2 comments`. В случае 1 комментария мы неправильно выводим `There are 1 comments`. Поэтому измените шаблон, чтобы преобразовать эту надпись в сообщение для перевода:

```
--- a/templates/conference/show.html.twig
+++ b/templates/conference/show.html.twig
@@ -37,7 +37,7 @@
                         </div>
                     </div>
                 {% endfor %}
-                <div>There are {{ comments|length }} comments.</div>
+                <div>{{ 'nb_of_comments'|trans({count: comments|length}) }}</div>
                 {% if previous >= 0 %}
                     <a href="{{ path('conference', { slug: conference.slug, offset: previous }) }}">Previous</a>
                 {% endif %}
```

Для этого сообщения воспользуемся иной стратегией перевода: английскую версию сообщения в шаблоне заменим на уникальный идентификатор. Такой подход лучше всего работает с большим и сложным текстом.

Добавьте новое сообщение в файл с переводом:

```
--- a/translations/messages+intl-icu.fr.xlf
+++ b/translations/messages+intl-icu.fr.xlf
@@ -17,6 +17,10 @@
         <source>View</source>
         <target>Sélectionner</target>
       </trans-unit>
+      <trans-unit id="Dg2dPd6" resname="nb_of_comments">
+        <source>nb_of_comments</source>
+        <target>{count, plural, =0 {Aucun commentaire.} =1 {1 commentaire.} other {# commentaires.}}</target>
+      </trans-unit>
     </body>
   </file>
 </xliff>
```

Мы ещё не закончили, так как теперь нам необходимо добавить перевод на английский язык. Создайте файл `translations/messages+intl-icu.en.xlf`:

translations/messages+intl-icu.en.xlf

```
<?xml version="1.0" encoding="utf-8"?>
<xliff xmlns="urn:oasis:names:tc:xliff:document:1.2" version="1.2">
  <file source-language="en" target-language="en" datatype="plaintext"
original="file.ext">
    <header>
      <tool tool-id="symfony" tool-name="Symfony"/>
    </header>
    <body>
      <trans-unit id="maMQz7W" resname="nb_of_comments">
        <source>nb of comments</source>
        <target>{count, plural, =0 {There are no comments.} one {There is one
comment.} other {There are # comments.}}</target>
      </trans-unit>
    </body>
  </file>
</xliff>
```

28.8 Обновление функциональных тестов

Не забудьте обновить URL в функциональных тестах:

```
--- a/tests/Controller/ConferenceControllerTest.php
+++ b/tests/Controller/ConferenceControllerTest.php
@@ -11,7 +11,7 @@ class ConferenceControllerTest extends WebTestCase
     public function testIndex()
     {
         $client = static::createClient();
-        $client->request('GET', '/');
+        $client->request('GET', '/en/');

         $this->assertResponseIsSuccessful();
         $this->assertSelectorTextContains('h2', 'Give your feedback');
@@ -20,7 +20,7 @@ class ConferenceControllerTest extends WebTestCase
     public function testCommentSubmission()
     {
         $client = static::createClient();
-        $client->request('GET', '/conference/amsterdam-2019');
+        $client->request('GET', '/en/conference/amsterdam-2019');
         $client->submitForm('Submit', [
             'comment_form[author]' => 'Fabien',
             'comment_form[text]' => 'Some feedback from an automated
functional test',
```

```
@@ -41,7 +41,7 @@ class ConferenceControllerTest extends WebTestCase
     public function testConferencePage()
     {
         $client = static::createClient();
-        $crawler = $client->request('GET', '/');
+        $crawler = $client->request('GET', '/en/');

         $this->assertCount(2, $crawler->filter('h4'));

@@ -50,6 +50,6 @@ class ConferenceControllerTest extends WebTestCase
         $this->assertPageTitleContains('Amsterdam');
         $this->assertResponseIsSuccessful();
         $this->assertSelectorTextContains('h2', 'Amsterdam 2019');
-        $this->assertSelectorExists('div:contains("There are 1 comments")');
+        $this->assertSelectorExists('div:contains("There is one comment")');
     }
 }
```

Двигаемся дальше

- *Перевод сообщений с использованием форматирования ICU*;
- *Использование Twig-фильтров перевода*.

Шаг 29
Оптимизация производительности

> Преждевременная оптимизация — корень всех зол в программировании.

