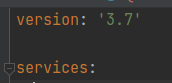
**1. Для начала создайте новый каталог с именем symfony\_docker** и переключитесь на него.

**2. Создайте конфигурацию Docker Compose.**

Поскольку различные контейнеры, составляющие наше приложение, должны взаимодействовать, мы будем использовать Docker Compose для их определения. В корне каталога symfony\_docker создайте новый файл с именем docker-compose.yml.

Этот файл будет содержать всю конфигурацию контейнеров, которые будут созданы в нашем стеке приложений, от того, как должны быть построены контейнеры, до сетей и томов, доступных для контейнеров.

В docker-compose.yml добавьте приведенную ниже конфигурацию.



version относится к версии схемы. services определяет список контейнеров, из которых будет состоять наш стек приложений.

В следующих разделах мы опишем контейнеры для нашей базы данных MySQL, PHP и веб-сервера Nginx.

**3. Определить контейнер базы данных**

Чтобы определить контейнер базы данных, в файле docker-compose.yml обновите элемент services, чтобы он соответствовал следующему примеру:

container\_name задает фактическое имя контейнера при его запуске, а не позволяет Docker Compose сгенерировать его.

image сообщает Docker, из какого образа мы хотим построить контейнер. В данном случае мы указали mysql:8.0, потому что хотим использовать версию 8 MySQL.

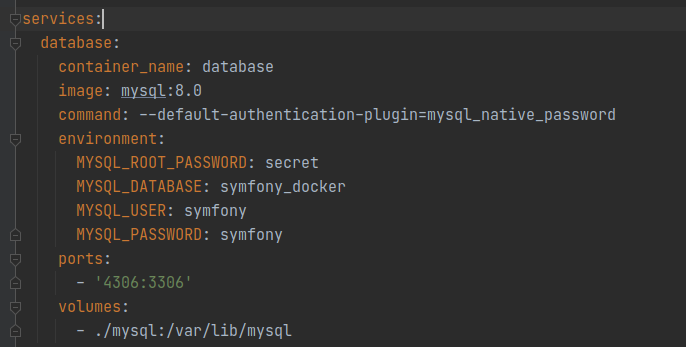
command указывает подключаемый модуль аутентификации, который будет использоваться MySQL для аутентификации пользователей. Используя ключ среды, мы можем указать переменные среды, такие как имя базы данных, пользователя и пароль, а также пароль пользователя root.

Нам нужен порт для подключения к нашей базе данных. Используя ключ ports, мы указываем порт на нашей локальной машине разработки и сопоставляем его с портом в контейнере, который будет использоваться для обработки соединений с базой данных.

Наконец, мы объявляем том, используя ключ volume. Согласно документации Docker:

Тома являются предпочтительным механизмом для сохранения данных, созданных и используемых контейнерами Docker.

В этом случае мы объявляем том, чтобы наша база данных не была потеряна при уничтожении или перестроении контейнеров.



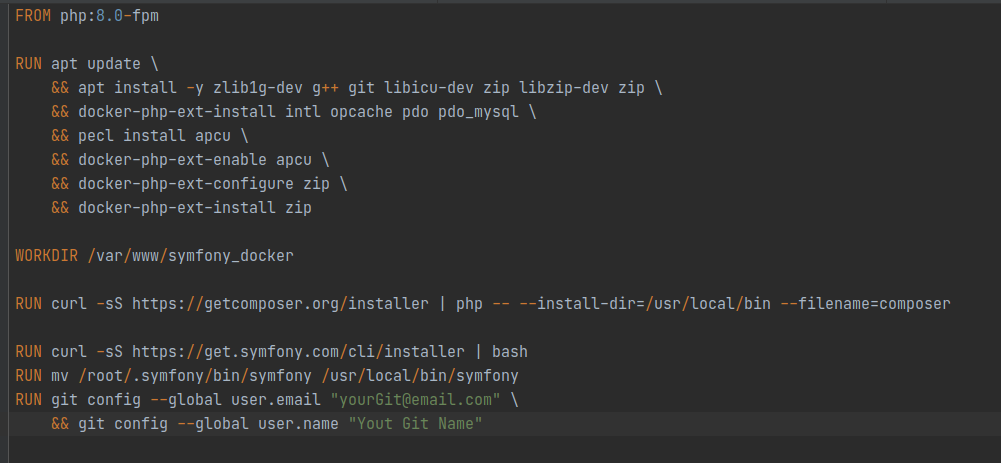
**4. Определить контейнер PHP**

В отличие от контейнера базы данных, нам нужно указать некоторые дополнительные инструкции для настройки нашего контейнера PHP. Для этого мы создадим контейнер PHP из Dockerfile.

