

## БЫЛО ВОТ ТАКОЕ ЗАДАНИЕ:

### Задачи по symfony:

#### #1. Реализовать две сущности: Категория, Товар с следующими полями:

Category:

- id (int)
- title (string length min: 3, max 12)
- eld (int|null)

Product:

- id (int)
- categories (связь: ManyToMany)
- title (string length min: 3, max 12)
- price (float range min: 0, max 200)
- eld (int|null)

*\*eld – произвольный id из любой другой системы*

#### #2. Реализовать CRUD контроллер и сервис для сущности Product

Логика добавления, удаления должна быть вынесена в сервис, данные методы также использовать в задаче #3.

#### #3. Реализовать консольную команду, которая читает два нижеприведенных файла Json и добавляет/обновляет записи в БД:

categories.json:

```
[
  {"eld": 1, "title": "Category 1"},
  {"eld": 2, "title": "Category 2"},
  {"eld": 2, "title": "Category 33333333"}
]
```

products.json

```
[
  {"eld": 1, "title": "Product 1", "price": 101.01, "categoriesEld": [1,2]},
  {"eld": 2, "title": "Product 2", "price": 199.01, "categoryEld": [2,3]},
  {"eld": 3, "title": "Product 33333333", "price": 999.01, "categoryEld": [3,1]}
]
```

*\*учесть валидацию данных*

#### #4. Реализовать подписчик/слушатель на добавление/изменение product.

Подписчик или слушатель при обновлении/добавлении данных о продукте должен отправлять сообщение на email указанный в параметре (произвольное название) в .env файл

---

ЧТО Я СДЕЛАЛ (коротко) <https://adv.peter.keenetic.pro:8000/>

1. Я сделал две сущности по первой задаче – **Category** и **Product**, связанные Many-to-Many по id
2. Мигрировал в базы, сделал несколько экземпляров **Categories** и **Products**
3. Сделал CRUD – контроллеры ("/category", "/product")
4. Сделал сервис **ProductManager**. Оркеструет работу, отделяет сущность от контроллера.
5. Разработал консольную команду `App\Command\UpdateProductCommand`  
>> `php bin/console app:product-updater {1|2}` аргумент 1 (импорт категорий) или 2 (импорт товаров).  
Файлы находятся импорта «**categories.json**» и «**products.json**» здесь: **App\Service**
6. Подключил к БД **DatabaseSubscriber**, который вызывает `sendMail()` контроллера **PosterController**.
7. Ссылка на Github: [Peter-Korobeynikov/symfony](https://github.com/Peter-Korobeynikov/symfony)

## ПОЯСНЕНИЯ ПО ЗАДАЧАМ

### • **Задача 1. Создание сущностей**

Созданы сущности Category и Product. (php bin/console make:entity) Связи настроены нотацией Doctrine. Требовалось создать связь Many-to-Many, но по условиям задачи в объекте Category не присутствует коллекция связей с объектами Product, поэтому связь не симметричная, а в БД добавлена таблица product\_categories. Миграция - `php bin/console doctrine:migrations:diff && bin/console doctrine:migrations:migrate`

The screenshot shows the phpMyAdmin interface. On the left, the database structure is listed, with 'symfony\_db' selected. Under 'symfony\_db', the 'products\_categories' table is highlighted. The main panel displays the 'product' table with the following data:

id	title	price	e_id
1	Product-1	1	111
3	Product 1	101.01	1
5	Product 2	66.5	2
6	Product 3	18.34	4
8	Product-8	90	55
9	Product-34	34	34

Below the 'product' table, the 'products\_categories' table is shown with the following data:

product_id	category_id
1	5
1	7
3	11
3	24
5	24
5	25
6	24
6	25
8	17
9	14

### • **Задача 2. Создание контроллеров CRUD. Вынос обработки в сервис**

С контроллерами ничего сложного - `bin/console make:crud`

Затем я создал класс **ProductManager**. Это некоторая оболочка, которая имплементирует EntityManager, вживленный через сеттер (см. трейт **TEntityManager**) и инстанцирует (присоединяет) entity Category и Product для манипуляций с их жизнью и здоровьем (update-remove). Также этот сервис реализует оболочку для импорта и погружения в базу из файлов: public function **import**(string \$className, string \$fileName). И, наконец, чтобы наш сервис мог работать с сущностями разных классов, к наши сущности реализуют интерфейс **EntityIntegrityInterface** (обеспечивает сохранение целостности объектов сущностей):

```
public function json_serialize(): string;
public function json_deserialize(EntityManagerInterface $em, $json_data): object;
public function onUpdate(EntityManagerInterface $em, array $context = []): bool;
public function onRemove(EntityManagerInterface $em): bool;
```

