

Разработка абстрактных компонентов и *Symfony* бандлов

Павел Степанец
21 июня 2019



АЛГОРИТМ РАБОТЫ

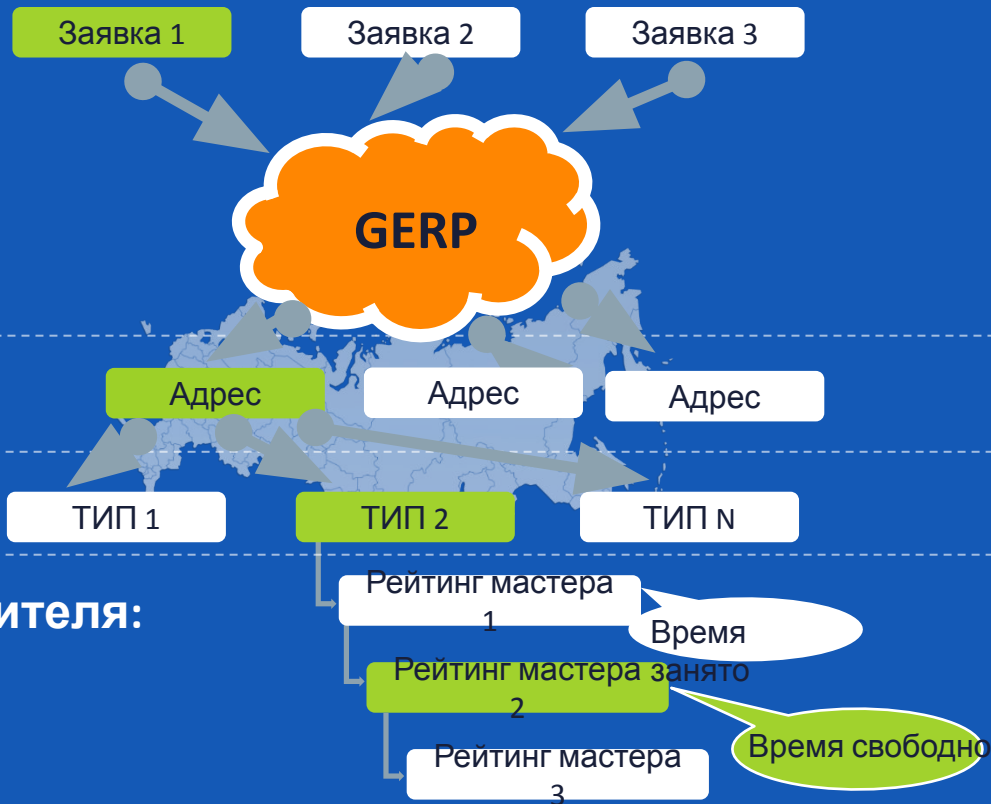
Входящие данные по заявке:

1. Контакты жителя
2. Тип проблем/аварий
3. Удобное время выполнения

Адрес с Тип проблем/аварий

Настройка служб ЖКХ/исполнителя:

1. Задается карта (зона) работы
2. Аттестация на типы проблем/аварий
3. Настройка рейтинга мастера, уровня качества
4. Задается график работы



Технологии

- PHP 7.2
- MySQL 5.6
- Symfony 3.4
- Memcache
- Docker
- Gitlab CI/CD



Symfony



docker



Умный город


Население
(Потребитель)