Возможно вы встречали эту цитату и ранее, однако я люблю цитировать её полностью:

> Нам следует забывать о небольшой эффективности, например, в 97% случаев: преждевременная оптимизация — корень всех зол. И напротив, мы должны уделить всё внимание оставшимся 3%.
>
> *—Дональд Кнут*

Даже незначительное увеличение производительности имеет важное значение, особенно для интернет-магазинов. Теперь, когда приложение гостевой книги готово, давайте проверим его

производительность.

Лучший способ определить потенциальные области для оптимизации — использовать *профилировщик*. Одним из наиболее популярных на сегодняшний день является *Blackfire* (*важное примечание*: я также основатель проекта Blackfire).

29.1 Знакомство с Blackfire

Blackfire состоит из нескольких частей:

- *Клиент*, запускающий профили (CLI-инструмент Blackfire или расширение для браузера Google Chrome или Firefox);
- *Агент*, который подготавливает и собирает данные перед их отправкой на сайт blackfire.io для отображения;
- PHP-модуль (*зонд*), который инструментирует PHP-код.

Зарегистрируйтесь, чтобы начать работу с Blackfire.

Установите Blackfire на вашу локальную машину, запустив скрипт быстрой установки:

```
$ curl https://installer.blackfire.io/ | bash
```

Установщик загрузит CLI-инструмент Blackfire, а затем установит PHP-модуль зонда (не активируя его) для всех доступных версий PHP.

Включите PHP-зонд для нашего проекта:

```
--- a/php.ini
+++ b/php.ini
@@ -6,3 +6,7 @@ max_execution_time=30
 session.use_strict_mode=On
 realpath_cache_ttl=3600
 zend.detect_unicode=Off
+
+[blackfire]
+# use php_blackfire.dll on Windows
+extension=blackfire.so
```

Перезапустите веб-сервер, чтобы PHP смог загрузить Blackfire:

```
$ symfony server:stop
$ symfony server:start -d
```

Для хранения профилей приложений в вашей учётной записи, необходимо настроить инструмент Blackfire CLI, используя ваши персональные **клиентские** учётные данные. Найдите их в верху *страницы* `Settings/Credentials` и выполните следующую команду, предварительно подставив свои данные в соответствующие места:

```
$ blackfire config --client-id=xxx --client-token=xxx
```

Полную инструкцию по установке вы сможете найти в *официальном подробном руководстве по установке*. Она также полезна при установке Blackfire на сервер.

29.2 Установка Blackfire-агента в Docker

Последним шагом будет добавление сервиса Blackfire-агента в стек Docker Compose:

```
--- a/docker-compose.yaml
+++ b/docker-compose.yaml
@@ -20,3 +20,8 @@ services:
     mailcatcher:
         image: schickling/mailcatcher
         ports: [1025, 1080]
+
+    blackfire:
+        image: blackfire/blackfire
+        env_file: .env.local
+        ports: [8707]
```

Для соединения с сервером вам необходимо получить ваши персональные **серверные** учётные данные. Эти учётные данные указывают, где вы хотите хранить профили, которые вы можете создать для каждого проекта. Их можно найти внизу *страницы* `Settings/Credentials`. Сохраните данные локально в файле `.env.local`:

```
BLACKFIRE_SERVER_ID=xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
BLACKFIRE_SERVER_TOKEN=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

Теперь запустите новый контейнер:

```
$ docker-compose stop
$ docker-compose up -d
```

29.3 Исправление неработающей установки Blackfire

Если во время профилирования появляется ошибка — увеличьте уровень логирования ошибок Blackfire, чтобы собирать больше информации в отчётах об ошибках:

```
--- a/php.ini
+++ b/php.ini
@@ -10,3 +10,4 @@ zend.detect_unicode=Off
 [blackfire]
 # use php_blackfire.dll on Windows
 extension=blackfire.so
+blackfire.log_level=4
```

Перезапустите веб-сервер:

```
$ symfony server:stop
$ symfony server:start -d
```

И следите за логами:

```
$ symfony server:log
```

Запустите профилирование снова и проверьте записи лога.

29.4 Настройка Blackfire в продакшене

По умолчанию Blackfire добавлен во все проекты на SymfonyCloud.

Сохраните *серверные* учётные данные в переменных окружения:

```
$ symfony var:set BLACKFIRE_SERVER_ID=xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
$ symfony var:set BLACKFIRE_SERVER_TOKEN=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

И активируйте PHP-модуль зонда по аналогии с любым другим PHP-модулем:

```
--- a/.symfony.cloud.yaml
+++ b/.symfony.cloud.yaml
@@ -4,6 +4,7 @@ type: php:7.3

 runtime:
     extensions:
+        - blackfire
         - xsl
         - amqp
         - redis
```

29.5 Настройка Varnish для работы с Blackfire

Перед развёртыванием, чтобы приступить к профилированию, вам необходимо обойти HTTP-кеш Varnish. Если этого не сделать — Blackfire никогда не сможет получить доступ к вашему PHP-приложению. Достаточно разрешить HTTP-запросы профилирования, приходящие только с вашего локального компьютера.

