Урок 10: Создание PHP плагина Metabot для расширения бота и платформы

В этом уроке вы научитесь создавать PHP плагин для платформы Metabot и использовать его в JavaScript коде бота. Этот навык позволяет расширять функциональность платформы, добавляя новые возможности в платформу и ваши боты. Вы поймете, как безопасно и эффективно интегрировать PHP и JavaScript, обеспечивая гибкость и мощь в разработке ботов.

Урок состоит из двух частей. Сперва вы создаете условно бэк - сам PHP плагин. Затем вы подключаете его на фронте, в боте, для доступа к PHP через JS Low-Code.

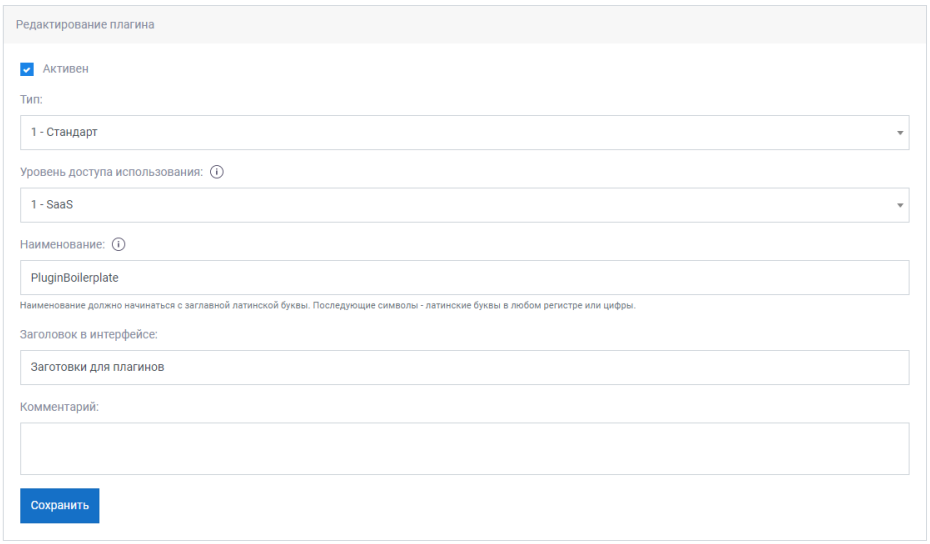
Дополнительно смотрите разделы документации:

- Плагины
- Плагины PHP

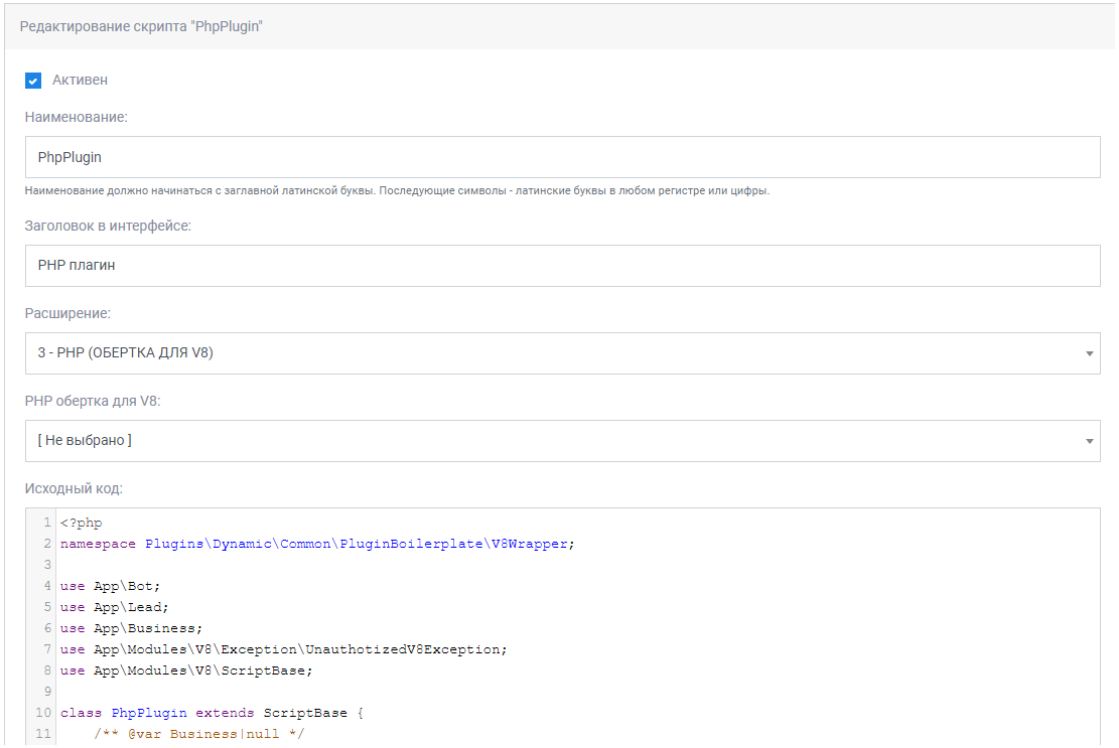
1. Создание PHP плагина

> PHP плагины могут быть только общими и добавляются только администраторами платформы из-за доступа к базе данных и другим ботам. Это мера предосторожности для обеспечения безопасности.

1. Создайте новый плагин, если необходимо. В этом уроке мы назвали плагин PluginBoilerplate.



2. В плагине создайте скрипт с расширением PHP (ОБЕРТКА ДЛЯ V8).



> Metabot поддерживает три вида скриптов в плагине: JavaScript, PHP, PHP (ОБЕРТКА для V8). **JavaScript** — это обычный Low-Code, как и скриптах самого бота, в который вы можете вынести код для повторного использования в разных скриптах в боте или в других JS плагинах. **PHP** — это модуль PHP, который используется в других PHP модулях или в PHP обертках. **PHP Wrapper for V8** или PHP (Обертка для V8) — это специальный модуль PHP, который доступен через JS на уровне бота, т.е. это некий промежуточный слой на PHP для которого можно определить JS интерфейс.

3. Скопируйте эту заготовку кода в скрипт вашей PHP обертки.

```php
<?php
// Определяем пространство имен для плагина
namespace Plugins\Dynamic\Common\PluginBoilerplate\V8Wrapper;

// Подключаем необходимые классы
use App\Bot;
use App\Lead;
use App\Business;
use App\Modules\V8\Exception\UnauthotizedV8Exception;
use App\Modules\V8\ScriptBase;

// Определяем класс PhpPlugin, наследуя его от базового класса ScriptBase
class PhpPlugin extends ScriptBase {
    /** @var Business|null */
    protected $_business = null; // Свойство для хранения объекта бизнеса, с которым взаимодействует плагин

    /** @var Bot */
    protected $_bot = null; // Свойство для хранения текущего объекта бота, с которым взаимодействует плагин

    /** @var Lead */
    protected $_lead; // Свойство для хранения объекта лида, с которым взаимодействует плагин

    /** @var Bot */
    protected $_runFromBot = null; // Свойство для хранения объекта бота, от которого был запущен плагин

    /**
     * Инициализация JavaScript обёртки
     *
     * @param array $params Параметры, передаваемые из JS скрипта
     *
     * @throws UnauthotizedV8Exception Выбрасывает исключение при нарушении политик безопасности
     */
    public function initializeJs($params)
    {
        // Инициализация свойств из переданных параметров
        $this->_business = $params['business'] ?? null;
```

```
        $this->_runFromBot = $params['bot'] ?? null;
        $this->_lead = $params['lead'] ?? null;

        // Установка объекта бота, если он доступен у лида
        if ($this->_lead && $this->_lead->bot) {
            $this->_bot = $this->_lead->bot;
        }

        // Вызов метода для проверки политик безопасности
        $this->checkPolicy();
    }

    /**
     * Метод для проверки политик безопасности.
     * Этот метод гарантирует, что взаимодействие с плагином происходит в контексте
     * правильно определенных объектов бизнеса, бота и лида.
     * Выбрасывает исключение при обнаружении несоответствий,
     * обеспечивая безопасное использование плагина.
     */
    protected function checkPolicy()
    {
        // Проверка на наличие и корректность объекта бизнеса
        if (empty($this->_business) || !($this->_business instanceof Business)) {
            throw new UnauthotizedV8Exception();
        }
        // Проверка на наличие и корректность объекта бота
        if (empty($this->_bot) || !($this->_bot instanceof Bot)) {
            throw new UnauthotizedV8Exception();
        }
        // Проверка на наличие и корректность объекта лида
        if (empty($this->_lead) || !($this->_lead instanceof Lead)) {
            throw new UnauthotizedV8Exception();
        }
        // Проверка на наличие и корректность объекта бота, который запустил плагин
        // Используется, например, когда:
        //   нужно запустить скрипт в одном боте (_runFromBot),
        //   а получить тикет/др. сущности по другому боту (_bot).
        //   _bot и _runFromBot запоминают, что к чему принадлежит и откуда получено.
        if (empty($this->_runFromBot) || !($this->_runFromBot instanceof Bot)) {
            throw new UnauthotizedV8Exception();
```

```
        }

        // Проверка на на принадлежность бота, указанному бизнесу
        if (!empty($this->_bot) && $this->_bot->business_id != $this->_business->id) {
            throw new UnauthotizedV8Exception();
        }

        // Здесь могут быть добавлены дополнительные условия безопасности
    }

    // Метод для проверки работоспособности плагина
    public function itWorks() {
        $this->checkPolicy();

        return "Plugin works!"; // Возвращаем подтверждающее сообщение
    }
}
```

