Homebrew и разобраться с соответствующей документацией, но в целом путь от начальной установки до обслуживания приложений — несколько простых шагов.

Установите среду разработки Valet — для этого прочтите в документации по адресу http://bit.ly/2U7uy7b актуальные инструкции по установке — и укажите один или несколько каталогов, в которых будут находиться ваши сайты. Так, я запустил команду `valet park` из каталога `~/Sites` на моем устройстве, где расположены все приложения, над которыми я работаю. Теперь вы можете открыть папку в своем браузере, просто добавив окончание `.test` к названию каталога.

Valet позволяет легко настроить обслуживание всех вложенных папок определенного каталога в формате *`{folderName}`*`.test` с помощью команды `valet park`; только одного каталога — командой `valet link`. Открыть для каталога домен, обслуживаемый этой средой, можно, введя команду `valet open`; настроить обслуживание сайта с использованием протокола HTTPS — `valet secure`; открыть туннель ngrok для совместного использования сайта — `valet share`.

Laravel Homestead

Homestead используется для настройки локальной среды разработки. Это инструмент конфигурирования, устанавливаемый поверх Vagrant — программного обеспечения для управления виртуальными машинами — и предоставляющий предварительно сконфигурированный образ виртуальной машины, который идеально настроен для разработки с помощью Laravel и имитирует наиболее типичный вариант эксплуатационной среды для сайтов Laravel. Homestead, вероятно, лучший вариант локальной среды разработки для программистов, работающих в Windows.

Документация по Homestead регулярно обновляется (http://bit.ly/2FwQ7EZ), поэтому ознакомьтесь с ней, если хотите узнать, как настроить и использовать этот инструмент.

Vessel

Это не официальный проект Laravel, но Крис Фидао из Servers for Hackers (https://serversforhackers.com/) и Shipping Docker (https://shippingdocker.com/) сделал простой инструмент создания сред Docker для разработки Laravel под названием Vessel (https://vessel.shippingdocker.com/). Посмотрите документацию, чтобы узнать больше.

Создание нового проекта Laravel

Существует два способа создания нового проекта, но они запускаются из командной строки. Первый способ: глобально установить установщик Laravel (с помощью менеджера пакетов Composer), а второй — использовать функцию `create-project` менеджера пакетов Composer.

Вы можете узнать об этих вариантах более подробно на странице документации по установке (http://bit.ly/2HFzBFY), но я бы порекомендовал использовать установщик Laravel.

Установка Laravel с помощью установщика Laravel

Если у вас глобально установлен менеджер пакетов Composer, то для Laravel достаточно выполнить следующую команду:

```
composer global require "laravel/installer"
```

Далее можно легко развернуть новый проект, выполнив из командной строки такую команду:

```
laravel new projectName
```

Эта команда создаст в текущем каталоге подкаталог с именем *{projectName}* и установит в него пустой проект Laravel.

Установка Laravel с помощью функции create-project менеджера пакетов Composer

Можно воспользоваться функцией `create-project` менеджера пакетов Composer, которая позволяет создавать проекты с определенной структурой. Для этого выполните следующую команду:

```
composer create-project laravel/laravel projectName
```

В текущем каталоге также будет создан подкаталог с именем *{projectName}* с предварительным каркасом приложения.

Lambo: улучшенный вариант команды laravel new

Я написал простой сценарий Lambo (http://bit.ly/2TCcQo8) для автоматизации повторяющихся действий при создании нового проекта Laravel.

Lambo запускает команду `laravel new`, после чего регистрирует ваш код в репозитории Git, задает в качестве параметров доступа указанные в файле `.env` значения по умолчанию, открывает проект в браузере, (опционально) открывает его в редакторе и выполняет еще несколько полезных действий, касающихся создания проекта.