Оператор связи
37 городов

+ 3 пилота



Gorserv
UBER в ЖКХ
10 городов

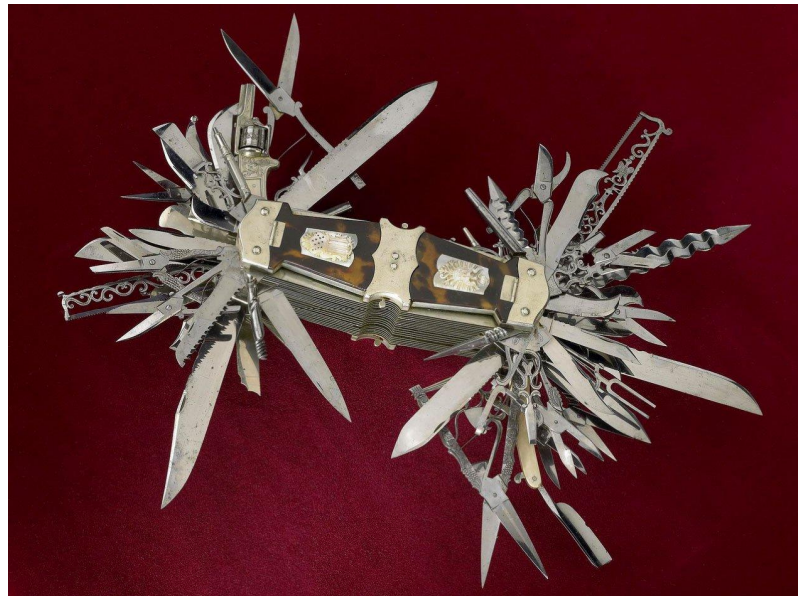
ПЛАТФОРМА GERP



МосЭнергоСбыт
Москва

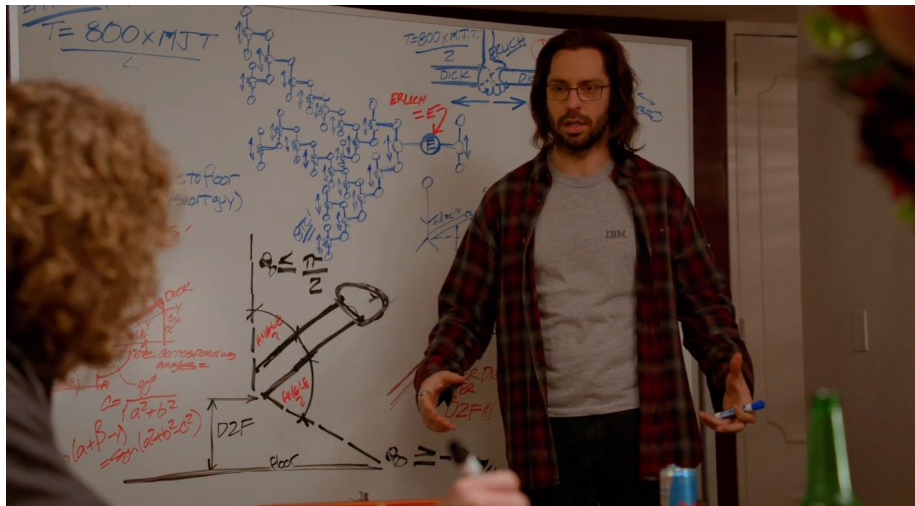
Проблема — несколько приложений в одном

- Конфликты функционала
- Раздутые конфигурации
- Деплой
- Производительность
- Стабильность



Решение — разделить проекты

- Избежать дублирования кода
- Анализ предметной области
- Определить контексты
- Разработать библиотеки и симфони-бандлы



Анализ предметной области

Task — заявка, задача.

Процессы:

- создание/изменение
- изменение состояния
- назначение исполнителю
- формирование графика
- уведомления
- отчетность

Связанные сущности:

- клиенты
- адрес
- каталог
- инвентарь

Нарезаем контексты

Заявки, операции,
состояние

Исполнители,
расписание, график

Клиенты,
реквизиты,
контакты

Уведомления

Адреса

Склад,
инвентарь

Контроль
качества

Приступаем к разработке

SOLID

Single responsibility

```
interface SchedulerManager
{
    public function scheduleSubject(
        SlotAware $subject,
        Slot $slot,
        Timeframe $timeframe): void;

    public function unscheduleSubject(
        SlotAware $subject): void;
}
```

