Как сделать свой пакет для Laravel за 10 минут

24 ОКТЯБРЯ 2019

Во фреймворке Laravel мы можем писать библиотеки и вызывать их функции, не беспокоясь о том, как это было реализовано. Мы можем использовать их повторно и управлять отдельно от основного кода нашего проекта. Речь идет о так называемых пакетах (packages) Laravel.

Что такое пакеты?

Пакеты представляют собой автономный код, который разработчики могут импортировать в свои проекты с помощью инструмента управления пакетами (я опишу этот инструмент через минуту). Вместо того, чтобы копипастить код библиотек по проектам, разработчикам нужно всего лишь импортировать пакет и вызвать любой сервис, который он предоставляет.

Этот код имеет свои собственные версии, тесты и зависимости. Разработчики могут рассматривать пакеты, как небольшие API в рамках проекта. Разработчикам нужно только предоставить требуемые входные данные, пакет их обработает, согласно своему предназначению и предоставит выходные данные.

На уровне компании создание собственных пакетов ведет к высокой производительности и эффективности. Пакеты будут стандартизированы, и если потребуется обновление, в случае проблемы с уязвимостью или ошибками, то разработчики пакета, исправят их, а авторам проектов всего лишь нужно обновить сам пакет и задеплоить. Меньше головной боли и работы для разработчиков, больше доходов для компании. Все счастливы и это главное!

Почему нужно использовать пакеты?

Когда у нас много проектов, имеющих схожий функционал, то было бы лучше, если бы мы начали реализовывать его в отдельных пакетах. Когда у вас есть команда или человек, обслуживающие пакет, скажем, в GitLab, то гораздо проще совместно работать над возможными исправлениями ошибок или улучшениями. Это означает, что каждый пакет будет проходить тот же процесс, что и обычный проект.

Другая проблема, которую решают пакеты, — когда разработчикам нужно изменить свою библиотеку, для исправления ошибки, добавления нового функционала или устранения уязвимости. Внедрение всего этого во все проекты, где использовалась библиотека, весьма сложная задача. А создание пакетов отличная возможность избежать этих проблем в будущем.

Как мы управляем пакетами Laravel?

Composer — это инструмент управления зависимостями пакетов для PHP. Он помогает нам управлять нашими пакетами, вытягивая все зависимости, которые требуются каждому пакету, без дублирования их в каталоге vendor. Вы можете определить нужную версию пакета и даже настроить автоматическое обновление до последней версии при каждом запуске обновлений.

Вам не нужно изобретать велосипед. Представьте, если каждый должен открыть электричество, перед тем как зажечь лампочку? Да, это преувеличение, но я хочу, чтобы вы поняли мою точку зрения.

В этой статье я расскажу вам, как можно создать свой собственный пакет для Laravel. Мы будем использовать GitLab для версии и Composer для импорта в наш проект. Чтобы получить общее представление о том, как создаются пакеты, мы создадим простой пакет greetr. Если бы у меня была такая возможность, я бы проработал каждый шаг и создал бы более реалистичный и пригодный для повторного использования пакет.

Создание пакетов Laravel

Требования

Прежде чем начнем, я надеюсь, что в вашей системе уже установлены Composer и Git. Также необходимо, чтобы вы могли создать удаленный репозиторий для любого VCS сервиса, например GitLab, GitHub или Bitbucket. В статье мы будем использовать GitLab и Laravel 5.8.

