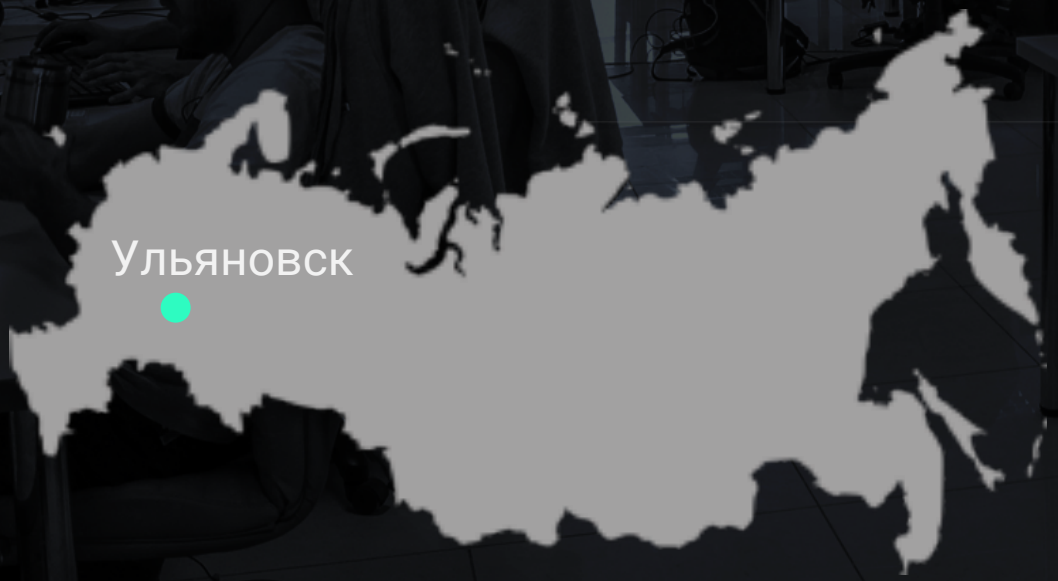
MEDIASOFT
PHP Software Engineering

СИМФОНИЧЕСКИЙ / ДРУГОЙ ПОДХОД К РАЗРАБОТКЕ НА 1С-БИТРИКС

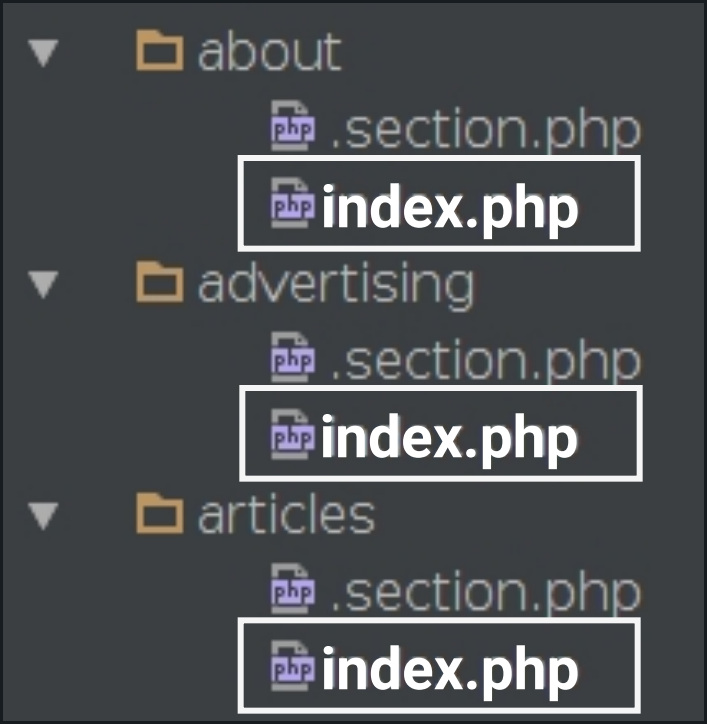


Андрей Клюев
Веб-разработчик

Что нас не устроило в стандартном подходе?