Узнайте ваш текущий IP-адрес:

```
$ curl https://ifconfig.me/
```

И используйте его для настройки Varnish:

```
--- a/.symfony/config.vcl
+++ b/.symfony/config.vcl
@@ -1,3 +1,11 @@
+acl profile {
+    # Authorize the local IP address (replace with the IP found above)
+    "a.b.c.d";
+    # Authorize Blackfire servers
+    "46.51.168.2";
+    "54.75.240.245";
+}
+
 sub vcl_recv {
     set req.backend_hint = application.backend();
     set req.http.Surrogate-Capability = "abc=ESI/1.0";
@@ -8,6 +14,16 @@ sub vcl_recv {
         }
         return (purge);
     }
+
+    # Don't profile ESI requests
+    if (req.esi_level > 0) {
+        unset req.http.X-Blackfire-Query;
+    }
+
+    # Bypass Varnish when the profile request comes from a known IP
+    if (req.http.X-Blackfire-Query && client.ip ~ profile) {
+        return (pass);
+    }
 }

 sub vcl_backend_response {
```

Теперь можно разворачивать.

29.6 Профилирование веб-страниц

Для профилирования обычных веб-страниц в Firefox или Google Chrome используйте *соответствующие расширения*.

Во время профилирования не забудьте отключить HTTP-кеш на вашей локальной машине в файле `public/index.php`. Если этого не выполнить, то вместо своего кода, вы будете профилировать слой HTTP-кеша Symfony.

```
--- a/public/index.php
+++ b/public/index.php
@@ -24,7 +24,7 @@ if ($trustedHosts = $_SERVER['TRUSTED_HOSTS'] ??
$_ENV['TRUSTED_HOSTS'] ?? false
 $kernel = new Kernel($_SERVER['APP_ENV'], (bool) $_SERVER['APP_DEBUG']);

 if ('dev' === $kernel->getEnvironment()) {
-    $kernel = new HttpCache($kernel);
+//    $kernel = new HttpCache($kernel);
 }

 $request = Request::createFromGlobals();
```

Для получения наиболее полного представления о производительности приложения в продакшене, вам также необходимо профилировать окружение продакшена («production»). По умолчанию ваше локальное окружение использует среду разработки («development»), что влечёт за собой существенные накладные расходы (это происходит в основном из-за сбора данных для отладочной панели и Symfony-профилировщика).

Переключите окружение вашей локальной машины на работу в продакшене путём изменения значения переменной окружения `APP_ENV` в файле `.env.local`:

```
APP_ENV=prod
```

Также вы можете использовать команду `server:prod`:

```
$ symfony server:prod
```

Не забудьте переключиться обратно на окружение разработки после завершения сеанса профилирования:

```
$ symfony server:prod --off
```

29.7 Профилирование API-ресурсов

Профилирование API или SPA лучше выполнять в командной строке с

помощью установленного ранее инструмента Blackfire CLI:

```
$ blackfire curl `symfony var:export SYMFONY_DEFAULT_ROUTE_URL`api
```

Команда `blackfire curl` принимает те же аргументы и опции, что и *cURL*.

29.8 Сравнение производительности

В шаге про кеширование для повышения производительности мы добавили слой кеширования, однако мы не проверили, как внесённые изменения повлияли на производительность. Так как довольно сложно угадать, что будет работать быстрее, а что медленнее, то можно оказаться в ситуации, когда из-за оптимизации чего-либо ваше приложение на самом деле станет только медленнее.

С помощью профилировщика всегда необходимо измерять влияние каждой сделанной оптимизации. Blackfire позволяет упростить визуальную оценку производительности благодаря *возможности сравнения*.

29.9 Написание функциональных тестов по принципу чёрного ящика

Мы уже знакомы с тем, как писать функциональные тесты с помощью Symfony. Blackfire, в свою очередь, может быть использован для написания браузерных сценариев, которые можно запускать по желанию с помощью приложения *Blackfire player*. Давайте напишем сценарий, который отправит новый комментарий и проверит его по ссылке в электронной почте в окружении для разработки, а также в административной панели в продакшене.