Single responsibility

```
interface TimetableManager
{
    public function openTimeframe(Timeframe $timeframe): Timeframe;

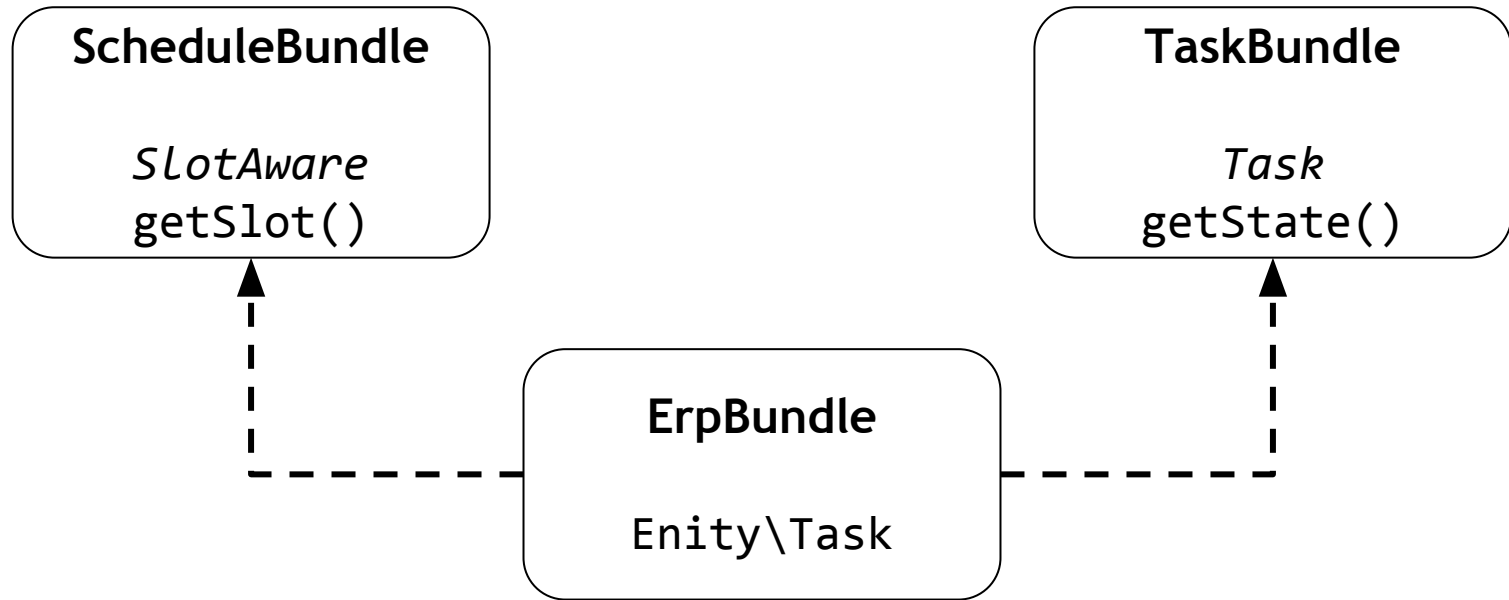
    public function updateTimeframe(Timeframe $timeframe): void;

    public function closeTimeframe(Timeframe $timeframe): void;
}
```

Interface segregation

```
class Task
{
    public function getSlot(): ?Slot {
        return $this->slot;
    }
    public function getState(): array {
        return $this->state;
    }
}
```

Interface segregation



Open-Closed, Dependency inversion

```
interface PointProvider
{
    public function find(string $address, array $options = []): ?Point;

    public function findAll(string $address,
        array $options = []): array;
}
```

Open-Closed, Dependency inversion

```
abstract class AbstractPointProvider implements PointProvider
{
    public function __construct(PointBuilder $pointBuilder,
        PointFinder $pointFinder)
    {
        $this->pointBuilder = $pointBuilder;
        $this->pointFinder = $pointFinder;
    }
}
```

Зависимости от других библиотек

```
class TacticianTaskManager implements TaskManager
{
    public function __construct(CommandBus $bus)
    {
        $this->bus = $bus;
    }
}
```

Liskov substitution

Пусть $q(x)$ является свойством верным относительно объектов x некоторого типа T . Тогда $q(y)$ также должно быть верным для объектов y типа S , где S является подтипом типа T .

Функции, которые используют базовый тип, должны иметь возможность использовать подтипы базового типа, не зная об этом



Итоги

- Разбиваем предметную область на независимые контексты.
- Абстрагируем контексты с помощью интерфейсов.
- Делаем интерфейсы максимально “тонкими”.
- Реализации делаем зависимыми от интерфейсов.
- Не придумываем функционал, который сейчас не нужен (KISS).
- Зависимости от сторонних библиотек допустимы, если возможно — зависьте от PSR.