Вы можете установить Lambo с помощью команды `global require` менеджера пакетов Composer:

```
composer global require tightenco/lambo
```

После этого его можно будет использовать так же, как команду `laravel new`:

```
cd Sites
lambo my-new-project
```

Структура каталогов Laravel

При открытии каталога с заготовкой приложения Laravel вы увидите следующие файлы и каталоги:

```
app/
bootstrap/
config/
public/
resources/
routes/
storage/
tests/
vendor/
.editorconfig
.env
.env.example
.gitattributes
.gitignore
artisan
composer.json
composer.lock
package.json
phpunit.xml
readme.md
server.php
webpack.mix.js
```

Различные инструменты сборки в Laravel до версии 5.4

В проектах, созданных до Laravel 5.4, вы можете увидеть файл gulpfile.js вместо webpack.mix.js. Это означает, что проект использует Laravel Elixir (http://bit.ly/2JCToYp) вместо Laravel Mix (http://bit.ly/2U4X09P).

Кратко ознакомимся с ними.

Каталоги

Корневой каталог по умолчанию содержит следующие папки.

- `app` — здесь размещается основная часть вашего приложения — модели, контроллеры, команды и PHP-код домена.
- `bootstrap` — содержит файлы, которые Laravel использует для загрузки при каждом запуске.
- `config` — здесь находятся все конфигурационные файлы.
- `database` — содержит миграции баз данных, сидеры и фабрики.

- `public` — каталог, на который указывает сервер при обслуживании сайта. Содержит файл `index.php` — фронтальный контроллер, который запускает процесс начальной загрузки и маршрутизирует все запросы. Здесь также размещаются все публичные файлы: изображения, таблицы стилей, сценарии или загружаемые файлы.
- `resources` — здесь находятся файлы для других сценариев: представления, языковые файлы, а также (опционально) файлы исходного кода CSS/Sass/Less и файлы исходного кода JavaScript.
- `routes` — содержит все определения маршрутов как для HTTP-маршрутов, так и для «консольных маршрутов» или команд Artisan.
- `storage` — здесь находятся кэши, логи и скомпилированные системные файлы.
- `tests` — хранит модульные и интеграционные тесты.
- `vendor` — сюда устанавливаются зависимости менеджера пакетов Composer. Этот каталог игнорируется системой управления версиями Git (помечается как не контролируемый ею) в силу того, что действия Composer являются составной частью процесса развертывания на любых удаленных серверах.

Отдельные файлы

Корневой каталог также содержит следующие файлы.

- `.editorconfig` — инструкции для вашей среды разработки/текстового редактора в отношении предписываемых фреймворком стандартов кодирования (например, о размере отступов, кодировке и о том, следует ли обрезать конечные пробелы). Этот файл есть в любом приложении Laravel версии 5.5 или более новой.
- `.env` и `.env.example` — задают переменные среды (предположительно являются разными в разных средах и потому не регистрируются в системе управления версиями). `.env.example` — это шаблон, который дублируется каждой конкретной средой для создания собственного файла `.env`, игнорируемого системой управления версиями Git.
- `.gitignore` и `.gitattributes` — конфигурационные файлы системы управления версиями Git.
- `artisan` — позволяет запускать команды Artisan (см. главу 8) из командной строки.
- `composer.json` и `composer.lock` — конфигурационные файлы для Composer, при этом файл `composer.json` может редактироваться пользователем, а файл `composer.lock` — нет. Содержат некоторые базовые сведения о проекте, а также определяют его PHP-зависимости.
- `package.json` — файл, аналогичный `composer.json`, но предназначенный для ресурсов клиентской части и зависимостей системы сборки. Содержит указания

для менеджера пакетов NPM в отношении того, какие зависимости JavaScript следует подгрузить.

- `phpunit.xml` — конфигурационный файл для PHPUnit — инструмента, который Laravel использует для тестирования системы.
- `readme.md` — файл Markdown, содержащий базовые сведения о фреймворке. Вы его не увидите, если используете установщик Laravel.
- `server.php` — резервный сервер, позволяющий выполнять предварительный просмотр приложения Laravel даже маломощным серверам.
- `webpack.mix.js` — конфигурационный (опциональный) файл для Mix. Если вы используете Elixir, то вместо этого файла увидите файл `gulpfile.js`. Эти файлы содержат указания для системы сборки в отношении способа компиляции и обработки ресурсов клиентской части.

Конфигурация

Основные настройки вашего приложения Laravel — настройки подключения к базе данных, параметры обработки очередей, электронной почты и т. д. — содержатся в файлах папки `config`. Каждый из этих файлов возвращает массив языка PHP, доступ к элементам которого осуществляется по конфигурационному ключу, состоящему из имени файла и всех ключей-потомков, разделенных точками (`.`).