Создайте файл `.blackfire.yaml` со следующим содержимым:

.blackfire.yaml

```
scenarios: |
    #!blackfire-player

    group login
        visit url('/login')
        submit button("Sign in")
            param username "admin"
            param password "admin"
            expect status_code() == 302

    scenario
        name "Submit a comment on the Amsterdam conference page"
        include login
        visit url('/fr/conference/amsterdam-2019')
            expect status_code() == 200
        submit button("Submit")
            param comment_form[author] 'Fabien'
            param comment_form[email] 'me@example.com'
            param comment_form[text] 'Such a good conference!'
            param comment_form[photo] file(fake('image', '/tmp', 400, 300,
'cats'), 'awesome-cat.jpg')
            expect status_code() == 302
        follow
            expect status_code() == 200
            expect not(body() matches "/Such a good conference/")
            # Wait for the workflow to validate the submissions
            wait 5000
        when env != "prod"
            visit url(webmail_url ~ '/messages')
                expect status_code() == 200
                set message_ids json("[*].id")
            with message_id in message_ids
                visit url(webmail_url ~ '/messages/' ~ message_id ~ '.html')
                    expect status_code() == 200
                    set accept_url css("table a").first().attr("href")
                visit url(accept_url)
                    # we don't check the status code as we can deal
                    # with "old" messages which do not exist anymore
                    # in the DB (would be a 404 then)
        when env == "prod"
            visit url('/admin/?entity=Comment&action=list')
                expect status_code() == 200
                set comment_ids css('table.table tbody tr').extract('data-id')
            with id in comment_ids
                visit url('/admin/comment/review/' ~ id)
                    # we don't check the status code as we scan all comments,
                    # including the ones already reviewed
        visit url('/fr/')
            wait 5000
```

```
        visit url('/fr/conference/amsterdam-2019')
            expect body() matches "/Such a good conference/"
```

Загрузите Blackfire player для выполнения сценария на локальной машине:

```
$ curl -OLsS https://get.blackfire.io/blackfire-player.phar
$ chmod +x blackfire-player.phar
```

Запустите этот сценарий в окружении разработки:

```
$ ./blackfire-player.phar run --endpoint=`symfony var:export
SYMFONY_DEFAULT_ROUTE_URL` .blackfire.yaml --variable "webmail_url=`symfony
var:export MAILCATCHER_URL 2>/dev/null`" --variable="env=dev"
```

Или в продакшене:

```
$ ./blackfire-player.phar run --endpoint=`symfony env:urls --first`
.blackfire.yaml --variable "webmail_url=NONE" --variable="env=prod"
```

Сценарии Blackfire, также могут использовать профили для каждого запроса и запускать тесты производительности, если добавить соответствующий флаг `--blackfire`.

29.10 Автоматизация проверок производительности

Отслеживание производительности позволяет понять не только как улучшить производительность существующего кода, но также и контролировать её падение, вызванное новыми изменениями.

Написанный в предыдущем разделе сценарий может запускаться автоматически через непрерывную интеграцию или на регулярной основе в продакшене.

На SymfonyCloud также возможно *запускать сценарии <https://blackfire.io/docs/integrations/paas/symfonycloud#builds-level-enterprise>* при создании новой ветки или развёртывании в продакшене, чтобы автоматически проверять производительность нового кода.

Двигаемся дальше

- *Книга Blackfire: объяснение производительности PHP-кода*;
- *Обучающий видеокурс по Blackfire на SymfonyCasts*.

Шаг 30
Изучение внутренностей Symfony

Мы уже достаточно долгое время используем Symfony и разработали внушительное приложение, при этом большая часть выполняемого кода является самим Symfony. Несколько сотен строк нашего кода и тысячи фреймворка.

Мне нравится узнавать, как всё работает изнутри. Я всегда был очарован инструментами, помогающими мне докопаться до сути. Первый раз, когда я использовал пошаговый отладчик или когда открыл для себя `ptrace` — это волшебные воспоминания.

Хотите лучше понять, как работает Symfony? Тогда самое время разобраться, что происходит под капотом вашего приложения. Вместо объяснения довольно скучной обработки HTTP-запроса с теоретической точки зрения, мы будем использовать Blackfire, чтобы наглядно посмотреть как всё работает, а также познакомиться с продвинутыми темами.

30.1 Разбираемся во внутренностях Symfony и Blackfire

Вы уже знаете, что все HTTP-запросы обрабатываются в одной точке входа — в файле `public/index.php`. Но что происходит потом? Как вызываются контроллеры?