Symfony Bundles

- EventDispatcher
- DependencyInjection
- Doctrine
- Forms
- Twig

EventDispatcher

- паттерн Observer

client

```
$disp->dispatch(`event_name`, new SomeEvent());
```

subscriber

```
$disp->subscribe(`event_name`, $handler);
```

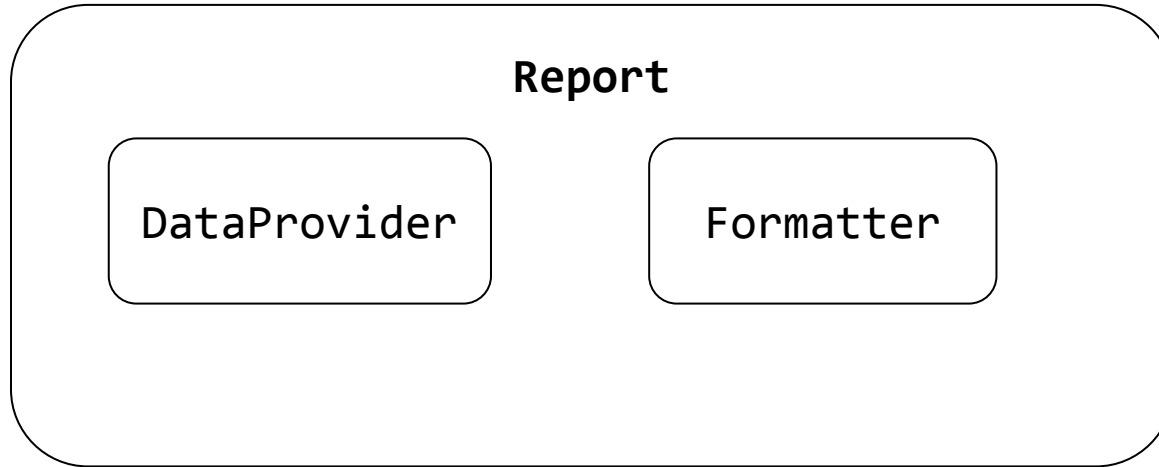
AbstractScriptRunner

```
public function start(Script $script, ScriptContext $context): ScriptResult {  
    $scriptResult = $this->createScriptResult($script, $context);  
    /* more code here */  
    $this->eventDispatcher->dispatch(  
        ScriptEvents::SCRIPT_STARTED,  
        new ScriptEvent($context, $scriptResult)  
    );  
    return $scriptResult;  
}
```

ScriptSubscriber

```
public function onScriptStarted(ScriptEvent $event): void
{
    $scriptResult = $event->getScriptResult(); $subject = ...;
    $comment = new CommentCommand($subject, [
        'tags' => ['Начался опрос'],
        'text' => 'Опрос: ' . $scriptResult->getScript()->getTitle(),
    ]);
    $this->taskManager->addComment($comment);
}
```

DI component, CompilerPass



DI component, CompilerPass

```
reportDef = findDef(Report::class)
provider = findByTag(provider_tag)
formatter = findByTag(formatter_tag)
reportDef.addConstructorArg(provider)
reportDef.addConstructorArg(formatter)
```

SmartDataProvider

provider_tag

HtmlFormatter

formatter_tag

Doctrine

```
abstract class Task implements TaskInterface
{
    // properties
}
```

Doctrine

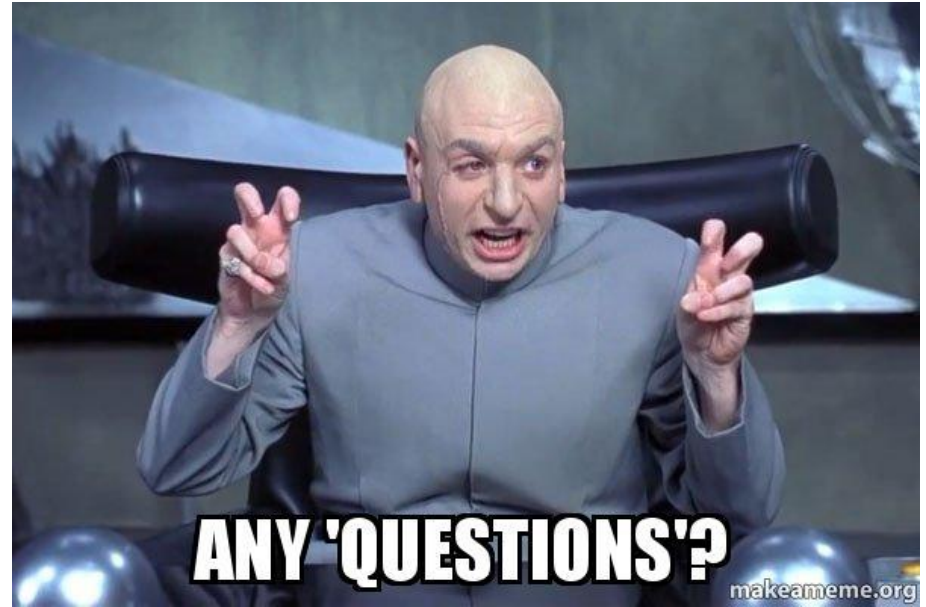
```
<?xml version="1.0" encoding="UTF-8"?>
<doctrine-mapping
  xmlns="http://doctrine-project.org/schemas/orm/doctrine-mapping">
  <mapped-superclass name="SymfonyBro\TaskBundle\Entity\Task">
    <field name="state" column="state" type="json" />
    <field name="createdAt" column="created_at" type="datetime" />
    <field name="updatedAt" column="updated_at" type="datetime" />
  </mapped-superclass>
</doctrine-mapping>
```

Forms, Twig

- FormTypes
- Базовые шаблоны отображения: списки, детализация, формы
- Расширения: фильтры, функции

Важно — формы, шаблоны и расширения должны зависеть только от тех абстракций, которые определены в бандле.

Спасибо за внимание!



@pablo_step
pablo@symfony-bro.com