> Создание PHP плагинов - ответственный процесс, требующий строгого контроля безопасности. Подробнее о `checkPolicy()` смотрите ниже.

4. Важно правильно указать `namespace`, иначе ваш плагин не подключится. Замените ИмяПлагина на название вашего плагина. В нашем примере это PluginBoilerplate.

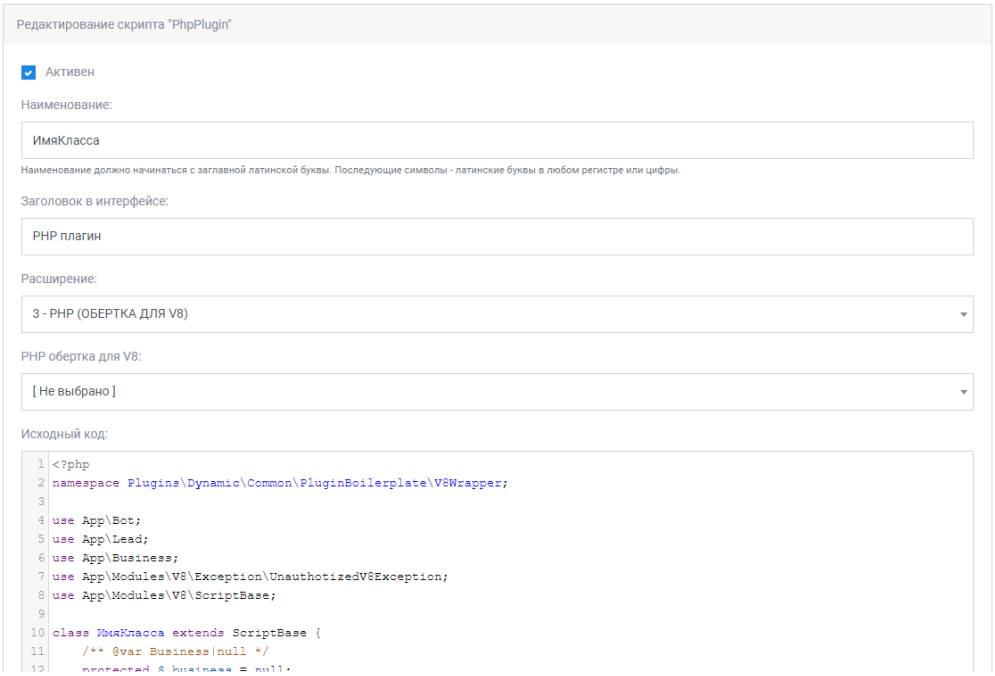
```
namespace Plugins\Dynamic\Common\ИмяПлагина\V8Wrapper;
```

> Указывайте имя плагина с учетом регистра. PluginBoilerplate и pluginBoilerplate это разные плагины!

