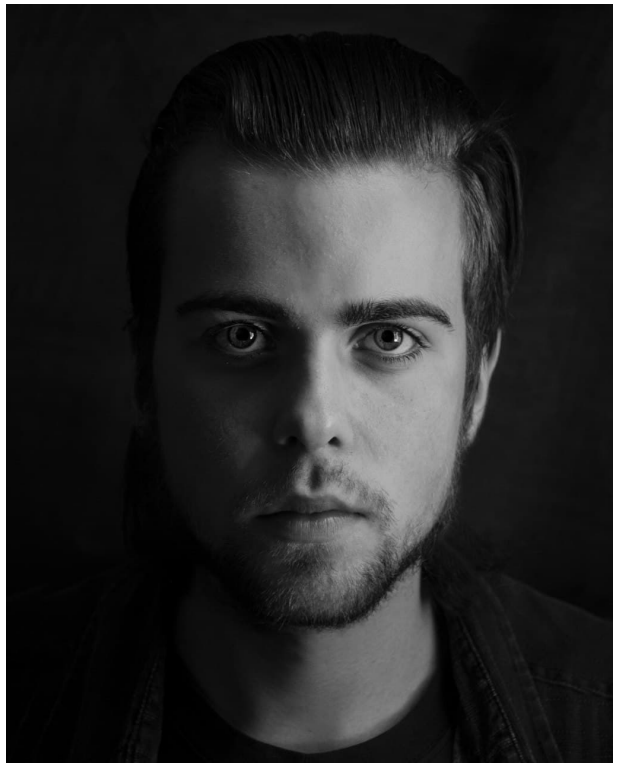
Веб разработка на PHP Symfony

Doctrine (репозитории, entity менеджер, фикстуры)

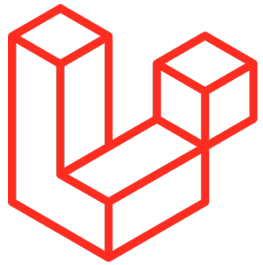
Веб разработка на PHP Symfony

Введение



Демьян Костельный
Middle PHP Developer

demian-kostelny-613b90151



Тема урока

Doctrine
(репозитории, entity менеджер, фикстуры)

План урока

1. Немного теории
2. Генерация Entity
3. Генерация миграций
4. Паттерн репозиторий
5. Сохранение и обновление Entity
6. Генерация фикстур
7. Генерация фикстур с помощью Alice

Немного теории

Немного теории

В Symfony framework уже есть все нужные инструменты для того чтобы полноценно работать с базами данных. Компонент Doctrine представляет собой целый набор инструментов, который полностью обеспечивает работу с такими типами БД как MySQL, PostreSQL, MongoDB и многие другие. Благодаря Doctrine можно генерировать все нужные компоненты для БД - прямо через консоль.

Для того чтобы установить Doctrine ORM, нужно воспользоваться Composer:

```
composer require symfony/orm-pack
```

Генерировать все нужные компоненты для того чтобы работать с БД мы будем через maker-bundle, который у нас уже установлен.

Немного теории

Перед тем как приступать к использованию Doctrine, нам нужно настроить файл конфигурации `.env` под нашу БД, который находится в корневой папке проекта.

```
29  # DATABASE_URL="sqlite:///%kernel.project_dir%/var/data.db"
30  DATABASE_URL="mysql://db_user:db_password@127.0.0.1:3306/db_name?serverVersion=5.7"
31  # DATABASE_URL="postgresql://db_user:db_password@127.0.0.1:5432/db_name?serverVersion=13&charset=utf8"
32  ###< doctrine/doctrine-bundle ###
```

db_user - логин пользователя в phpMyAdmin (в OpenServer это root)
db_password - пароль пользователя в phpMyAdmin (в OpenServer его просто нет, так что оставляем пустое место)
127.0.0.1:3306 - адрес сервера и порт сервера где находится БД для подключения (это оставляем как есть)
db_name - название БД для подключения (здесь нужно указать название БД которую мы создадим в phpMyAdmin)

Генерация Entity

Генерация Entity

Entity, либо сущности - это классы от Doctrine ORM, которые представляют собой полноценные объекты, содержащие в себе поля БД. К примеру, может быть Entity для постов блога, который в себе будет содержать функции полей названия поста, описания, имени автора и т.п.
Сейчас мы будем генерировать новые Enity через командную строку с помощью следующей команды:

```
php bin/console make:entity
```

После ввода данной команды у нас появится возможность полностью настроить то, какие поля должен иметь Entity и все их характеристики.

Генерация Entity

После того как мы сгенерировали новый Entity (который мы назвали Product), он появится у нас в папке src/Entity/Product.php. Открыв файл мы сможем рассмотреть названия полей в качестве параметров класса и функции, которые позволяют работать с полями в базе данных. Именно такие функции как, например, getId() и т.п. мы будем использовать в коде - для того, чтобы полноценно работать с таблицей в БД, которая будет привязана к данному Entity. Это то, по какой логике работает Doctrine. Есть Entity и в нём есть функции на все поля для таблицы в БД, которые как раз и используются для работы с данными.