В корневом каталоге symfony\_docker создайте каталог с именем php. Затем в symfony\_docker/php создайте файл с именем Dockerfile.

П*римечание: этот файл не имеет расширения.*

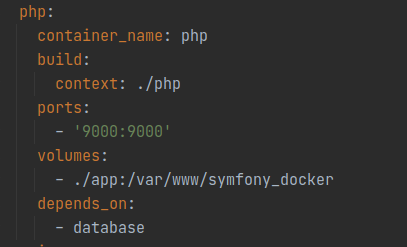
Затем в symfony\_docker/php/Dockerfile добавьте следующее.



Примечание. В последних двух строках файла symfony\_docker/php/Dockerfile замените «yourGit@email.com» своим адресом электронной почты, а «Yout Git Name» — своим настоящим именем.

Помимо создания контейнера из образа PHP-FPM, мы сделаем следующее:

1. Установка расширения PHP, от которых зависит Symfony.
2. Установка рабочего каталога контейнера в /var/www/symfony\_docker.
3. Установка composer
4. Установка интерфейса командной строки Symfony
5. Настройка Git

Затем добавьте приведенный ниже пример в файл docker-compose.yml после настройки базы данных.

Как видно, контейнер PHP определяется по-другому. Вместо указания образа мы указываем контекст сборки. Таким образом, при запуске команды docker-compose инструкции, объявленные в php/Dockerfile, будут использоваться для сборки контейнера.

Порт 9000 на компьютере сопоставляется с портом 9000 на контейнере, так же как мы сопоставляем порт на компьютере с портом на контейнере для базы данных MySQL.

Мы снова объявляем том, чтобы сохранить данные, сгенерированные контейнером. В этом случае наше приложение Symfony будет создано в каталоге /var/www/symfony\_docker контейнера PHP. Однако он будет сохранен в каталоге приложения в проекте.

Наконец, был использован ключ depend\_on. Он создает зависимость между контейнерами PHP и базы данных, предписывая Docker построить и запустить контейнер базы данных перед контейнером PHP.

**5. Создание каталога приложения.**

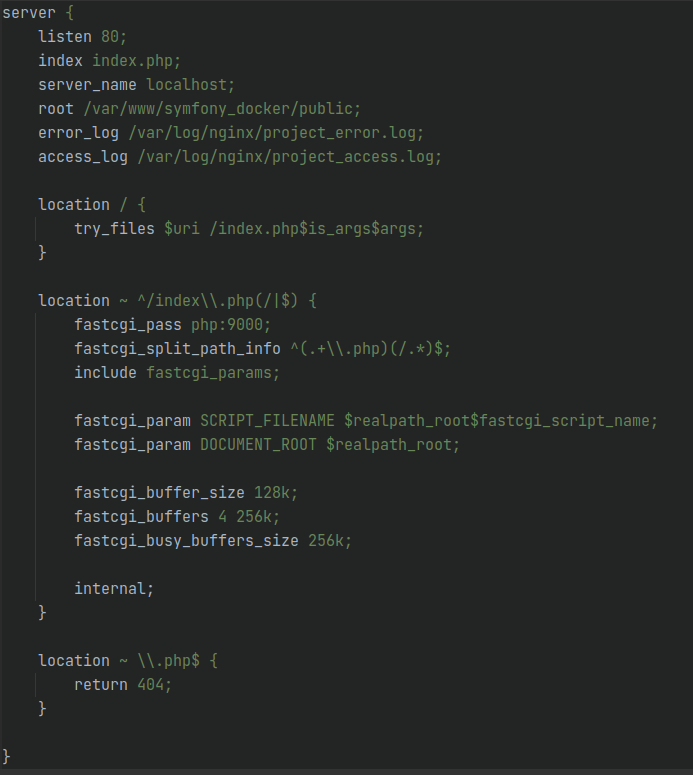
Определив контейнер PHP, создайте каталог приложения в корневом каталоге проекта, который требуется для контейнера, с помощью приведенной ниже команды.



**6.Определите контейнер Nginx**

Прежде чем мы создадим контейнер Nginx, давайте напишем конфигурацию по умолчанию для сервера. В корне проекта создайте каталог с именем nginx и в нем создайте файл конфигурации с именем default.conf

Добавьте приведенную ниже конфигурацию в nginx/default.conf



Это базовая конфигурация Nginx, необходимая для запуска проекта Symfony. Единственное, что сделано по-другому, это спецификация для fastcgi\_pass. Обратите внимание, что в этом случае мы указываем порт 9000 контейнера PHP, потому что это порт по умолчанию, который PHP-FPM прослушивает для запросов.