(\*) Благодаря методам onRemove и onUpdate могут дополнительно валидировать себя, а также ликвидировать невалидные ссылки при удалении (Category) для связанных объектов:

```
public function onRemove(EntityManagerInterface $em): bool {
    // При удалении категории - удалим её из коллекций продуктов
    $repository = $em->getRepository(Product::class);
    $products = $repository->findAll();
    foreach ($products as $product) $product->removeCategory($this);
    return true;
}
```

- **Задача 3. Создание контроллеров CRUD. Вынос обработки в сервис**

Создана служба (команда) **UpdateProductCommand**.

Тут как бы и говорить особо не о чем. Команда читает файлы **categories.json** и **products.json** и выполняет импорт записей в объекты сущностей с погружением в БД. Для этого вызывается функция **import** нашего любимого инкапсулятора **ProductManager** (из задачи 2). Который, в свою очередь, даёт возможность сущностям сериализовать самих себя, что, в общем-то по логике и должно быть.

Аргумент один – **file\_type** является обязательным, он задает тип импортируемого файла 1 – категории, 2 – продукты (товары).

**App\Command\UpdateProductCommand:**

```
arguments:
- "%app.serialize_format%"
tags:
- { name: console.command, command: app:product-updater }
```

```
class UpdateProductCommand extends Command
{
    use TProductManager;

    private $file_type = 1;
    protected static $defaultName = 'app:product-updater';

    // -----
    public function __construct()
    {
        parent::__construct();
    }

    // -----
    protected function configure(): void
    {
        $this
            ->setDescription( description: 'Updates categories and products from files: categories.json and products.json')
            ->setHelp( @help 'Updates categories and products from files: "categories.json" and "products.json".
- categories.json file format: [ {"eId": 1, "title": "Category 1"}, ... ],
- products.json file format: [ {"eId": 1, "title": "Product 1", "price": 101.01, "categoriesEId": [1,2] }, ... ]')
            ->addArgument( name: 'file_type', mode: InputArgument::REQUIRED, description: '1 - categories, 2 - products')
        ;
    }

    // -----
    protected function execute(InputInterface $input, OutputInterface $output): int
    {
        $output->writeln(['Products & Categories import','-----','Do not panic. A lot to do...'],);
        $this->file_type = $input->getArgument( name: 'file_type');
        $output->write(['File = ', $this->file_type == 1 ? 'categories' : 'products', "\n"]);
        $path = $path = __DIR__ . DIRECTORY_SEPARATOR . '..' . DIRECTORY_SEPARATOR . 'Service' . DIRECTORY_SEPARATOR;
        $output->write(['Path = ', __DIR__, "\n"]);
        switch ($this->file_type) {
            case 1:
                $output->write( messages: "Importing categories ... ");
                $filePath1 = $path . 'categories.json';
                $this->_pm->import( className: Category::class, $filePath1);
                break;
            case 2:
                $output->write( messages: "Importing products ... ");
                $filePath2 = $path . 'products.json';
                $this->_pm->import( className: Product::class, $filePath2);
                break;
        }
        $output->writeln( messages: "OK");
        return 0;
    }
}
```

Работу команды сделал «тихой» - записи не соответствующие ограничениям просто игнорируются и не попадают в БД. По идее нужно сделать еще аргументы команды, например, останавливать при ошибке или логировать ошибки в отдельный файл. Но это уже нам не задавали))

Формат вызова команды **bin/console app:product-updater {1|2}**

- **Задача 4. Создание «прослушки» на изменение**

Повесил простого подписчика на мониторинг базы.

```
class DatabaseSubscriber implements EventSubscriber
{
    public function getSubscribedEvents() {
        return [Events::postUpdate, Events::postPersist, Events::postRemove];
    }

    private $_poster;
    public function __construct(PosterController $poster) { $this->_poster = $poster; }
    public function postPersist(LifecycleEventArgs $args) { $this->onEvent( act: 'persist', $args); }
    public function postRemove(LifecycleEventArgs $args) { $this->onEvent( act: 'remove', $args); }
    public function postUpdate(LifecycleEventArgs $args) { $this->onEvent( act: 'update', $args); }

    private function onEvent(string $act, LifecycleEventArgs $args) {
        $entity = $args->getEntity();
        $manager = $args->getEntityManager();
        if (!$entity instanceof Product) return;
        $entity_str = $entity->json_serialize();
        switch ($act) {
            case 'persist': $this->_poster->sendEmail($manager, $entity, act: 'persist'); break;
            case 'update': $this->_poster->sendEmail($manager, $entity, act: 'update'); break;
            case 'remove': $this->_poster->sendEmail($manager, $entity, act: 'remove'); break;
        }
    }
}
```