Генерация миграций

Генерация миграция

У нас уже есть Entity с названием Product. Это полностью готовый класс для того, чтобы была готова уже таблица product и которая хранила бы в себе поля, прописанные именно в этом Entity. Следующим шагом, чтобы создать эту таблицу, является создание новой миграции. Сделать это можно с помощью maker-bundle:

```
php bin/console make:migration
```

После того, как будет запущена данная команда, создастся новый файл миграции для БД, который будет в себе содержать SQL запросы для обновления нашей БД. В свою очередь это будет хранить в себе и SQL-запросы для создания таблицы product. Чтобы этот файл запустился, нужно ввести команду с обращением к Doctrine для запуска миграций:

```
php bin/console doctrine:migrations:migrate
```

Генерация миграция

Конечно может быть ситуация когда нужно добавить новое поле к уже существующему Entity. Это можно сделать, добавив новый параметр класса и функции внутри Entity и создав новую миграцию через консоль.

```php
class Product
{

    // ...

    /**
     * @ORM\Column(type="text")
     */
    private $description;

    public function getDescription(): ?string
    {
        return $this->description;
    }

    // И также добавляем setDescription()
    // ...
}
```

Паттерн репозиторий

Паттерн репозиторий

Во многих фреймворках для работы с БД, используется такой паттерн проектирования как “Репозиторий”. Он представляет из себя набор отдельных классов, созданных для работы с самой моделью, не обращаясь к классу модели напрямую. Для каждой модели создаются отдельные файлы классов (репозитории) в папке src/Repository. Сами репозитории создаются уже автоматически после создания нового Entity. После этого в ProductController можно воспользоваться нашим репозиторием `src/Repository/ProductRepository.php`:

```php
// Сначала вызываем Doctrine, а после этого уже вызываем репозиторий внутри функции
// getRepository указав класс Entity - Product
$repository = $this->getDoctrine()->getRepository(Product::class);

// После этого мы   сможем работать с нашим Entity через репозиторий, не обращаясь
// напрямую к классу Entity
$product = $repository->find($id);
```

Паттерн репозиторий

Внутри репозитория Product можно добавлять собственные функции для создания запросов для работы с БД.

```php
/**
 * @return Product[]
 */
public function findAll(): array
{
    $entityManager = $this->getEntityManager(); // Достаём entity менеджер чтобы в дальнейшем создать  запрос

    $query = $entityManager->createQuery(
        'SELECT *
        FROM App\Entity\Product'
    ); // Создаём наш SQL запрос

    return $query->getResult(); // Получаем результат от запроса
}
```

Сохранение и обновление Entity

Сохранение и обновление Entity

Чтобы сохранить новый объект (или, проще говоря, запись в таблицу БД) - это можно сделать с помощью следующего способа:

```php
/**
 * @Route("/product")
 */
public function createProduct(): Response
{

    // Для того чтобы можно было добавлять новые записи в таблицу в БД
    // нам нужно вызвать entity менеджер в переменную $entityManager.
    // Также нужно добавить use Doctrine\ORM\EntityManagerInerface;
    $entityManager = $this->getDoctrine()->getManager();

    // Также вначале класса нужно и подключить наш Entity
    // use App\Entity\Product
    $product = new Product();
    $product->setName('Cool book');
    $product->setPrice(300);
    $product->setDescription('This is very cool and nice book.');

    // Чтобы сохранить новый объект в таблицу, мы вызываем
    // entity менеджер и функцию persist внутри которой
    // указываем новый объект от entity Product
    $entityManager->persist($product);

    return new Response('New product saved with id ' . $product->getId());

}
```

Сохранение и обновление Entity

Чтобы достать какой-то объект из базы данных, нужно воспользоваться репозиторием и указать id объекта, который мы ищем:

```php
/**
 * @Route("/product/{id}", name="product_show")
 */
public function findProduct($id): Response
{

    $product = $this->getDoctrine()
        ->getRepository(Product::class)
        ->find($id); // Будет находить  объект через его id указаном в url

    if (!$product) {
        throw $this->createNotFoundException(
            'No product found for id '.$id
        );
    } // Здесь будет выпадать ошибка если объект не будет найден

    return new Response('Product found: ' . $product->getName());

}
```

Сохранение и обновление Entity

Если мы хотим обновить объект, который уже в БД, то нужно будет сначала найти его id, после чего внести изменения и обновить:

```php
/**
 * @Route("/product/edit/{id}")
 */
public function updateProduct($id): Response
{

    $product = $this->getDoctrine()
        ->getRepository(Product::class)
        ->find($id);

    if (!$product) {
        throw $this->createNotFoundException(
            'No product found for id '.$id
        );
    }

    $product->setName('Super new name for product'); // Задаём новое имя для товара
    $entityManager->flush(); // Обновляем наш объект в БД

    return $this->redirectToRoute('product_show', [
        'id' => $product->getId()
    ]); // Делаем редирект на только что обновленный нами товар

}
```