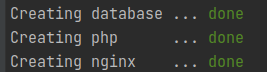
Затем добавьте конфигурацию контейнера Nginx в файл docker-compose.yml в приведенном ниже примере после конфигурации контейнера PHP.



**7. Запустите контейнеры**



Когда контейнеры будут построены, вы увидите в своем терминале что-то похожее на скриншот ниже.



**8. Создайте приложение Symfony**

Запустите команду docker exec. Для этого выполните команду ниже



В только что открытом терминале убедитесь, что ваша установка соответствует требованиям для приложения Symfony, выполнив следующую команду.



Затем создайте новый проект Symfony, выполнив следующую команду.



В случае успеха вы увидите следующий текст в выводе терминала:



Создав приложение, перейдите по адресу http://localhost:8080/, и вы увидите индексную страницу Symfony по умолчанию. Кроме того, если вы посмотрите в каталог приложения, вы увидите, что проект Symfony сохранился там.

Вернувшись в CLI контейнера PHP, давайте добавим некоторые зависимости разработки для приложения, которое мы создаем. Для этого выполните следующую команду.



Для сохранения и извлечения котировок в базе данных и из нее нам понадобится ORM, Doctrine ORM,

и механизм шаблонов Twig потребуется для рендеринга внешнего интерфейса. Добавьте эти зависимости, выполнив команду ниже



Затем создайте файл .env.local из существующего файла .env, который Symfony сгенерировал во время создания проекта. Для этого выполните команду ниже.



Наконец, обновите параметры базы данных в .env.local, чтобы ваше приложение могло подключаться к контейнеру базы данных. Замените текущую запись DATABASE\_URL в файле приведенной ниже версией.

**DATABASE\_URL="mysql://root:secret@database:3306/symfony\_docker?serverVersion=8.0"**

**9. Создайте сущность цитаты**

Далее нам нужно создать объект ORM, который будет обрабатывать взаимодействие с базой данных MySQL. Для этого используйте пакет Symfony Maker, выполнив команду ниже.



Ответьте на заданные вопросы, как показано ниже.

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

> quote

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

> text

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

> no

updated: src/Entity/Quote.php

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

> historian

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

> string

Field length [255]:

> 50

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

> no

updated: src/Entity/Quote.php

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

> year

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

> string

Field length [255]:

> 5

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

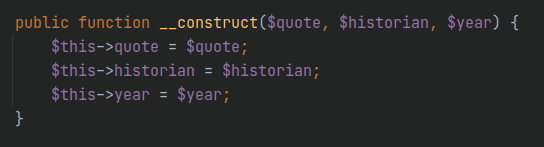
> no

updated: src/Entity/Quote.php

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

>

Сущность с именем Quote.php будет создан в каталоге src/Entity. Откройте src/Entity/Quote.php и добавьте функцию-конструктор, как показано ниже.



Создайте миграцию для обновления базы данных с помощью следующей команды.

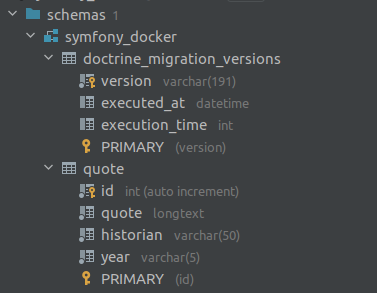


Запустите миграцию с помощью следующей команды.



Ответьте «да» на запрос, и ваша база данных будет обновлена ​​новой таблицей с именем «quote».

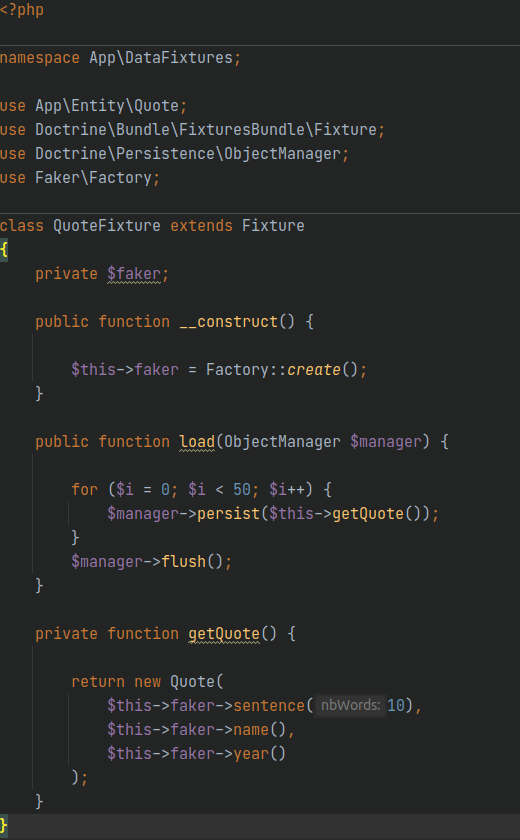
Далее, если вы используете PhpStorm, вы можете подключиться к бд и посмотреть наличие таблицы quote и её поля.