- Постоянно приходится следить за подключенными модулями
- Жёсткие и неудобные требования к именованию классов и файлов
- Нет единой точки входа в приложение
- Нет чёткой границы между подготовкой данных и их рендерингом
- Инфоблоки это медленно
- Highload-инфоблоки - внутри тот же ORM, но добавляют свои ограничения
- Для ORM нужно писать админку

Типовая структура файлов проекта на 1С-Битрикс



- Очень много однотипных файлов
- В комплексных компонентах приходится описывать логику обработки URL
- В целом исполняется слишком много лишнего кода
- Нет чёткого MVC

+ Все привыкли к такой структуре
+ Можно создавать страницы через WEB-интерфейс

Реализация MVC в 1С-Битрикс

MODEL - модули и их ORM сущности

VIEW - шаблоны компонентов,
(с ними всё уже не так однозначно)

CONTROLLER - отсутствует совсем,
(едва ли компоненты можно назвать контроллерами)

Composer - простое решение

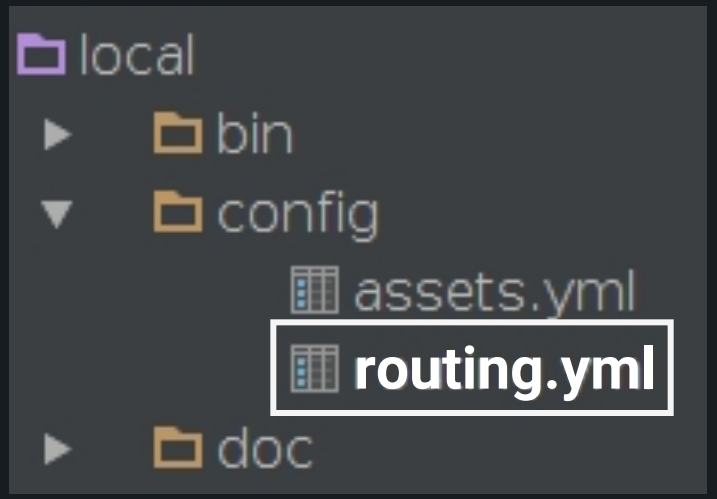
+ Подключается одной строкой в init.php

+ Можно не плодить модули

+ Весь код в одном месте

+ PSR-4 не заставляет вас именовать файлы в нижнем регистре

Мы организовали другую структуру – “честный MVC”



Файл index.php стал единой точкой входа в приложение
в нём не подключается ни хедера, ни футера, лишь роутер, он отправляет запрос к нужному контроллеру

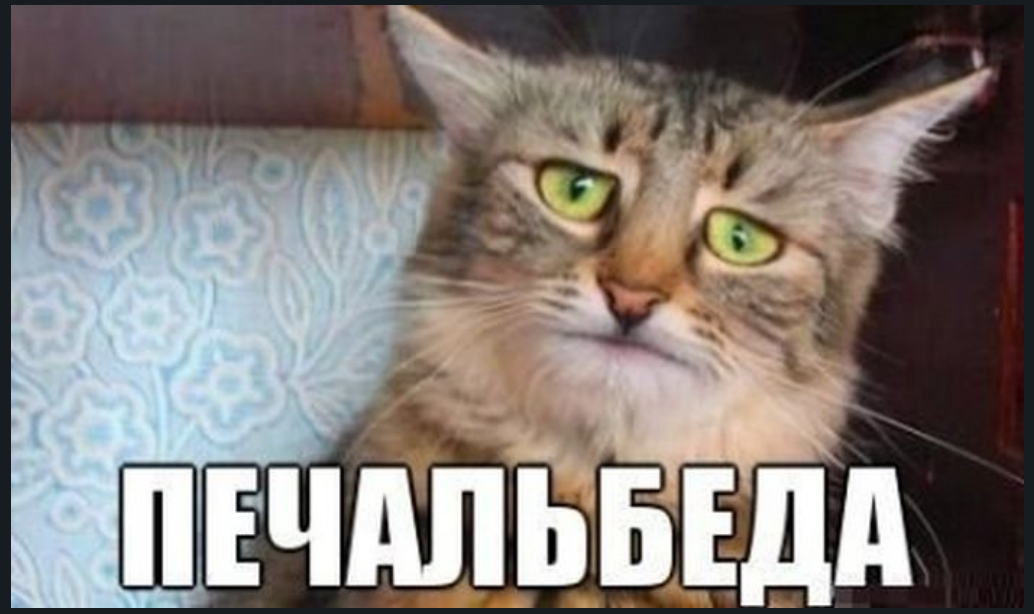
+ Настоящие MVC контроллеры
+ Легко следить за URL-ами
+ Всё логика маршрутизации описана единожды и инкапсулирована внутри роутера