Сохранение и обновление Entity

Удаление объекта из БД:

```
// Чтобы удалить объект в БД - достаточно указать внутри функции remove()
// переменную с объектом Entity который мы хотим удалить
$entityManager->remove($product);
$entityManager->flush() // И с помощью flush() делаем запрос в БД
```

Генерация фикстур

Генерация фикстур

Для того чтобы загружать в таблицу БД некоторые фейковые данные, можно воспользоваться фикстурами. Они представляют из себя объекты, которые хранят в себе самые разные данные, которые уже готовы для загрузки в таблицу в БД. Перед тем как воспользоваться возможностями фикстур, нужно установить их Bundle через Composer:

```
composer require orm-fixtures --dev
```

После установки данного Bundle можно будет создавать новые фикстуры с помощью команды:

```
php bin/console make:fixtures
```

Генерация фикстур

После запуска команды генерации новой фикстуры, единственное, что требуется ввести - это название фикстуры, после заполнения будет выведено сообщение об успешно созданной фикстуре:

```
The class name of the fixtures to create (e.g. AppFixtures):
> ProductFixtures

created: src/DataFixtures/ProductFixtures.php

 Success!
```

Генерация фикстур

После запуска команды генерации новой фикстуры, единственное, что требуется ввести - это название фикстуры, после заполнения будет выведено сообщение об успешно созданной фикстуре:

```php
namespace App\DataFixtures;

use Doctrine\Bundle\FixturesBundle\Fixture;
use Doctrine\Persistence\ObjectManager;

class ProductFixtures extends Fixture
{
    public function load(ObjectManager $manager)
    {
        $product = new Product();
        $product->setName('testProduct');
        $product->setPrice(200);
        $manager->persist($product);

        // После этого будет создан новый объект в БД с параметрами
        // которые мы указали выше

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

Генерация фикстур

Чтобы загрузить созданные нами фикстуры в БД - достаточно воспользоваться следующей командой:

```
php bin/console doctrine:fixtures:load
```

Генерация фикстур с помощью Alice

Генерация фикстур с помощью Alice

Для того чтобы было проще генерировать фикстуры - существует отдельный Bundle на GitHub, который называется Alice. Сейчас мы его рассмотрим.

https://github.com/hautelook/AliceBundle

README.md

AliceBundle

A Symfony bundle to manage fixtures with nelmio/alice and fzaninotto/Faker.

The database support is done in FidryAliceDataFixtures. Check this project to know which database/ORM is supported.

Warning: this is the documentation for HautelookAliceBundle 2.0. If you want to check the documentation for 1.x, head this way.

Проверка знаний

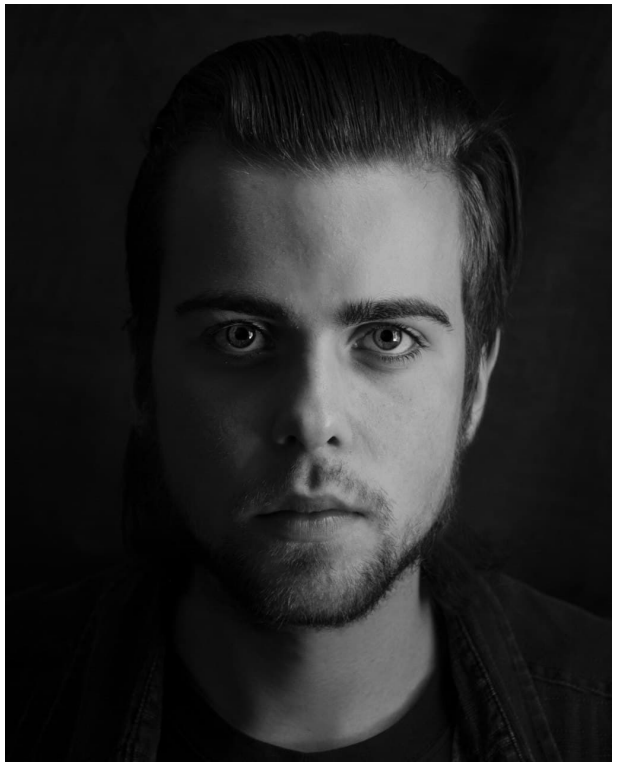
TestProvider.com

TestProvider – это online сервис проверки знаний по информационным технологиям. С его помощью Вы можете оценить Ваш уровень и выявить слабые места. Он будет полезен как в процессе изучения технологии, так и для общей оценки знаний IT специалиста.

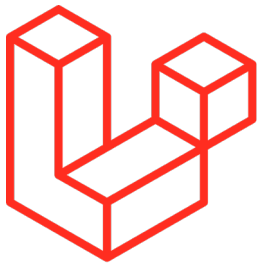
Проверьте как Вы усвоили данный материал на TestProvider.com

Успешное прохождение финального тестирования позволит Вам получить соответствующий Сертификат.

Спасибо за внимание! До новых встреч!



Демьян Костельный
Middle PHP Developer



Информационный видеосервис для разработчиков программного обеспечения

TestProvider Pearson CyberBionic systematics Microsoft Partner Windows Azure EBA