5. Важно правильно указать имя класса в соответствии с названием скрипта плагина.

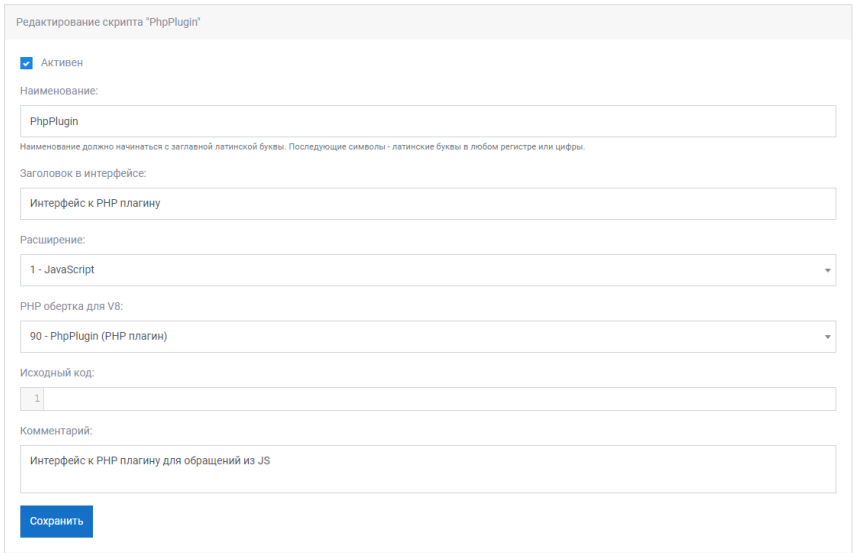
Вместо ИмяКласса укажите название вашего класса.

```
class ИмяКласса extends ScriptBase {
```

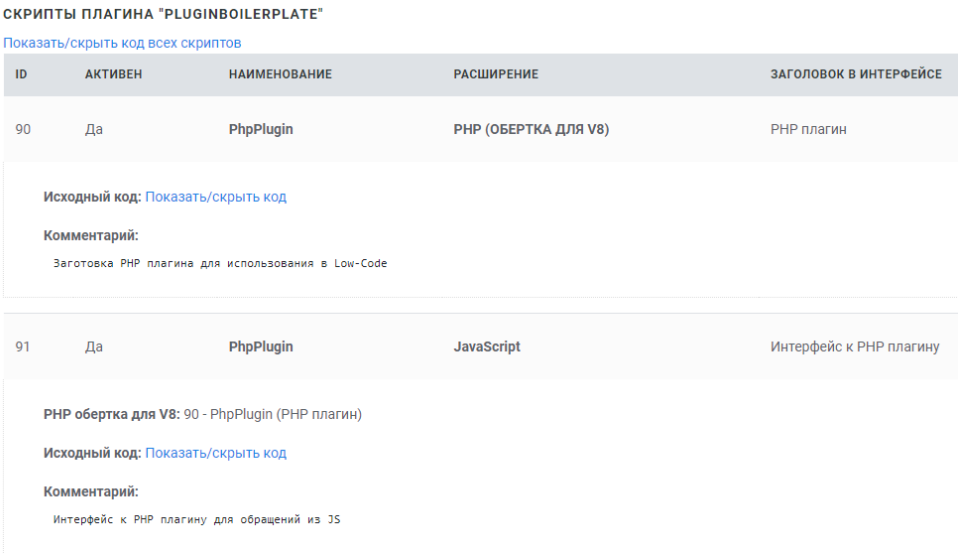


> Название класса должно совпадать с названием скрипта PHP обертки! В противном случае ваш плагин не будет работать!

6. Создайте JS скрипт в плагине. Выберите созданную ранее PHP обертку. Название может быть любое, но мы рекомендуем JS скрипт называть также как PHP скрипт, это поможет вам избежать путаницы в дальнейшем.