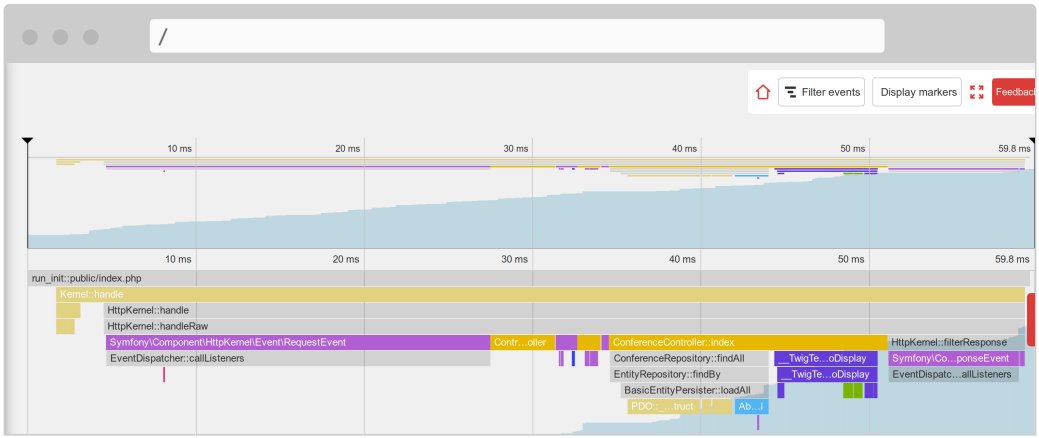
Давайте спрофилируем английскую главную страницу на продакшене с помощью браузерного расширения Blackfire:

```
$ symfony remote:open
```

Или непосредственно через командную строку:

```
$ blackfire curl `symfony env:urls --first`en/
```

Откройте вкладку «Timeline» в профилировщике и вы увидите что-то похожее на это:



Наведите курсор на цветные полосы, расположенные на временной шкале, чтобы получить больше информации о каждом вызове; вы узнаете много нового о том, как работает Symfony:

- Основной точкой входа является файл `public/index.php`;
- Метод `Kernel::handle()` обрабатывает запрос;
- Он вызывает `HttpKernel`, который отправляет несколько событий;
- Первым создаётся событие `RequestEvent`;

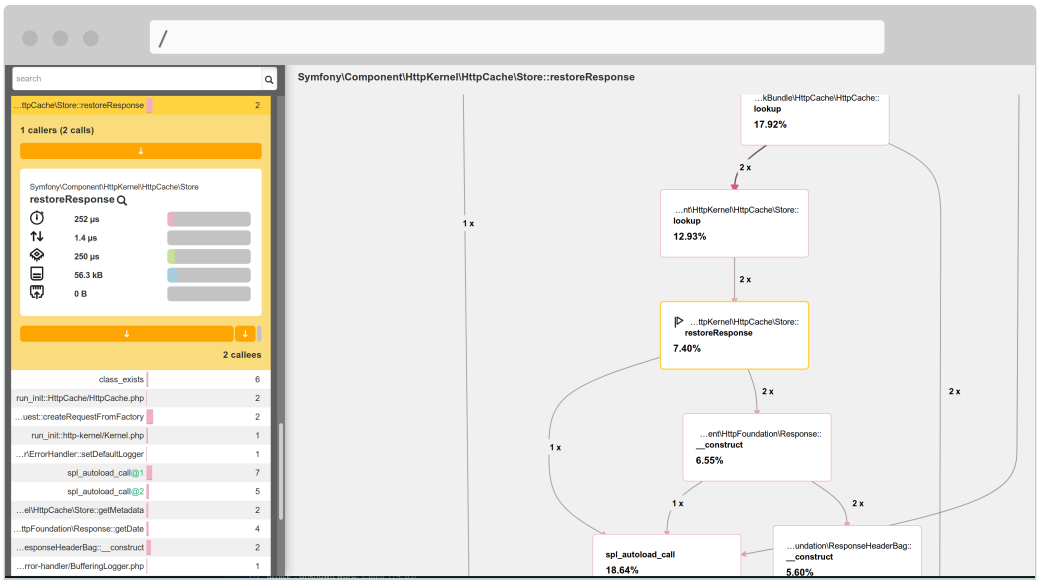
- Вызывается метод `ControllerResolver::getController()` для определения, какой контроллер должен обрабатывать входящий URL-адрес;
- Вызывается метод `ControllerResolver::getArguments()` для определения того, какие аргументы нужно передать контроллеру (для этого используется преобразователь параметров);
- Вызывается метод `ConferenceController::index()` и большая часть нашего кода выполняется этим вызовом;
- Метод `ConferenceRepository::findAll()` получает все конференции из базы данных (обратите внимание на соединение с базой данных с помощью `PDO::__construct()`);
- Метод `Twig\Environment::render()` отображает шаблон;
- Довольно быстро выполнились `ResponseEvent` и `FinishRequestEvent`, но, видимо, из-за того, что нет обработчиков этих событий.