**10. Создайте фикстуру**

Затем нам нужно загрузить в базу данных набор сгенерированных цитат, чтобы, когда приложение будет готово, у нас были цитаты для извлечения и отображения. Для этого в контейнере PHP выполните следующую команду.

Фикстура будет расположена в src/DataFixtures/QuoteFixture.php. Откройте файл и обновите его, чтобы он соответствовал приведенному ниже коду.



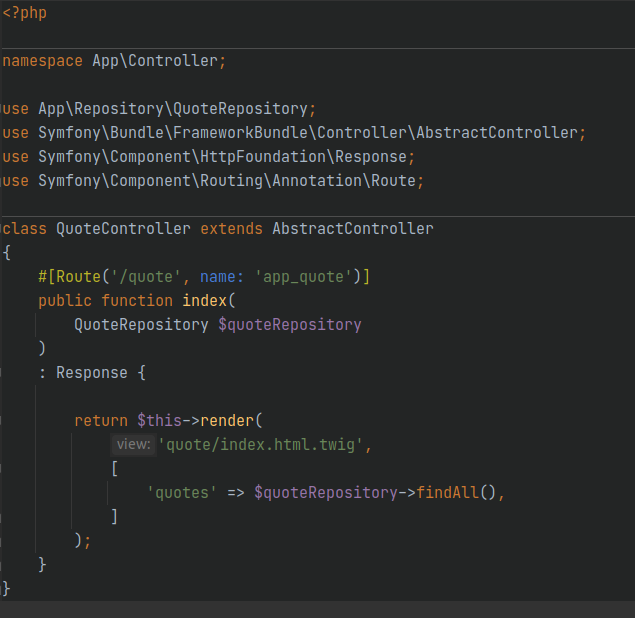
Чтобы загрузить фикстуры в нашу базу данных, выполните следующую команду в контейнере PHP.

**11. Создание контроллера цитат**

Затем нам нужно создать новый контроллер, который будет использовать объект ORM для извлечения котировок из базы данных и отображения их в представлении, которое мы можем видеть. Для этого в контейнере php выполните следующую команду.



Когда команда завершится, в каталоге src/Controller будет создан новый контроллер с именем QuoteController.php. Откройте файл и обновите его, чтобы он соответствовал приведенному ниже коду.



**12. Стилизовать представление**

Наконец, давайте сделаем представление профессиональным и удобным для чтения. Для этого мы стилизуем его с помощью Bootstrap. Обновите app/templates/base.html.twig, чтобы он соответствовал приведенному ниже коду.

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1">

<link

href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-beta3/dist/css/bootstrap.min.css"

rel="stylesheet"

integrity="sha384-eOJMYsd53ii+scO/bJGFsiCZc+5NDVN2yr8+0RDqr0Ql0h+rP48ckxlpbzKgwra6"

crossorigin="anonymous">

<title>{% block title %}Welcome!{% endblock %}</title>

{% block stylesheets %}

{% endblock %}

{% block javascripts %}

{% endblock %}

</head>

<body>

{% block body %}{% endblock %}

<script

src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-beta3/dist/js/bootstrap.bundle.min.js"

integrity="sha384-JEW9xMcG8R+pH31jmWH6WWP0WintQrMb4s7ZOdauHnUtxwoG2vI5DkLtS3qm9Ekf"

crossorigin="anonymous">

</script>

</body>

</html>

Затем откройте app/templates/quote/index.html.twig и отредактируйте его содержимое, чтобы оно соответствовало следующему.

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

{% block title %}Quotes{% endblock %}

{% block body %}

<style>

.wrapper {

margin: 1em auto;

width: 95%;

}

</style>

<div class="wrapper">

<h1>Great Quotes</h1>

<table class="table table-striped table-hover table-bordered">

<thead>

<tr>

<th scope="col">#</th>

<th scope="col">Quote</th>

<th scope="col">Historian</th>

<th scope="col">Year</th>

</tr>

</thead>

<tbody>

{% for quote in quotes %}

<tr>

<td>{{ loop.index }}</td>

<td>{{ quote.quote }}</td>

<td>{{ quote.historian }}</td>

<td>{{ quote.year }}</td>

</tr>

{% endfor %}

</tbody>

</table>

</div>

{% endblock %}

**13. Протестируйте приложение**

Теперь, когда приложение готово, пришло время его протестировать.Перейдите на http://localhost:8080/quote страницу, чтобы увидеть известные цитаты, хранящиеся в вашей базе данных.