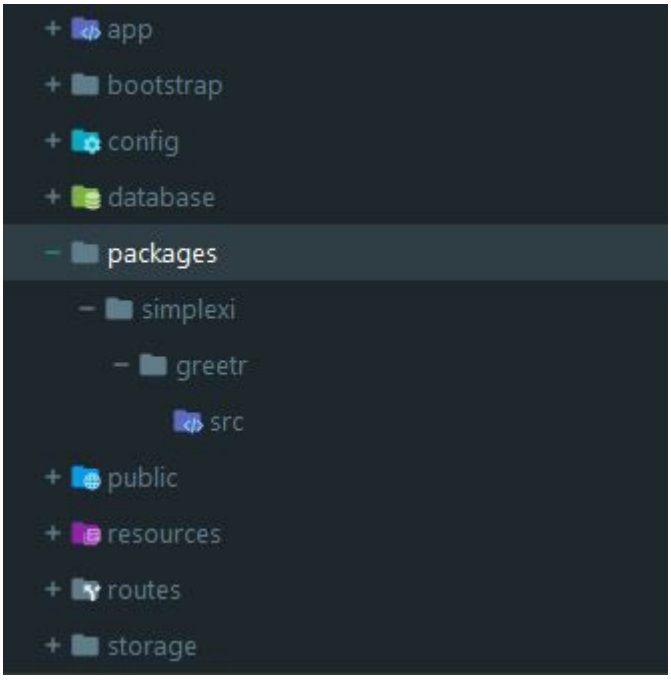
Создаем новый проект Laravel

Создадим проект под названием «greetr», который просто будет показывать приветствие при вызове. Используем Composer в командной строке для создания проекта.

```
composer create-project — prefer-dist laravel/laravel greetr
```

Файловая структура

В корне проекта создайте файловую структуру, как указано на изображении. В ней будет размещен наш проект.



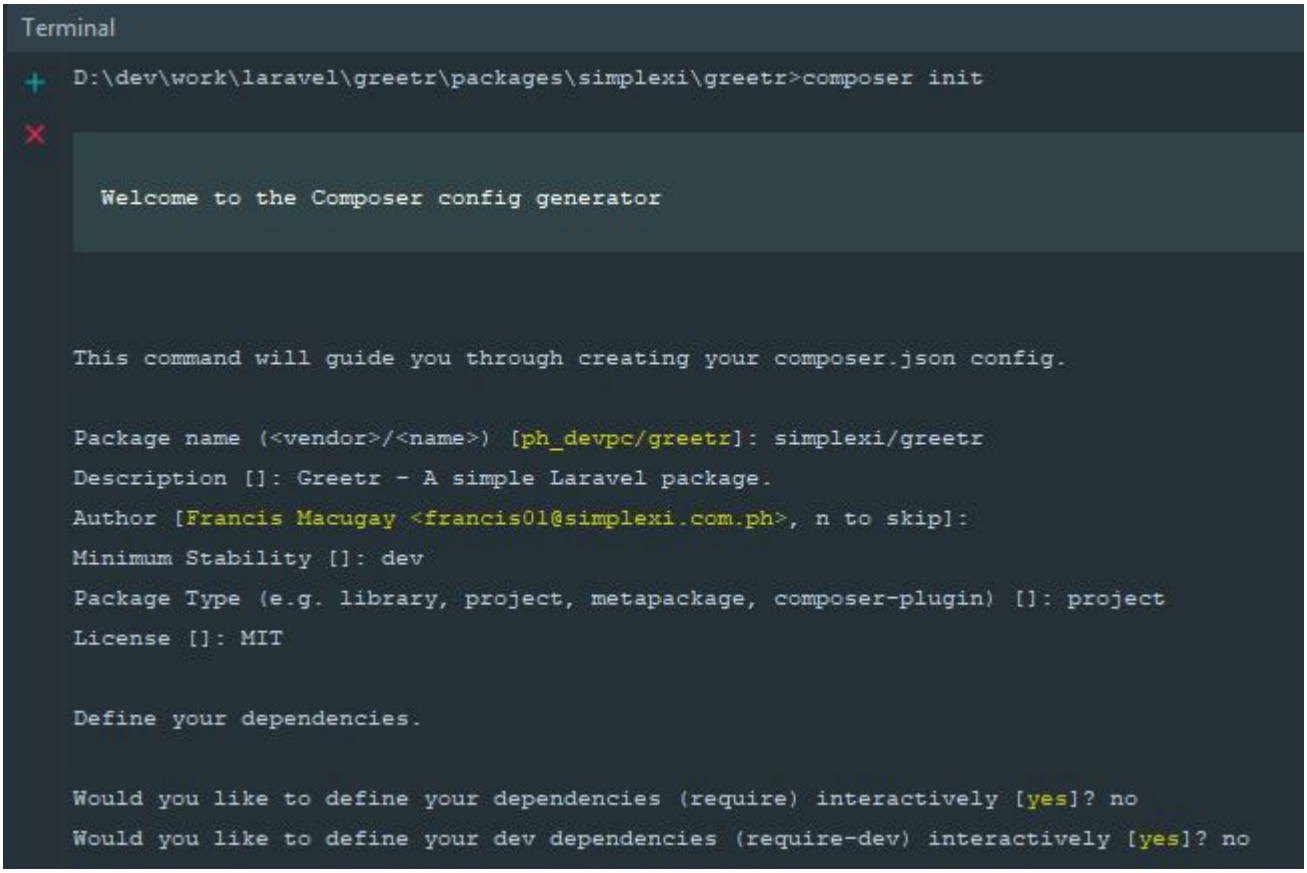
Обратите внимание, что внутри папки `packages` должна быть быть папка с названием вендора, в нашем случае это `simplexi` , а в ней папка с названием пакета — `greetr` . Папка `src` будет содержать исходный код нашего пакета. В некоторых случаях вы можете также создать ветку `dist` , если компилируете свои ресурсы для продакшн сервера. Тогда разработчики, при установке вашего пакета через Composer, будут иметь возможность выбирать между исходниками из папки `src` и дистрибутивом из папки dist.