Временная шкала — отличный способ понять, как работает код; она крайне полезна в случае унаследованного проекта, разработанного кем-то другим.

Теперь профилируйте ту же страницу с локального компьютера в окружении разработки:

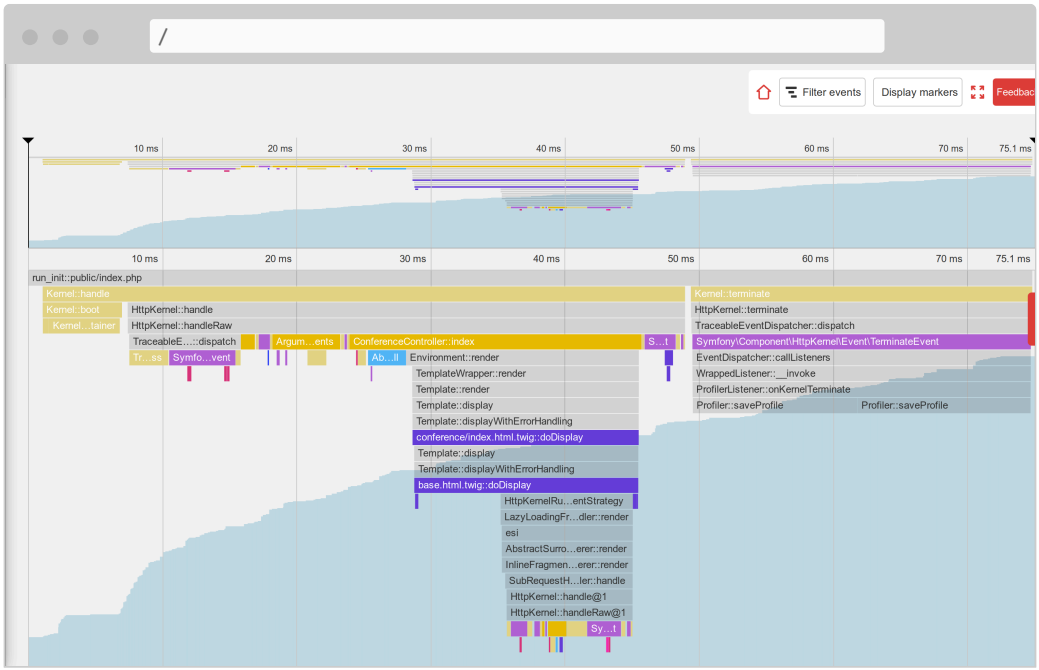
```
$ blackfire curl `symfony var:export SYMFONY_DEFAULT_ROUTE_URL`en/
```

Откройте профилировщик. Вас перенаправит на граф вызовов, так как запрос очень быстро выполнился, то и временная шкала будет совершенно пустой:



Вы понимаете, что происходит сейчас? У нас включено HTTP-кеширование, поэтому мы профилируем кеширующий слой Symfony HTTP. А так как страница закеширована, вызов `HttpCache\Store::restoreResponse()` получает HTTP-ответ из кеша, следовательно контроллер никогда не будет вызван.

Отключите кеширующий слой в `public/index.php`, как мы делали это в предыдущем шаге, и попробуйте снова профилировать код. Вы сразу увидите, что результат будет совершенно другим:



Основные различия заключаются в следующем:

- Обработка события `TerminateEvent`, которая не была видна в продакшене, выполняется довольно много времени; При внимательном рассмотрении, вы увидите, что это событие отвечает за хранение данных профилировщика Symfony, собранных во время запроса;
- Под вызовом `ConferenceController::index()` обратите внимание на метод `SubRequestHandler::handle()`, который отображает ESI (поэтому у нас два вызова `Profiler::saveProfile()`: один для основного запроса, второй — для ESI).

Изучите временную шкалу, чтобы узнать больше о выполненном коде. Затем переключитесь на граф вызовов для получения другого представления данных.