- Не привычно для Битрикс-разработчиков
- Нет возможности создавать страницы через WEB-интерфейс

Роутинг

```yaml
routes:
    homepage:
        pattern: index.php
        methods:
            - GET
        action:
            - \OurGreatProject\Controller\HomepageController
            - indexAction

    news_list:
        pattern: news/
        methods:
            - GET
        action:
            - \OurGreatProject\Controller\NewsController
            - listAction
```

К сожалению, 1С-Битрикс так не работает =(
Как теперь рендерить HTML?



У каждого роута в конфигурации добавился путь до View

```yaml
news_list:
    pattern: news/
    methods:
        - GET
    action:
        - \OurGreatProject\Controller\NewsController
        - listAction
    view: news/list.php

news_detail:
    pattern: 'news/#code#/'
    methods:
        - GET
    action:
        - \OurGreatProject\Controller\NewsController
        - getAction
    view: news/detail.php
```

View - теперь тоже стали честными

+ Шаблоны чётко структурированы

+ Для XHR запросов view просто пропускается и в ответ клиенту отправляется JSON с результатом работы контроллера

- По-прежнему нельзя создавать страницы через админку

Честное MVC

MODEL - по прежнему ORM
VIEW - файлы шаблонов, которые указаны в роутинге
CONTROLLER - методы, которые указаны в роутинге

+ Слой приложения. Чётко разделены модели и вьюхи, ничего не знают друг о друге
+ MVC - позволило нам писать лёгкие контроллеры вместо громоздких компонентов
+ Оставшиеся компоненты стали использоваться как виджеты
+ А так же позволило быстро создать легковесное и расширяемое JSON REST api

Костыли и велосипеды реализации

Мы все же решили вернуть возможность создавать страницы через админку

header.php - собираем в буфер весь выводимый контент стандартных страниц

footer.php - вызывается отрисовка вьюхи с собранным контентом

Symfony-way

“Битрикс-битриксом, а делать хочется красивее!” (с)
Именно после этой фразы у нас появился Twig

+ Наследование шаблонов (честная система лейаутов)
+ Легко делить большие шаблоны на мелкие включаемые части
+ Опрятный синтаксис
+ Twig C Extension для ускорения рендеринга шаблонов на сервере

Show must go on!

И мы пошли дальше

Первым делом мы полностью отказались от передачи HTML по xhr

Мы подключили Twig.js и начали рендерить HTML на клиенте

Twig.js

+ Одни и те же шаблоны рендерятся как на сервере так и на клиенте

- Twig.js не поддерживает авто-экранирование
- Нельзя писать сложную логику в расширениях
- Шаблоны надо как-то передавать клиенту

Twig.js

Шаблоны надо как-то передавать клиенту,
плюс количество скриптов росло не по дням, а по часам