Пакет composer.json

В командной строке перейдите в `packages/simplexi/greetr` . Теперь нам нужно инициализировать эту папку как пакет композера, выполнив следующую команду:

```
composer init
```

Эта команда запросит информацию о вашем пакете. Вы можете согласится с дефолтными значениями, просто нажав enter и позже подредактировать свежесозданный файл composer.json.



После инициализации мы можем внести изменения в файл composer.json

```
// packages/simplexi/greetr/composer.json

{
    "name": "simplexi/greetr",
    "description": "Greetr - A simple Laravel package.",
    "type": "project",
    "license": "MIT",
    "authors": [
        {
            "name": "Francis Macugay",
            "email": "francis01@simplexi.com.ph"
        }
    ],
    "minimum-stability": "dev",
    "autoload": {
        "psr-4": {
            "Simplexi\\Greetr\\": "src/"
        }
    },
    "require": {}
}
```

Единственное, что я добавил — значение объекта `autoload`. Это позволит автоматически загрузить пакет с указанным пространством имен `Simplexi\Greetr`. Вы также можете включить любые зависимости пакета в свойство `require`. Чтобы узнать больше о схеме файлов composer.json, нажмите здесь.

Обратите внимание, что composer.json проекта отличается от composer.json пакета. Убедитесь, что вы редактируете правильный файл.

Проект composer.json

Укажите пространство имен для вашего пакета, добавив строку `"Simplexi\\Greetr\\": “packages/simplexi/greetr/src" в autoload->psr-4` вашего файла проекта composer.json. При этом классы вашего пакет в папке `src` будут автоматически загружены и доступны для использования в проекте.

```
// composer.json

"autoload": {
    "psr-4": {
        "App\\": "app/",
        "Simplexi\\Greetr\\": "packages/simplexi/greetr/src"
    },
    "classmap": [
        "database/seeds",
        "database/factories"
    ]
},
```

Существуют и другие варианты обнаружения пакетов, которые описаны здесь. Так как мы редактировали composer.json проекта, то необходимо его перезагрузить, выполнив следующую команду в корневом каталоге проекта:

```
composer dump-autoload
```

Класс Greetr.php

Создадим класс `Greetr.php` в папке `src`

```
// packages/simplexi/greetr/src/Greetr.php

namespace Simplexi\Greetr;

class Greetr
{
    public function greet(String $sName)
    {
        return 'Hi ' . $sName . '! How are you doing today?';
    }
}
```

Мы должны указать пространство имен для этого класса, которое должно быть таким же, как мы указывали в composer.json. В нашем случае, следуя соглашению о пространстве имен в Laravel, это будет `Simplexi\Greetr`.