Использовал класс **PHPMailer**. «Штатный» мэйлер начал выбрасывать исключения по SSL, да и громоздкий он показался

```
class PosterController extends AbstractController
{
    use TSingleton;

    private $_mail = null;
    protected function init() {
        $this->_mail = new PHPMailer;
        $this->_mail->isSMTP(); // telling the class to use SMTP
        $this->_mail->SMTPAuth = true; // enable SMTP authentication
        $this->_mail->Port = 465; // sets Port of SMTP server
        $this->_mail->Host = "smtp.yandex.ru"; // sets the SMTP server
        $this->_mail->Username = "korobeynikov.pv@yandex.ru"; // SMTP account username
        $this->_mail->Password = "Esmart-1734"; // SMTP account password
        $this->_mail->Priority = 3;
        $this->_mail->Encoding = '8bit';
        $this->_mail->ErrorInfo = '';
        $this->_mail->CharSet = 'UTF-8';
        $this->_mail->ContentType = 'text/html';
        $this->_mail->SMTPSecure = 'SSL';
    }

    public function __construct() { $this->init(); }
    public function getMail() { assert(isset($this->_mail)); return $this->_mail; }

    /**
     * @Route("/email", name="poster_new", methods={"GET","POST"})
     */
    public function sendEmail($manager, $entity, $act): Response {
        $mail = $this->getMail();
        $mail->isSendmail(); // Set PHPMailer to use the sendmail transport
        $mail->setFrom($_SERVER['MAIL_FROM'], 'Peter K. '); //Set who the message is to be sent from
        $mail->addReplyTo($_SERVER['MAIL_REPLY'], 'Peter K. '); //Set an alternative reply-to address
        $mail->addAddress($_SERVER['MAIL_TO'], 'John Doe'); //Set who the message is to be sent to

        // Само письмо ...
        $mail->Subject = 'PHPMailer sendmail test'; //Set the subject line
        $mail->msgHTML('<p>See Twig integration for better HTML integration!</p>', __DIR__);
        // $mail->msgHTML($this->renderView('poster/new.html.twig', ['param' => 'Привет!!!']), 'text/html');
        $mail->AltBody = 'This is a plain-text message body'; //Replace the plain text body with one created manually
        $mail->addAttachment('images/logo.png'); //Attach an image file
        //send the message, check for errors
        if (!$mail->send()) {
            echo 'Mailer Error: ' . $mail->ErrorInfo;
        } else {
            echo 'Message sent!';
        }

        return Response::create();
    }
}
```

Единственно, что бы я хотел доработать – рассинхронизировать подписчика, а то он нагружает обработчик контроллера, а для нас здесь синхронизация не нужна, в отличие от методов «onUpdate» и «onRemove».

Вот ещё общая картинка:

The image shows a development environment with two main windows. The left window is PhpStorm, displaying the code for a Symfony project. The right window is a web browser showing a category page.

**PhpStorm Window:**

- Project: symfony
- File: ProductManager.php
- Code editor shows the `ProductManager` class with methods like `find`, `serialize`, `deserialize`, `import`, `update`, and `remove`.
- Debugger window is open at the bottom, showing the current state of the application.

**Web Browser Window:**

- Address bar: `adv.peter.keenetic.pro/8000/category/`
- Page title: Категории
- Table of categories:

Id	Title	Eid	Content	Действия
1	Test-entity	333	Контент1	Смотреть Редактировать
5	Заголовок	777	Контент	Смотреть Редактировать
7	Test-entity	4535	Контент	Смотреть Редактировать
10	qqq	444	Магнетик	Смотреть Редактировать
11	Category 1	56	test-reg-type	Смотреть Редактировать

Navigation buttons: Previous, Next, Create (Создать).