В итоге, у вас должен получиться плагин с двумя скриптами:



СКРИПТЫ ПЛАГИНА "PLUGINBOILERPLATE"

Показать/скрыть код всех скриптов

ID	АКТИВЕН	НАИМЕНОВАНИЕ	РАСШИРЕНИЕ	ЗАГОЛОВОК В ИНТЕРФЕЙСЕ
90	Да	PhpPlugin	PHP (ОБЕРТКА ДЛЯ V8)	PHP плагин

Исходный код: Показать/скрыть код

Комментарий:

```
Заготовка PHP плагина для использования в Low-Code
```

91	Да	PhpPlugin	JavaScript	Интерфейс к PHP плагину

PHP обертка для V8: 90 - PhpPlugin (PHP плагин)

Исходный код: Показать/скрыть код

Комментарий:

```
Интерфейс к PHP плагину для обращений из JS
```

Дополнение: Важность методов `initializeJs` и `checkPolicy`

В создании PHP плагинов для Metabot, особое внимание следует уделить методам `initializeJs` и `checkPolicy`. Эти методы играют ключевую роль в обеспечении безопасности и правильной инициализации плагина.

Правильная реализация методов `initializeJs` и `checkPolicy` гарантирует, что ваш PHP плагин будет не только функциональным, но и безопасным для использования в рамках платформы Metabot. Это обеспечит стабильность и надежность ваших решений на платформе.

Метод	**Описание**
`initializeJs`	Этот метод отвечает за инициализацию JavaScript обёртки, которая является мостом между PHP кодом плагина и его использованием в JavaScript. При вызове этого метода передаются необходимые параметры из контекста бота, такие как информация о бизнесе, боте и пользователе (лиде). **Особенности `initializeJs`:** • **Передача контекстных данных:** Включает в себя данные о бизнесе, боте, и пользователе. • **Инициализация состояния:** Устанавливает начальное состояние для работы плагина.

`checkPolicy`	Метод `checkPolicy` выполняет проверку прав доступа и политик безопасности. Это критически важно, поскольку плагины PHP могут взаимодействовать с базой данных, с другими ботами и компонентами системы. Проверка обеспечивает, что плагин используется только в разрешенных и безопасных контекстах. **Значимость `checkPolicy`:** • **Проверка доступа:** Удостоверяется, что запросы к плагину исходят из авторизованных источников. • **Защита от несанкционированного доступа**: Предотвращает использование плагина вне предполагаемого бизнес-контекста.

2. Использование PHP плагина

Теперь подключим созданный PHP скрипт в диалоговом сценарии нашего чат-бота.

1. Создайте команды Вызвать JavaScript и разместите в ней следующий код.

```
// Подключаем PHP плагин.  Эта команда загружает JavaScript обёртку для PHP плагина.
require("Common.PluginBoilerplate.PhpPlugin")

// Создаём переменную 'plugin', которая ссылается на экземпляр класса PHP плагина.
let plugin = CommonPluginBoilerplatePhpPlugin

// Вызываем метод 'itWorks()' у объекта плагина.
let result = plugin.itWorks()

// Используем функцию 'debug' для вывода результата работы метода в отладочную консоль.
debug(result)
```

Разберем, что здесь происходит и обсудим важные нюансы.

1. Подключение JS Плагина.

```
require("Common.PluginBoilerplate.PhpPlugin")
```

Common.PluginBoilerplate.PhpPlugin — это идентификатор JS скрипта из нашего плагина в системе Metabot.

Обратите внимание, это название JS скрипта, который связан с PHP скрипом! Это не название PHP скрипта!

Регистр важен!

Не используйте `let plugin = require("Common.PluginBoilerplate.PhpPlugin")` синтаксис для подключения PHP. Используйте точный синтаксис `require` без присваивания переменной,

2. Обращение к PHP плагину.

После вызова require, вы можете обращаться к экземпляру PHP класса.

Платформа автоматически создает экземпляр. Название формируется путём объединения названий уровня доступа ('Common'), названия плагина ('PluginBoilerplate') и самого класса PHP ('PhpPlugin').

```
let result = CommonPluginBoilerplatePhpPlugin.itWorks()
```

Либо вы можете присвоить экземпляр в более короткую переменную:

```
let plugin = CommonPluginBoilerplatePhpPlugin
let result = plugin.itWorks()
```

3. Базовая заготовка PHP Plugin Boilerplate содержит метод `itWorks()`, который возвращает сообщение о корректной работе плагина.