Конечно, стоило бы следовать той же структуре, что и основном проекте. Но мы просто показываем обычное приветствие в браузере. Имейте в виду, что по мере роста вашего пакета рекомендуется хранить общедоступные функции в одном-двух сервисных классах. Все остальные классы не должны быть доступны за пределами пакета. Ваш пакет должен быть управляем независимо от других пакетов или основного проекта.

Тестируем пакет

Наш пакет готов к тестированию. Но, для начала, давайте создадим маршрут и вызовем класс `Greetr` нашего пакета из этого маршрута.

```
// routes/web.php

use Simplexi\Greetr\Greetr;

Route::get('/greet/{name}', function($sName) {
    $oGreetr = new Greetr();
    return $oGreetr->greet($sName);
});
```

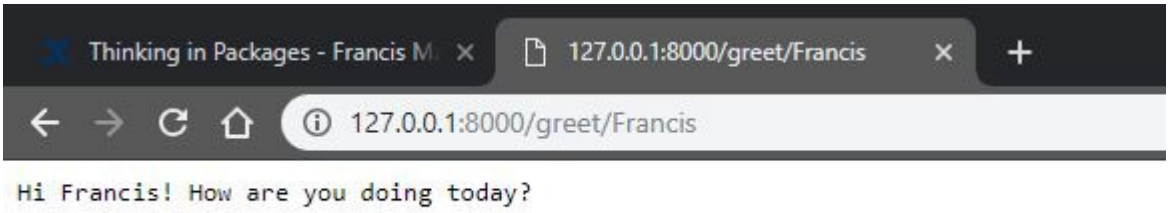
Мы можем импортировать класс `Greetr` из нашего пакета, указав его пространство имен с помощью ключевого слова `use` . Затем мы можем создать экземпляр класса и вызвать его функции в файле маршрута проекта.

Запустите сервер разработки Laravel, выполнив следующую команду в корневом каталоге вашего проекта:

```
php artisan serve
Laravel development server started: <http://127.0.0.1:8000>
```

Откройте браузер и введите следующий адрес:

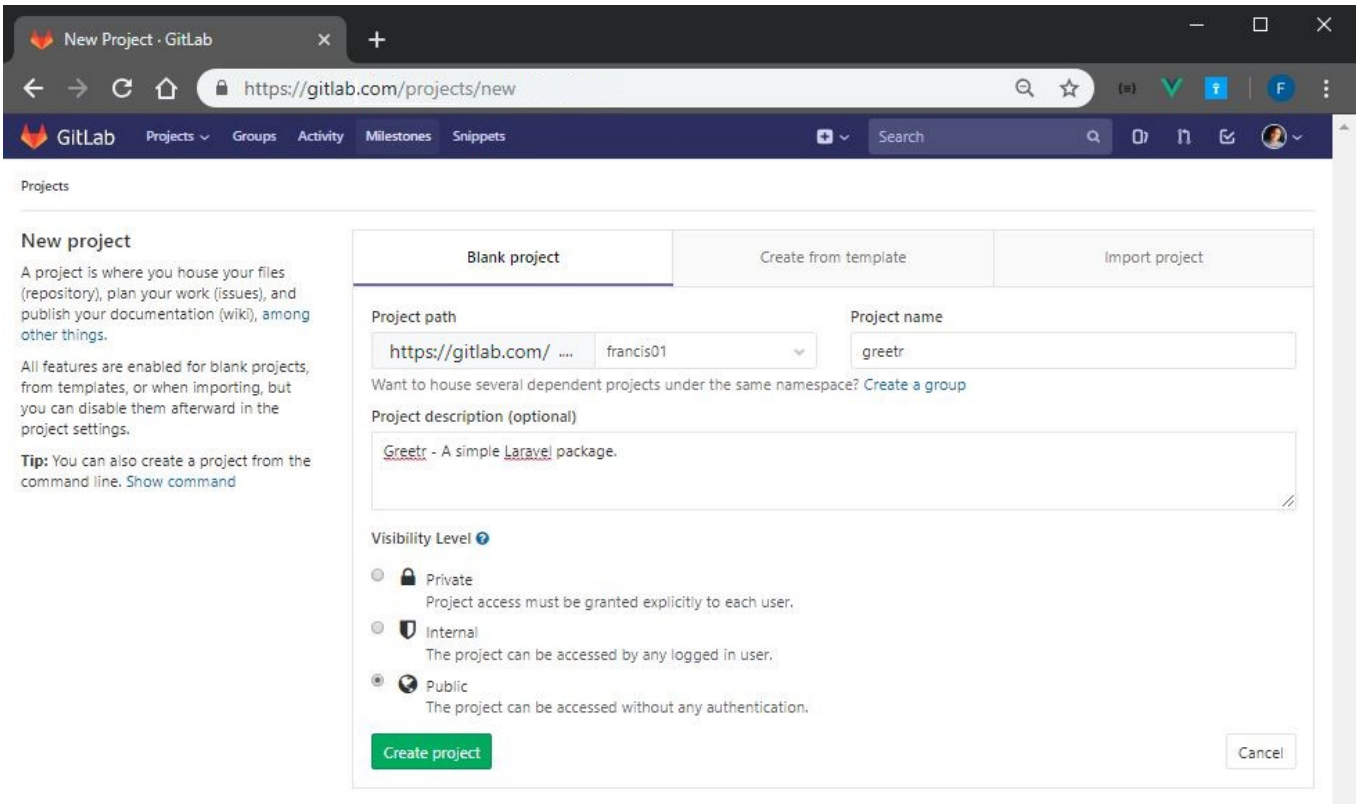
`http://127.0.0.1:8000/greetr/Francis`



Публикация пакета на GitLab

Создание GitLab-репозитория

Во-первых, мы должны создать новый проект в GitLab. Убедитесь, что уровень видимости установлен как `public`. Мы также можем установить его как приватный, если нужно контролировать, кто может использовать наш пакет. Для этого потребуются дополнительные шаги. Но в этой статье мы оставим его общедоступным.



Перейдите в каталог `packages/simplexi/greetr`. Инициализируйте git, выполните коммит и протегируйте версию, выполнив следующие команды.

```
// packages/simplexi/greetr

git init
git checkout -b master
git add .
git commit -m "initial commit"
git tag 1.0.0
```

Передача локального кода на удаленный сервер GitLab

Убедитесь, что ваша учетная запись настроена на компьютере, чтобы можно было отправить исходный код на сервер GitLab. Нам нужно всего лишь запушить в GitLab каталог `packages/simplexi/greetr`. Желательно включить такиее файлы, как `CHANGELOG.md` и `readme.md`, чтобы другие разработчики могли узнать больше о нашем пакете.

Чтобы запушить наш код во вновь созданный проект GitLab, выполните следующие команды. Обязательно замените исходный адрес адресом вашего репозитория.

```
// packages/simplexi/greetr

git remote add origin https://gitlab.com/francis01/greetr.git
git push -u origin --all
git push -u origin --tags
```

Все готово. Теперь другие разработчики могут устанавливать наш пакет в свои собственные проекты, дабы не тратить свое время на их разработку. В следующем разделе мы попытаемся импортировать наш пакет в новый проект на Laravel.

Импорт пакета в Laravel

Теперь мы можем импортировать наш пакет с GitLab в новый проект Laravel через Composer. Создайте новый проект на локалке.

