

Оглавление

| | |
|--|----|
| Предисловие | 20 |
| О чём эта книга..... | 20 |
| Для кого предназначена книга..... | 21 |
| Структура издания..... | 21 |
| О втором издании..... | 21 |
| Условные обозначения..... | 22 |
| Благодарности | 23 |
| Первое издание | 23 |
| Второе издание | 24 |
| От издательства | 25 |
| Глава 1. Зачем использовать Laravel | 26 |
| Для чего нужен фреймворк..... | 26 |
| «Все своими руками»..... | 27 |
| Согласованность и гибкость | 27 |
| Краткий экскурс в историю веб- и PHP-фреймворков | 27 |
| Ruby on Rails..... | 28 |
| Бум PHP-фреймворков | 28 |
| Преимущества и недостатки CodeIgniter | 28 |
| Laravel 1, 2 и 3..... | 29 |
| Laravel 4 | 29 |
| Laravel 5 | 29 |
| Чем уникален Laravel | 30 |
| Философия Laravel | 30 |
| Как Laravel делает разработчиков счастливее | 31 |
| Сообщество Laravel | 32 |
| Как работает Laravel | 32 |
| Почему стоит выбрать Laravel | 35 |

| | |
|---|----|
| Глава 2. Настройка среды разработки для использования Laravel..... | 36 |
| Системные требования..... | 36 |
| Composer..... | 37 |
| Локальные среды разработки..... | 37 |
| Laravel Valet..... | 37 |
| Laravel Homestead..... | 38 |
| Создание нового проекта Laravel..... | 38 |
| Установка Laravel с помощью установщика Laravel | 39 |
| Установка Laravel с помощью функции create-project менеджера пакетов Composer | 39 |
| Lambo: улучшенный вариант команды laravel new..... | 39 |
| Структура каталогов Laravel | 40 |
| Каталоги..... | 40 |
| Отдельные файлы..... | 41 |
| Конфигурация | 42 |
| Завершение настройки | 45 |
| Тестирование | 45 |
| Резюме | 46 |
| Глава 3. Маршрутизация и контроллеры..... | 47 |
| Краткое введение в MVC, команды HTTP и REST | 47 |
| Что такое MVC | 47 |
| HTTP-команды..... | 48 |
| Что такое REST | 49 |
| Определения маршрутов | 50 |
| Команды маршрутов..... | 51 |
| Обработка маршрутов | 52 |
| Параметры маршрутов..... | 53 |
| Имена маршрутов | 54 |
| Группы маршрутов..... | 56 |
| Middleware..... | 57 |
| Префиксы путей..... | 59 |
| Запасные маршруты..... | 60 |
| Поддоменная маршрутизация..... | 60 |
| Префиксы пространства имен..... | 61 |
| Префиксы имен | 61 |
| Подписанные маршруты..... | 62 |
| Подписание маршрута | 62 |
| Изменение маршрутов для разрешения подписанных ссылок..... | 63 |

| | |
|--|-----------|
| Представления (views) | 64 |
| Прямой возврат простых маршрутов с помощью метода Route::view() | 65 |
| Общий доступ представлений к переменным с использованием компоновщиков представлений | 65 |
| Контроллеры (controllers) | 65 |
| Получение ввода пользователя | 68 |
| Внедрение зависимостей в контроллеры | 69 |
| Контроллеры ресурсов | 71 |
| Контроллеры ресурсов API | 72 |
| Контроллеры одиночного действия | 72 |
| Привязка модели маршрута | 73 |
| Неявная привязка модели маршрута | 73 |
| Пользовательская привязка модели маршрута | 74 |
| Кэширование маршрутов | 75 |
| Подмена метода формы | 75 |
| HTTP-команды в Laravel | 76 |
| Подмена HTTP-метода в HTML-формах | 76 |
| Защита CSRF | 76 |
| Перенаправления | 78 |
| redirect()->to() | 79 |
| redirect()->route() | 79 |
| redirect()->back() | 80 |
| Другие методы перенаправления | 80 |
| redirect()->with() | 81 |
| Отмена запроса | 82 |
| Пользовательские ответы | 83 |
| response()->make() | 83 |
| response()->json() и ->jsonp() | 83 |
| response()->download(), ->streamDownload() и ->file() | 83 |
| Тестирование | 84 |
| Резюме | 85 |
| Глава 4. Движок шаблонов Blade | 86 |
| Отображение данных | 87 |
| Управляющие структуры | 88 |
| Условные конструкции | 88 |
| Циклы | 89 |
| Наследование шаблонов | 90 |
| Определение разделов страницы с помощью директив @section/@show и @yield | 90 |
| Включение составляющих представления | 93 |

| | |
|---|------------|
| Использование стеков | 94 |
| Использование компонентов и слотов | 96 |
| Компоновщики представлений и внедрение сервисов | 98 |
| Привязка данных к представлениям с использованием компоновщиков представлений..... | 98 |
| Внедрение сервиса Blade | 101 |
| Пользовательские директивы Blade | 102 |
| Параметры пользовательских директив Blade | 104 |
| Пример: применение пользовательских директив Blade для многоклиентских приложений | 104 |
| Упрощенные пользовательские директивы для операторов if | 105 |
| Тестирование | 106 |
| Резюме | 107 |
| Глава 5. Базы данных и Eloquent | 108 |
| Конфигурация | 108 |
| Подключения базы данных | 109 |
| Другие параметры конфигурации базы данных | 110 |
| Миграции | 110 |
| Определение миграций..... | 111 |
| Запуск миграций | 117 |
| Наполнение базы данными | 118 |
| Создание сидера | 119 |
| Фабрики моделей..... | 120 |
| Генератор запросов | 124 |
| Стандартное использование фасада DB | 125 |
| Чистый SQL..... | 126 |
| Выстраивание цепочки с генератором запросов..... | 127 |
| Транзакции..... | 135 |
| Введение в Eloquent..... | 136 |
| Создание и определение моделей Eloquent | 138 |
| Получение данных с помощью Eloquent | 139 |
| Вставки и обновления с помощью Eloquent | 141 |
| Удаление с помощью Eloquent | 145 |
| Области видимости | 147 |
| Настройка взаимодействия полей с аксессорами, мутаторами и приведением атрибутов..... | 150 |
| Коллекции Eloquent..... | 154 |
| Сериализация Eloquent | 156 |
| Связи в Eloquent | 158 |
| Обновление меток времени родительской записи дочерними записями | 170 |