Как мы только что выяснили, код, выполняемый при разработке и в продакшене, совершенно разный. В окружении разработки код работает медленнее, так как профилировщик Symfony пытается собрать больше данных, чтобы облегчить отладку. Поэтому профилировать код нужно всегда в продакшен-окружении, даже локально.

Проведите несколько интересных экспериментов: профилируйте страницу с ошибкой, страницу с путем“/“ (которая является редиректом) или ресурс API. Каждое профилирование расскажет вам немного больше о том, как работает Symfony, какой класс или метод вызывается, что работает долго, а что — быстро.

30.2 Использование отладчика Blackfire Debug Addon

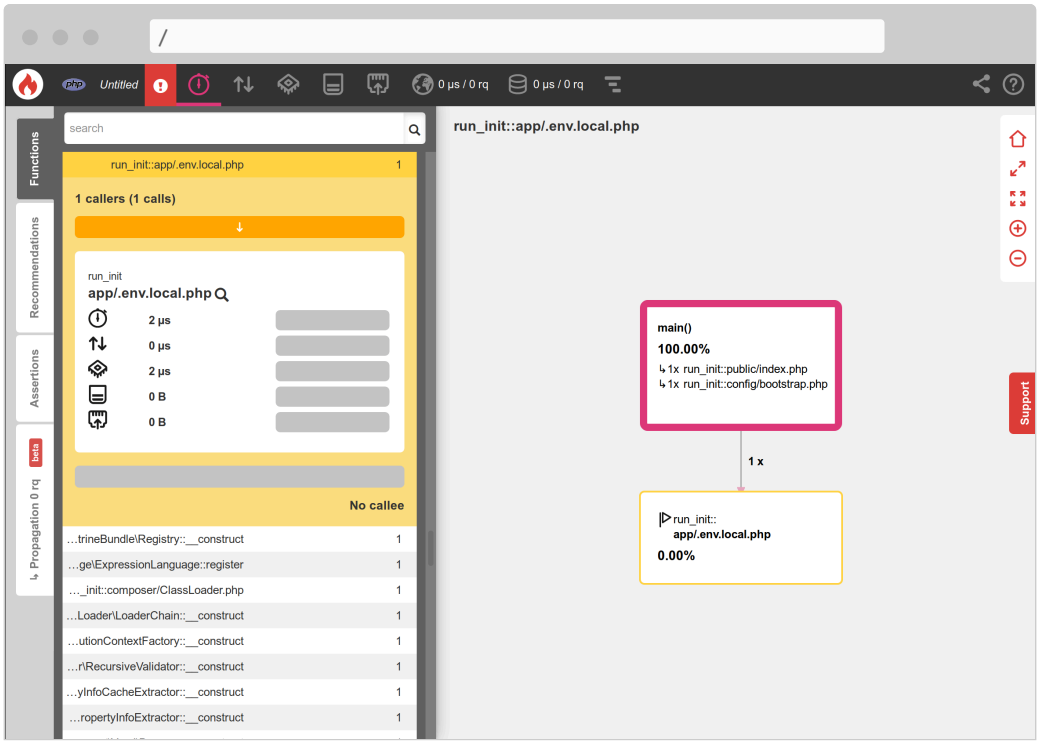
По умолчанию Blackfire удаляет вызовы незначительных методов, чтобы не создавать лишней нагрузки и больших графиков. При использовании Blackfire в качестве инструмента отладки лучше сохранять все вызовы. Для этого воспользуйтесь дополнением отладки.

В командной строке используйте флаг `--debug`:

```
$ blackfire --debug curl `symfony var:export SYMFONY_DEFAULT_ROUTE_URL`en/
```

```
$ blackfire --debug curl `symfony env:urls --first`en/
```

В продакшене вы увидите, например, загрузку файла `.env.local.php`:



Откуда он взялся? SymfonyCloud выполняет оптимизацию при развёртывании приложения Symfony, например, оптимизирует автозагрузчик Composer (`--optimize-autoloader --apcu-autoloader --classmap-authoritative`). Также, чтобы не парсить файл `.env` с переменными окружения при каждом запросе, сгенерируем файл `.env.local.php`:

```
$ symfony run composer dump-env prod
```

Blackfire — очень производительный инструмент, который помогает понять, как выполняется PHP-код. Улучшение производительности — это только один из способов использования профилировщика.

Шаг 31
Что дальше?