Так, вы можете создать файл `config/services.php`, содержащий код следующего вида:

```
// config/services.php
<?php
return [
    'sparkpost' => [
        'secret' => 'abcdefg',
    ],
];
```

А затем получать доступ к этой переменной конфигурации с помощью выражения `config('services.sparkpost.secret')`.

Любые переменные конфигурации, которые должны быть разными у разных сред (и, следовательно, игнорироваться системой управления версиями), нужно перенести из этой папки в файлы `.env`. Допустим, вы хотите использовать для каждой среды разные ключи API Bugsnag. В таком случае настройте файл конфигурации так, чтобы он извлекал их из файла `.env`:

```
// config/services.php
<?php
return [
    'bugsnag' => [
        'api_key' => env('BUGSNAG_API_KEY'),
    ],
];
```

Хелпер `env()` извлекает значение из файла `.env`, содержащего нужный вам ключ. Соответственно, следует добавить его в файл `.env` (хранящий настройки данной среды) и `.env.example` (являющийся шаблоном для всех сред):

```
# В .env
BUGSNAG_API_KEY=oinfp9813410942

# В .env.example
BUGSNAG_API_KEY=
```

Файл `.env` уже будет содержать довольно много специфических для среды переменных с необходимой фреймворку информацией: сведениями об используемом драйвере электронной почты или настройках базы данных.

Использование функции env() вне файлов конфигурации

При вызове функции env() за пределами конфигурационных файлов могут быть недоступны некоторые возможности фреймворка Laravel, включая ряд функций кэширования и оптимизации.

Наилучший способ получения переменных среды — присвоение всех специфичных для среды значений элементам конфигурации. Считайте переменные среды в эти элементы конфигурации, а затем ссылайтесь на переменные конфигурации в любом месте приложения:

```
// config/services.php
return [
    'bugsnag' => [
        'key' => env('BUGSNAG_API_KEY'),
    ],
];

// В контроллере или где-либо еще
$bugsnag = new BugsLnag(config('services.bugsnag.key'));
```

Файл .env. Кратко рассмотрим содержимое файла `.env` по умолчанию. Список ключей может немного варьироваться в зависимости от используемой версии приложения, но в случае Laravel 5.8 он выглядит так, как показано в примере 2.1.

Пример 2.1. Переменные среды по умолчанию в Laravel 5.8

```
APP_NAME=Laravel
APP_ENV=local
APP_KEY=
APP_DEBUG=true
APP_URL=http://localhost

LOG_CHANNEL=stack

DB_CONNECTION=mysql
```

```
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=homestead
DB_USERNAME=homestead
DB_PASSWORD=secret

BROADCAST_DRIVER=log
CACHE_DRIVER=file
QUEUE_CONNECTION=sync
SESSION_DRIVER=file
SESSION_LIFETIME=120

REDIS_HOST=127.0.0.1
REDIS_PASSWORD=null
REDIS_PORT=6379

MAIL_DRIVER=smtp
MAIL_HOST=smtp.mailtrap.io
MAIL_PORT=2525
MAIL_USERNAME=null
MAIL_PASSWORD=null
MAIL_ENCRYPTION=null

AWS_ACCESS_KEY_ID=
AWS_SECRET_ACCESS_KEY=

PUSHER_APP_ID=
PUSHER_APP_KEY=
PUSHER_APP_SECRET=
PUSHER_APP_CLUSTER=mt1

MIX_PUSHER_APP_KEY="${PUSHER_APP_KEY}"
MIX_PUSHER_APP_CLUSTER="${PUSHER_APP_CLUSTER}"
```

Я не буду описывать назначение всех ключей, поскольку многие из них представляют собой группы аутентификационных данных для различных сервисов (Pusher, Redis, DB, Mail). В то же время стоит обратить внимание на две важные переменные среды, о которых вы должны знать.