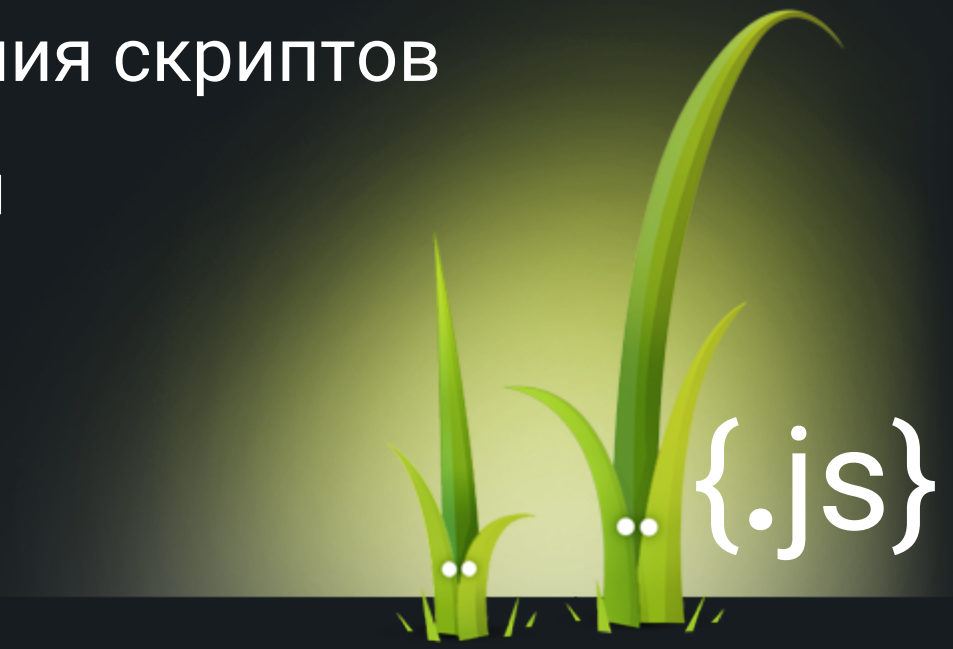
- Assetic?
- Gulp?
- Grunt?

Минусы

- Избыточная сложность
- Нужен node.js
- Нужно писать длинные конфиги

Так родился наш простейший AssetManager

+ Конфиг - банальный список файлов стилей, скриптов, шаблонов
+ Хорошо виден порядок подключения скриптов
+ В production окружении все ассеты минифицируются
+ Шаблоны также минифицируются



Теперь можно и бизнес-логику писать...

Инфоблоки

+ Всё работает “из коробки”
+ Готовая админка
+ Легко модифицировать “сущности”
+ Легко выводить разные данные вперемешку

- Работает медленно
- Страшный API
- Данные свалены в кучу
- Инфоблок могут случайно поменять
- Необходимо следить за тем, чтобы на разных площадках инфоблоки были идентичны

BitrixORM

+ Работает быстро
+ Приятный API
+ Сущности разделены
+ Легко разнести по разным серверам

- Нет готовой админки

Особенности реализации админки на 1С-Битрикс классическим способом

1. Нужно писать много кода
2. **Очень много кода!**
3. Код практически везде повторяется
4. Фильтр списка приходится собирать руками из REQUEST’а
5. Вёрстку фильтров списка тоже приходится делать руками
6. Паджинация \CAdminList не умеет нормально работать с выборкой из ORM

В общем, нам не нравилось всё!

Как же перестать жаловаться и начать делать админку красиво?

Мы вновь окунулись в мир Symfony
Источником вдохновения стал Sonata Admin Bundle

Почему?

+ Простые CRUD страницы можно делать за считанные минуты
+ Поля форм, списков и фильтров описываются декларативно
+ Для создания новой страницы админки почти не нужно писать код