Если всё выполнено правильно, вы увидите сообщение "It works!" в вашем боте.

Дополнение: Кэширование PHP плагинов и перезапуск очереди

При работе с PHP плагинами на платформе Metabot важно помнить о механизме кэширования, который может влиять на процесс разработки и отладки. PHP плагины кэшируются на уровне языка, что означает, что изменения, внесенные в код плагина, не будут немедленно отражены в работе бота.

Когда вы вносите изменения в PHP код плагина, для того чтобы они вступили в силу, необходимо перезапустить очередь на платформе Metabot. Это можно сделать следующим образом:

1. **Переход в Настройки Платформы:**
   - Откройте раздел настроек платформы Metabot. Это центральное место, где управляются основные параметры вашего бота и платформы в целом.

2. **Отключение и Включение Cron Scheduler:**
   - Найдите опцию или раздел, относящийся к Cron Scheduler (планировщику заданий).
   - Сначала отключите Cron Scheduler. Это остановит текущую очередь заданий, включая выполнение задач связанных с PHP плагинами.
   - Затем включите Cron Scheduler обратно. Это инициирует новую сессию очереди, при этом кэш PHP плагинов будет очищен и ваши изменения начнут действовать.

3. Отладка PHP плагина

Отладка - это важный элемент разработки, который требует терпения и внимания к деталям. Следуйте этим шагам, чтобы эффективно находить и устранять проблемы с вашими PHP плагинами в Metabot.

1: Проверка логов платформы

- **Доступ к Логам:** Обычно почти все ошибки можно получить через отладку бота(смотрите ниже). Однако, можно допустить ошибку, которая приведет к тому, что PHP на платформе не соберется и до уведомления в боте платформа просто не дойдет. В этом случае, запросите доступ к логам платформы Metabot. Обычно они находятся в `/storage/logs`. Логи могут предоставить ценную информацию о внутренних ошибках платформы и вашего плагина.
- **Чтение Логов:** Ищите ошибки, связанные с вашим плагином, включая исключения, предупреждения и другие сообщения об ошибках.

2: Режим отладки бота

- **Включение Режима Отладки:** Включите режим отладки в настройках вашего бота. Это позволит получать подробные сообщения об ошибках.
- **Отладка на Уровне Лида:** Для конкретных лидов, активируйте высший уровень отладки. Это обеспечит детальную информацию об ошибках, возникающих при взаимодействии с вашим плагином.

3: Проверка конфигурации плагина

- **Проверка Класса Обертки PHP:** Убедитесь, что класс обёртки PHP правильно спроектирован. Название класса должно соответствовать названию скрипта обёртки.
- **Обязательные Методы:** Проверьте, что в вашем классе реализованы методы `initializeJs` и `checkPolicy`. Эти методы критически важны для функционирования плагина.

4: Проверка JavaScript интерфейса

- **Создание JS Интерфейса:** Убедитесь, что JavaScript интерфейс для доступа к PHP обёртке создан корректно.
- **Синтаксис `require`:** Проверьте, правильно ли вы используете `require` для подключения плагина. Удостоверьтесь, что название плагина и скрипта указаны верно.

5: Проверка интеграции PHP и JavaScript

- **Название Класса PHP:** Проверьте, что название PHP класса, которое склеивается из трех компонентов (уровень доступа, название плагина, название класса), указано верно.
- **Тестирование Методов:** Попробуйте вызвать различные методы вашего PHP класса через JavaScript, чтобы проверить их работоспособность.

6: Обращение за помощью

- **Сообщество и Поддержка:** Если вы столкнулись с трудноуловимыми или сложными проблемами, не стесняйтесь обращаться за помощью к сообществу Metabot или в службу поддержки.
- **Документация:** Периодически обращайтесь к официальной документации Metabot, где могут быть ответы на ваши вопросы.

Заключение

Поздравляем! Теперь вы умеете создавать PHP плагины для Metabot и интегрировать их в JS код бота. Это открывает новые возможности для расширения функционала платформы.

Версия #14
Artem Garashko создал 8 February 2024 10:56:04
Ирина Петрова обновил 20 March 2024 08:58:35