**Git Window:**

- Repository: symfony
- Working Tree/index (30 changed, 17 staged)
- Changes of workspace.xml (Modified) - Index vs. Working Tree (EOL Units - expected Windows)

## ПОЯСНЕНИЯ

1. Мой локальный адрес находится за роутером Keenetic и через их DNS у меня зарегистрировано доменное имя **adv.peter.keenetic.pro**  
Для тестовой задачи порт сервера – **8000**

Поэтому ВХОД на мой локальный сервер: <https://adv.peter.keenetic.pro:8000/>  
(нужно сделать исключение по безопасности для этого сайта)

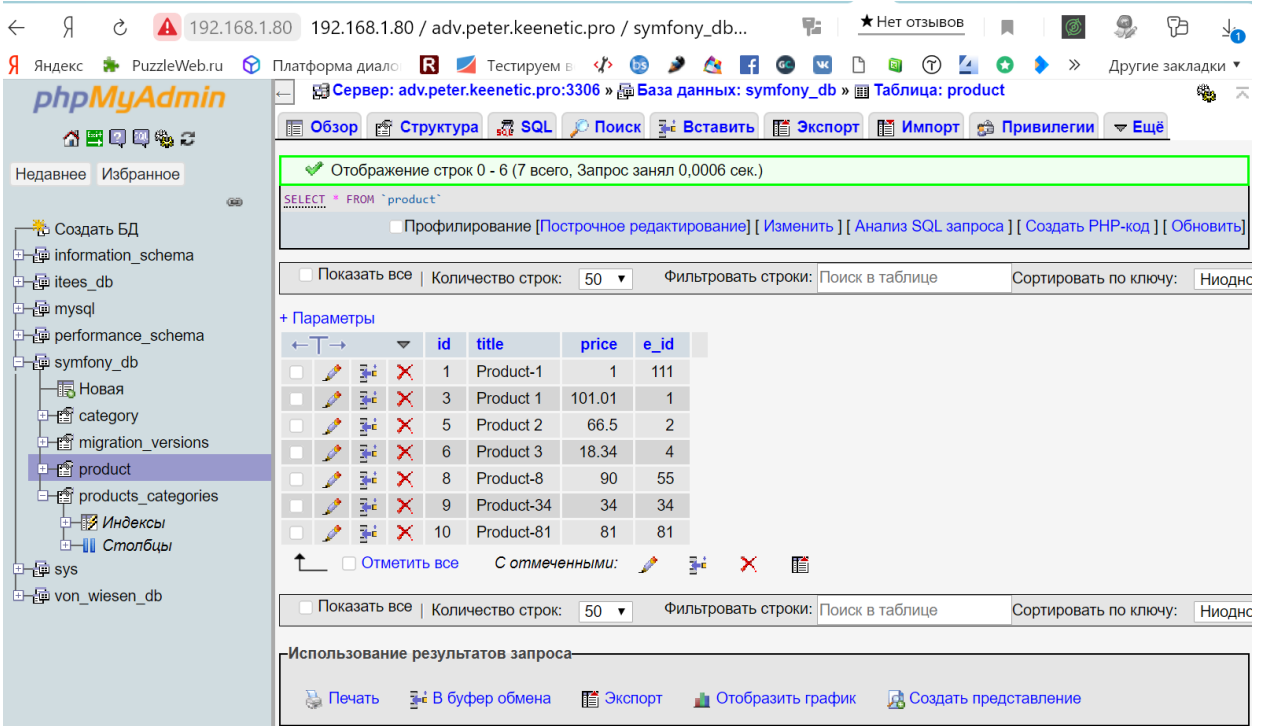
2. Доступ к базе (указан локальный IP)

**DATABASE\_URL=mysql://mastervw\_mysql:Esmart~!2@192.168.1.80:3306/symfony\_db?serverVersion=5.7**

3. ВХОД к базе и таблице:

[http://adv.peter.keenetic.pro/openserver/phpmyadmin/sql.php?server=1&db=symfony\\_db&table=product&pos=0](http://adv.peter.keenetic.pro/openserver/phpmyadmin/sql.php?server=1&db=symfony_db&table=product&pos=0)

**Сервер** adv.peter.keenetic.pro  
**Пользователь** mastervw\_mysql  
**Пароль** Esmart~!2



The screenshot shows the phpMyAdmin web interface. The left sidebar displays a tree of databases, with 'symfony\_db' selected. The main panel shows the 'product' table structure and data. The table has columns: id, title, price, e\_id. The data is as follows:

id	title	price	e_id
1	Product-1	1	111
3	Product 1	101.01	1
5	Product 2	66.5	2
6	Product 3	18.34	4
8	Product-8	90	55
9	Product-34	34	34
10	Product-81	81	81

4. Ссылка на **Github**:

[Peter-Korobeynikov/symfony](#)