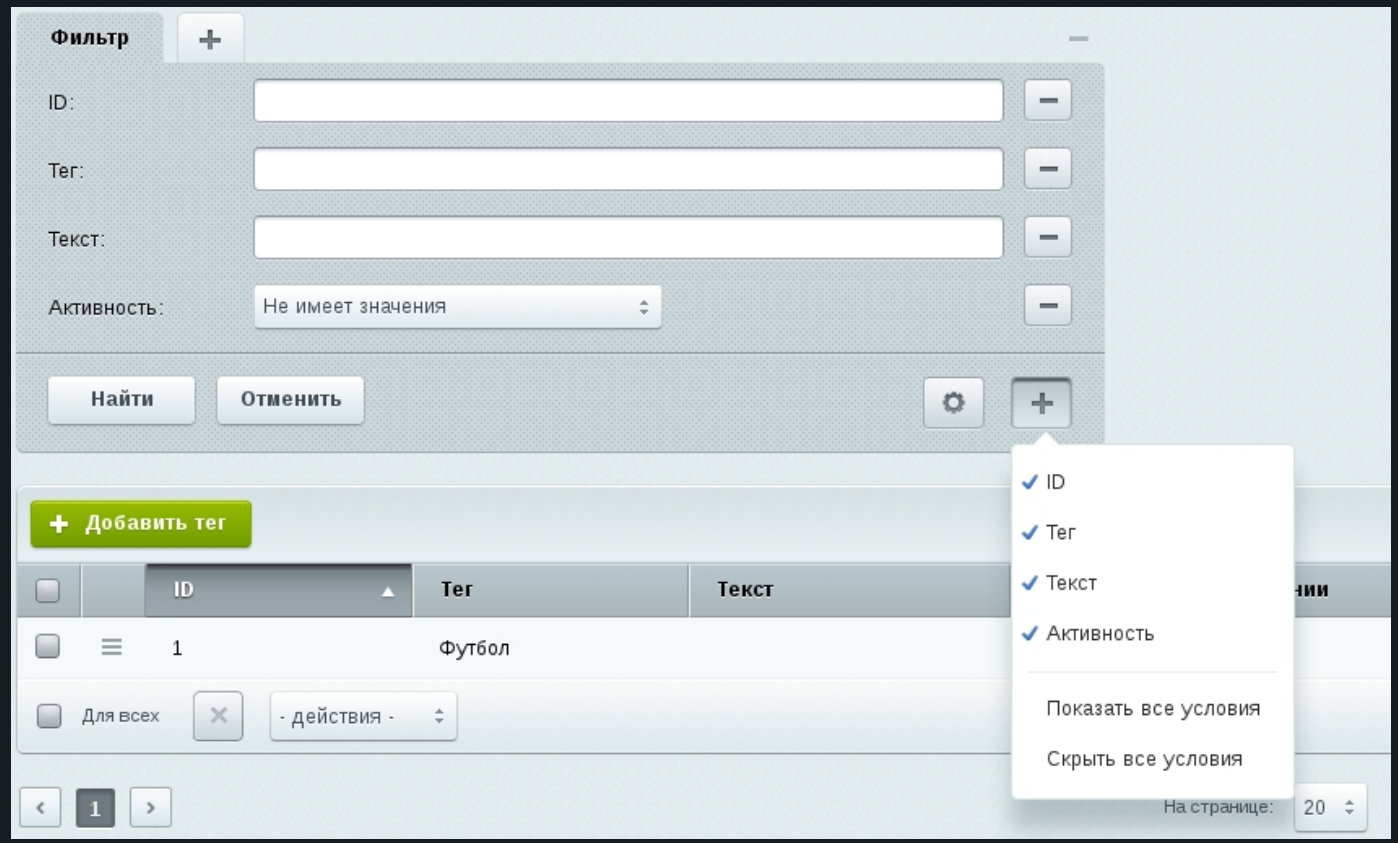
И у нас получилось

```php
/**
 * {@inheritdoc}
 */
protected function getEntity()
{
    return TagTable::getEntity();
}

/**
 * {@inheritdoc}
 */
protected function getTitle()
{
    return 'Теги';
}

/**
 * {@inheritdoc}
 */
protected function getFields()
{
    return [
        'ID' => ['title' => 'ID'],
        'NAME' => ['title' => 'Тег'],
        'DESCRIPTION' => ['title' => 'Текст'],
        'PICTURE' => [
            'title' => 'Изображение в сообщении',
            'filter' => 'N'
        ],
        'ACTIVE' => [
            'title' => 'Активность',
            'filter' => 'boolean'
        ],
    ];
}
```

На первом этапе мы создали генератор списков

1. Вся информация о типах полей берётся из ORM
2. Для создания страницы необходимо лишь указать сущность и перечислить необходимые поля
3. Внутри используются только стандартные методы Bitrix-framework