Настройка зависимости пакета в composer.json

Укажите в файле composer.json нового проекта зависимость от нашего пакета и адрес его репозитория. Это необходимо, что композер знал, где его можно получить, кроме стандартных репозиториев packagist.

```
// composer.json

...

"require": {
    "php": "^7.1.3",
    "fideloper/proxy": "^4.0",
    "laravel/framework": "5.7.*",
    "laravel/tinker": "^1.0",
    "simplexi/greetr": "^1.0.0"
},

...

"repositories": [
    {
        "type": "vcs",
        "url": "https://gitlab.com/francis01/greetr"
    }
]

...
```

Свойство репозитория содержит список всех не-packagist репозиториев. Если вы собираетесь установить несколько пакетов из одного домена GitLab, то вместо указания репозитория каждого пакета вы можете использовать тип «composer» и указывать в адресе только домен. Однако в этом случае администратор вашего GitLab должен сделать дополнительные настройки. А пока давайте остановимся на одном репозитории с типом «vcs».

Запустите следующую команду, для того чтобы загрузить внесенные нами изменения в файл composer.json проекта и установить наш пакет.

```
composer update
```

Теперь надо найти наш пакет в каталоге `vendor`. Надо использовать наш пакет, также как и любой другой, которые нам могут

потребоваться в нашем проекте. Мы можем протестировать наш пакет, как в предыдущем разделе, но в этот раз сделаем это в новом проекте.

Выбор версии пакета

Когда другие разработчики используют ваш пакет, то они будут находить в нем ошибки, предлагать улучшения и новые функции. Вам, как мейнтейнеру пакета, придется это делать и выпускать новые версии. Все разработчики, использующие ваш пакет, будут проинформированы об исправлениях в документации репозитория.

Дело в том, что каждый раз, когда выпускается новая версия, разработчики имеют возможность обновиться до неё, если у них проявлялась указанная ошибка, ну или решат этого не делать. В композере это сделать очень просто, достаточно прописать нужную версию в файле composer.json проекта и запустить `composer update`. Потом протестировать и задеплоить. Если новая версия не заработает, то можно легко вернуться к предыдущей. Это одно из полезных преимуществ Composer, который управляет зависимостями ваших пакетов.

В этом статье мы рассмотрели только верхушку пакетов в Ларавел. С пакетами можно сделать гораздо больше. А пока вы можете ознакомиться с официальной документацией по пакетам Laravel.

Резюме

Внедрение пакетов в проекты может иметь небольшие накладные расходы вначале разработки новых проектов. Но в долгосрочной перспективе наша база библиотек становится гораздо более управляемой.

Такой способ совместного использования пакетов, на мой взгляд, делает код гораздо более надежным. Пакетное мышление значительно облегчает нашу повседневную работу разработчиков, ведь уже есть множество пакетов, которые мы можем использовать в наших проектах. Обмен пакетами позволяет продлить жизнь наших творений и получить шанс быть оцененными за усилия, вложенные в него. Так что, прекращайте изобретать велосипед и начинайте собирать пакеты прямо сейчас!

Ссылки

- https://laravel.com/docs/
- https://getcomposer.org/doc/04-schema.md
- https://medium.com/teknomuslim/how-to-build-your-own-laravel-package-chapter-1-9ffc0da9c04d
- https://pineco.de/laravel-package-development-basics/

Автор: Francis Macugay
Перевод: Алексей Широков

Задать вопросы по урокам можно на нашем форуме.

РУБРИКИ

Пакеты, Статьи

ТЕГИ

composer laravel php пакеты

СТАТЬИ

Глобальные настройки приложения

В приложениях часто нужно хранить некоторые глобальные настройки, причем эти параметры не относятся к конкретной модели, например пользователю, а к...

СТАТЬИ

Еще один способ тестирования запросов в Laravel

В этом уроке я покажу вам еще один способ проверки запроса формы, гораздо более чистый и повышающий удобство сопровождения ваших...

Поиск …

УРОКИ LARAVEL

Сайт посвященный фреймворку Laravel. Уроки на русском языке, полезные пакеты и свежие новости.

email: ash@demiart.ru