Надеюсь, вам понравилось наше путешествие. Я постарался дать достаточно информации, чтобы вы могли как можно скорее начать разрабатывать собственные проекты на Symfony. Мы лишь прошлись по верхам того, что предлагает Symfony. Теперь самое время тщательно изучить остальные страницы документации Symfony, чтобы подробно разобраться во всех возможностях, которые мы вместе открыли.

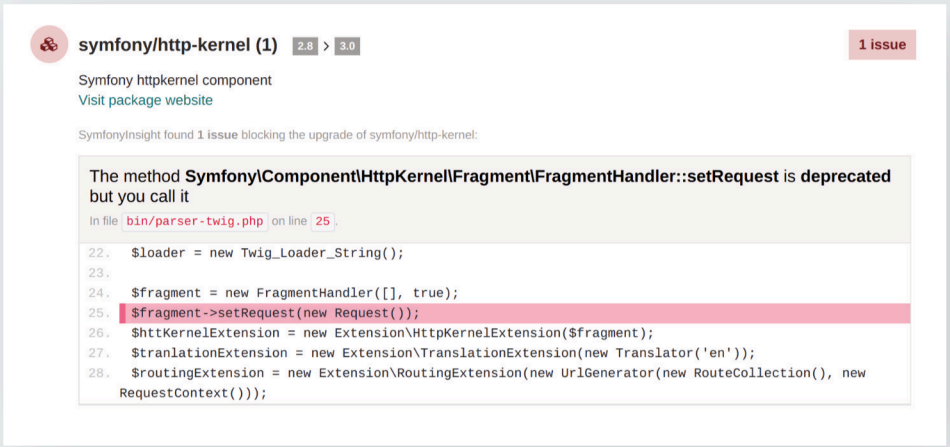
Удачной разработки на Symfony!

The more I live, the more I learn.
The more I learn, the more I realize, the less I know.
— Michel Legrand

Предметный указатель

Not yet on Symfony 5? Upgrade with confidence!

SymfonyInsight spots the deprecations in your projects and helps you fix them



Upgrade continuously

Monitor technical debt

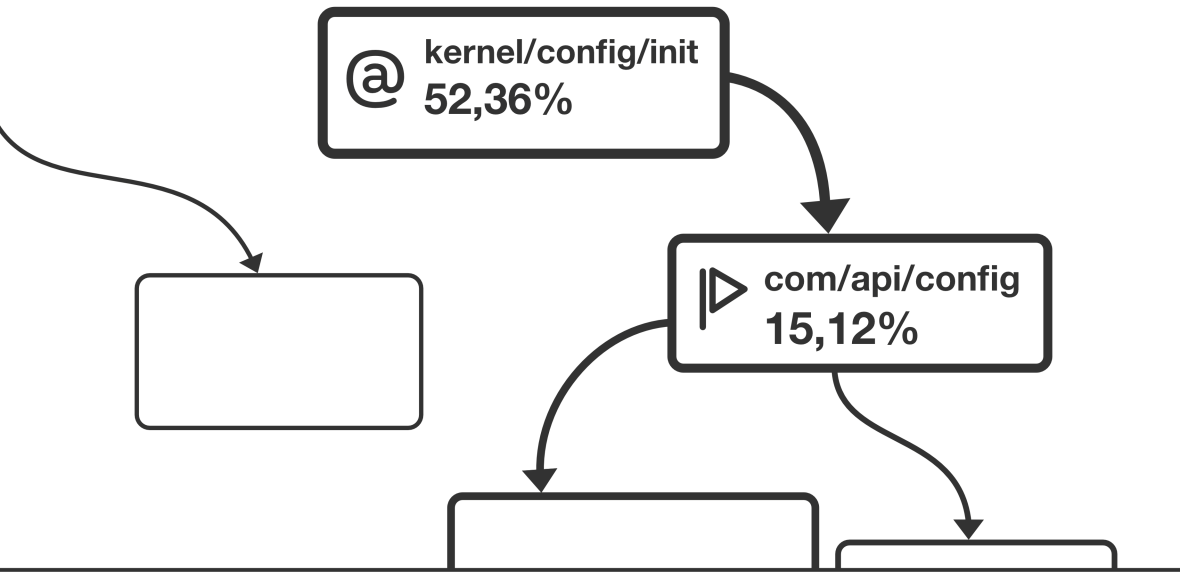
Be notified of security issues

Get started on **insight.symfony.com**

blackfire

Performance Profiling and Testing for Symfony

Profile,
Test,
Fix,
Repeat.



blackfire.io