Как результат вот такая страница



Позже мы добавили и генератор форм

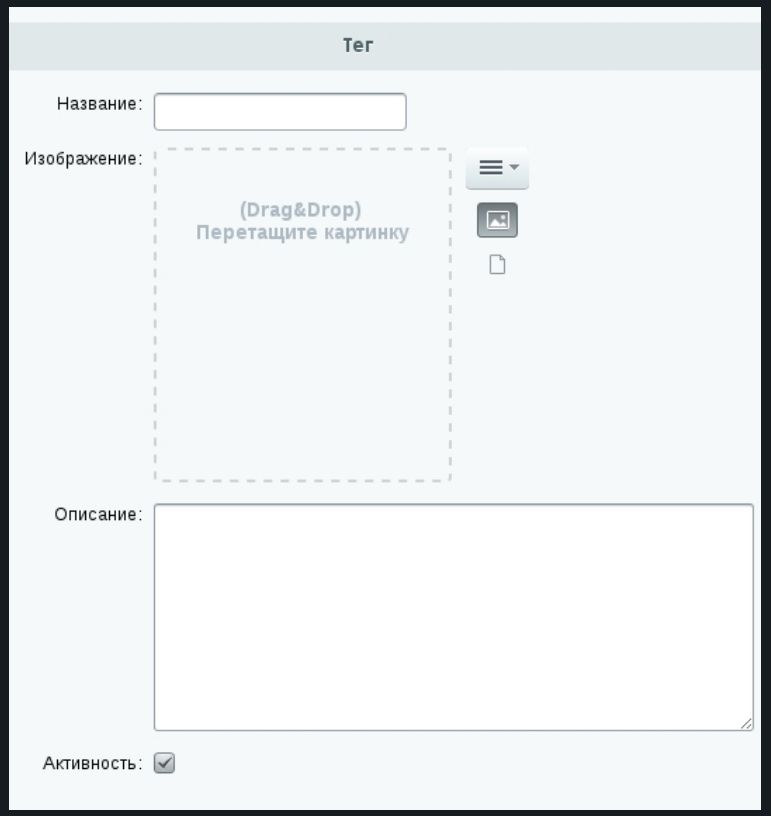
- Универсальные формы как для настроек, так и для сущностей
- Легкость добавления новых типов полей
- Все шаблоны полей в twig
- При желании можно использовать за пределами админки

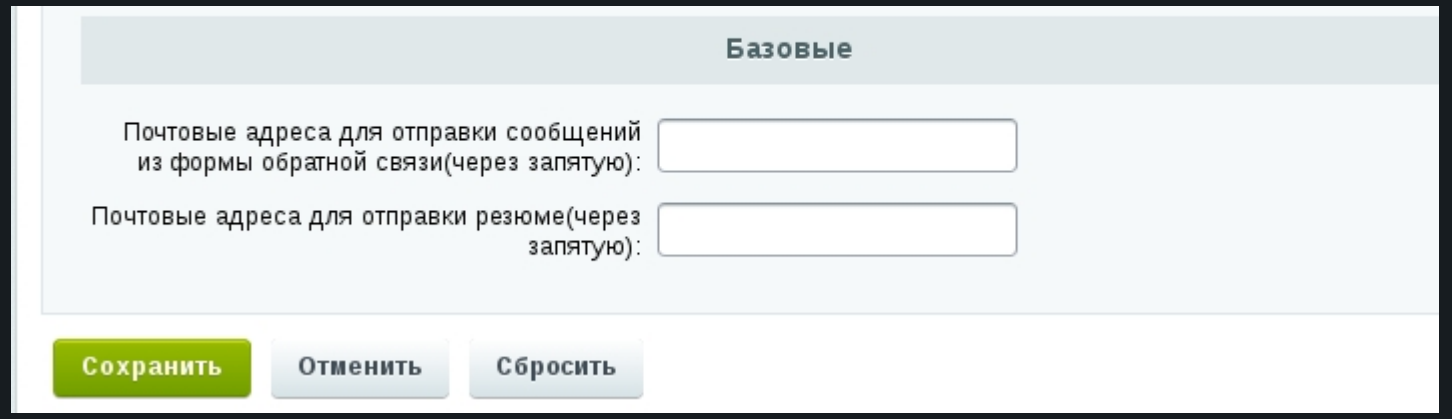
```php
$form = new OptionsForm(
    $module_id,
    'Настройки',
    [
        new Block('base_settings', 'Базовые', [
            new Text('feedback_emails', 'Почтовые адреса для отправки сообщений<br />из формы обратной связи(через запятую)'),
            new Text('hr_emails', 'Почтовые адреса для отправки резюме(через запятую)')
        ]),
        new Block('orm_link', 'Настройки привязок к ORM', [
            new CheckBox('orm_link_use_active_field', 'Использовать поле активности если есть', '', ['value' => 'Y']),
        ]),
        new Block('static_proxy', 'Настройки прокси сервера для статики', [
            new CheckBox('static_proxy_on', 'Включен', '', ['value' => 'Y']),
            new Text('static_proxy_domain', 'Домен'),
        ]),
        new Block('now_line', 'Настройки ленты "Сейчас"', [
            new Text('eom_link_pattern', 'Шаблон ссылки на поиск "#Все на матч"', '', ['size' => 50]),
        ]),
        new Block('main_news', 'Настройки основной ленты новостей', [
            new SelectBox('main_news_iblock_ids', 'Инфоблоки данные из которых попадают в ленту', '', ['multiple' => 'multiple'],
        ]),
    ]
);
```