- `APP_KEY` — случайно сгенерированная строка для шифрования данных. Если этот ключ будет пустым, вы можете столкнуться с ошибкой «Не указан ключ шифрования приложения». Тогда запустите команду `php artisan key:generate`, и Laravel сгенерирует ключ для вас.
- `APP_DEBUG` — логическое значение, определяющее, должны ли пользователи этого экземпляра вашего приложения видеть ошибки отладки, — хорошо подходит для локальных и промежуточных сред и абсолютно не подходит для эксплуатационной.

Остальным не связанным с аутентификацией параметрам (`BROADCAST_DRIVER`, `QUEUE_CONNECTION` и т. д.) присваиваются значения по умолчанию, обеспечивающие минимально возможную зависимость от внешних сервисов. Рекомендуется начинающим разработчикам.

Для большинства проектов после запуска приложения нужно изменить параметры конфигурации базы данных. Поскольку я использую Laravel Valet, то присваиваю параметру `DB_DATABASE` имя своего проекта, параметру `DB_USERNAME` — значение `root`, а параметру `DB_PASSWORD` — пустую строку:

```
DB_DATABASE=myProject
DB_USERNAME=root
DB_PASSWORD=
```

Затем я создаю базу данных с таким же, как у проекта, именем в том клиенте MySQL, который предпочитаю использовать. Готово.

Завершение настройки

На этом подготовку к работе пустого проекта Laravel можно считать завершенной. Остается лишь запустить команду `git init`, зарегистрировать пустые файлы с помощью команд `git add` и `git commit` — и можно приступать к кодированию! Если вы используете Valet, то можете сразу увидеть, как фактически выглядит ваш сайт в браузере, выполнив следующие команды:

```
laravel new myProject && cd myProject && valet open
```

Начиная новый проект, я выполняю следующие команды:

```
laravel new myProject
cd myProject
git init
git add .
git commit -m "Initial commit"
```

Поскольку я размещаю свои сайты в папке `~/Sites`, выбранной в качестве основного каталога среды Valet, после выполнения этих команд в браузере сразу же доступно имя `myProject.test`. Затем мне остается отредактировать файл `.env` так, чтобы он указывал на конкретную базу данных, добавить ее в свое приложение для работы с MySQL, и я готов кодировать! А если вы решите использовать Lambo, то все будет сделано автоматически.

Тестирование

В последующих главах в заключительном разделе «Тестирование» я буду показывать, как следует писать тесты для рассмотренных в главе функций. Поскольку в этой главе мы не рассматривали какие-либо тестируемые возможности, просто немного поговорим о тестировании (подробнее о написании и запуске тестов в Laravel вы прочитаете в главе 12).

По умолчанию фреймворк добавляет PHPUnit в качестве зависимости и настроен на запуск тестов, содержащихся в любом файле, который размещен в каталоге `tests` и имеет окончание `Test.php` (например, `tests/UserTest.php`).

Таким образом, самый легкий способ написания тестов состоит в том, чтобы создать в каталоге `tests` файл с именем, оканчивающимся на `Test.php`. И самый простой способ запуска — выполнить в командной строке команду `./vendor/bin/phpunit` (находясь в корневой папке проекта).

Если для каких-либо тестов требуется доступ к базе данных, то тесты следует запускать на том компьютере, где размещена ваша база данных, — поэтому, если вы размещаете свою базу данных в Vagrant, не забудьте подключиться к Vagrant-box по протоколу ssh и запустить свои тесты из него. Об этом и многом другом подробно написано в главе 12.

Следует отметить, что если вы читаете эту книгу впервые, то в разделах, посвященных тестированию, встретите незнакомый вам синтаксис и описание новых возможностей тестирования. Если вы не сможете разобраться в коде какого-либо из этих разделов, просто пропустите его и вернитесь уже после прочтения главы о тестировании.

Резюме

Поскольку Laravel является PHP-фреймворком, его очень просто обслуживать локально. Он также предоставляет два инструмента для управления вашей локальной разработкой: более простой — Valet, который использует для загрузки зависимостей локальную машину, и предварительно настроенную конфигурацию Vagrant под названием Homestead. Laravel применяет менеджер пакетов Composer и может устанавливаться с его помощью. По умолчанию загружается ряд папок и файлов, отражающих соглашения фреймворка и его взаимосвязи с другими инструментами с открытым исходным кодом.