Форма редактирования сущности

```php
class TagEditForm extends ElementEditForm
{

    /**
     * TagEditForm constructor.
     */
    public function __construct()
    {
        parent::__construct(
            'Ter',
            '\OurGrateProject\Entity\TagTable',
            [
                'add' => '/bitrix/admin/base_tag_add.php',
                'edit' => '/bitrix/admin/base_tag_edit.php',
                'list' => '/bitrix/admin/base_tag_list.php'
            ]
        );
    }
```

Как результат вот такие страницы





На создание генераторов ушло около 20 часов, польза которую это принесло - неоценима.

Вывод: Не стесняйтесь придумывать свои генераторы там где это возможно. Писать декларативные описания гораздо приятнее, чем тысячи строк шаблонного кода.

Когда всё уже было почти готово...

Мы обратили своё внимание на вывод ошибок:

```php
$arParams["ELEMENT_ID"] = intval($arParams["~ELEMENT_ID"]);
if($arParams["ELEMENT_ID"] > 0 && $arParams["ELEMENT_ID"]."" != $arParams["~ELEMENT_ID"])
{
    if (CModule::IncludeModule("iblock"))
    {
        \Bitrix\Iblock\Component\Tools::process404(
            trim($arParams["MESSAGE_404"]) ?: GetMessage("T_NEWS_DETAIL_NF")
            ,true
            ,$arParams["SET_STATUS_404"] === "Y"
            ,$arParams["SHOW_404"] === "Y"
            ,$arParams["FILE_404"]
        );
    }
    return;
}
```

- Много кода
- Нет единообразия
- Страшные константы
- Неочевидно как оно хендлится

Всего 50 строк кода...

И можно делать вот так

```php
public function getAction($id)
{
    $dm = new NewsDataManager();
    $post = $dm->getElementById((int)$id);

    if (!$post) {
        throw new HttpNotFoundException;
    }

    return ['posts' => $post];
}
```

Мы сделали Exceptions на все случаи жизни

HttpException и его наследники

- HttpNotFoundException
- HttpForbiddenException
- и т.д.

Не вызывают краха приложения, каждый из них преобразуются в страницу с понятной для клиента ошибкой,
а в случае XHR запроса в возвращают удобный json

Что в итоге?

Преимущества

+ Гибкая архитектура соблюдающая принципы SOLID и KISS
+ Новые сущности добавляются быстро и практически декларативно
+ Никакой рутины
+ Единая точка входа в приложение
+ Читабельные шаблоны
+ Часть нагрузки перенесена на клиент

Недостатки

- Поднялся уровень вхождения в проект
- Код плохо переносим в другие, уже существующие, проекты
- Некоторые фичи реализации 1с-Битрикс стали не нужны

Где это может быть полезно?

Любой проект где планируется большое количество сущностей

“Уникальные” проекты, части которых вы не планируете делать универсальными. Контроллеры, как и view, обычно плохо переносятся в другие проекты.

Что я хотел всем этим сказать?

+ Битрикс за последний годы сделал большой шаг вперёд

+ Не зацикливайтесь на стандартных подходах, сейчас Битрикс это не просто CMS, это полноценный Framework, который ни чем вас не ограничивает

+ Не бойтесь экспериментировать!

Спасибо за внимание!

Андрей Клюев
Веб-разработчик