| | |
|---|---------|
| События Eloquent..... | 172 |
| Тестирование | 174 |
| Резюме..... | 176 |
| Глава 6. Компоненты для клиентской части..... | 177 |
| Laravel Mix..... | 177 |
| Структура каталога Mix | 179 |
| Запуск Mix..... | 179 |
| Что предоставляет Mix | 180 |
| Предустановки клиентской части и генерация кода аутентификации | 186 |
| Предустановки клиентской части..... | 187 |
| Генерация кода аутентификации | 188 |
| Разбивка на страницы..... | 188 |
| Разбивка на страницы результатов из базы данных..... | 188 |
| Создание разбивщиков страниц вручную..... | 189 |
| Пакеты сообщений..... | 190 |
| Строковые хелперы, множественность и локализация..... | 192 |
| Строковые хелперы и множественность | 192 |
| Локализация | 193 |
| Тестирование | 196 |
| Тестирование пакетов сообщений и ошибок | 196 |
| Перевод и локализация..... | 197 |
| Резюме | 197 |
| Глава 7. Получение и обработка пользовательских данных | 198 |
| Внедрение объекта запроса..... | 198 |
| \$request->all() | 199 |
| \$request->except() и \$request->only()..... | 199 |
| \$request->has() | 200 |
| \$request->input() | 200 |
| \$request->method() и ->isMethod() | 201 |
| Ввод массива..... | 201 |
| Ввод JSON (и \$request->json()) | 201 |
| Маршрутные данные | 203 |
| Из Request | 203 |
| Из параметров маршрута | 203 |
| Загруженные файлы..... | 203 |
| Валидация | 206 |
| Метод validate() объекта Request..... | 206 |
| Ручная валидация | 208 |

| | |
|---|------------|
| Объекты пользовательских правил..... | 208 |
| Отображение валидационных сообщений | 209 |
| Запросы формы..... | 209 |
| Создание запроса формы | 210 |
| Использование запроса формы..... | 211 |
| Модель массового назначения Eloquent | 212 |
| Синтаксис {{ и>{!!..... | 213 |
| Тестирование | 213 |
| Резюме | 215 |
| Глава 8. Интерфейсы Artisan и Tinker | 216 |
| Введение в интерфейс Artisan..... | 216 |
| Основные команды Artisan | 217 |
| Параметры..... | 217 |
| Сгруппированные команды | 218 |
| Написание пользовательских команд Artisan..... | 220 |
| Пример команды | 222 |
| Аргументы и параметры | 223 |
| Использование ввода..... | 225 |
| Приглашения..... | 226 |
| Вывод | 227 |
| Команды на основе замыканий | 229 |
| Вызов команд Artisan в нормальном коде | 229 |
| Tinker | 230 |
| Сервер дампа Laravel..... | 231 |
| Тестирование | 232 |
| Резюме | 233 |
| Глава 9. Аутентификация и авторизация пользователей..... | 234 |
| Модель User и миграция..... | 235 |
| Использование глобального хелпера auth() и фасада Auth | 238 |
| Контроллеры аутентификации..... | 239 |
| Контроллер RegisterController..... | 239 |
| Контроллер LoginController..... | 240 |
| Контроллер ResetPasswordController | 242 |
| Контроллер ForgotPasswordController | 242 |
| Контроллер VerificationController | 243 |
| Метод Auth::routes()..... | 243 |
| Каркас аутентификации..... | 244 |
| Токен «Запомнить меня»..... | 245 |

| | |
|--|-----|
| Выполнение вручную аутентификации пользователей | 246 |
| Выполнение вручную выхода пользователя из системы | 247 |
| auth | 248 |
| Верификация адресов электронной почты | 249 |
| Blade-директивы для аутентификации..... | 249 |
| Гарды..... | 250 |
| Указание другого гарда по умолчанию | 250 |
| Использование других гардов без изменения базового | 251 |
| Добавление нового гарда..... | 251 |
| Гарды на основе замыкания запроса | 252 |
| Создание собственного провайдера пользователей..... | 252 |
| Собственные провайдеры пользователей для нереляционных баз данных | 253 |
| События аутентификации..... | 253 |
| Система авторизации (список управления доступом) и роли | 254 |
| Определение правил авторизации..... | 255 |
| Фасад Gate (и его внедрение)..... | 256 |
| Ресурсы гейтов..... | 256 |
| Authorize..... | 257 |
| Авторизация внутри контроллера | 257 |
| Проверка с помощью экземпляра класса User | 259 |
| Проверки с помощью Blade-директив..... | 260 |
| Перехват проверок..... | 260 |
| Политики | 261 |
| Тестирование | 263 |
| Резюме | 266 |
| Глава 10. Запросы, ответы и middleware | 267 |
| Жизненный цикл запроса в Laravel | 267 |
| Начальная загрузка приложения | 267 |
| Сервис-провайдеры..... | 269 |
| Объект Request | 271 |
| Получение объекта Request в Laravel | 271 |
| Получение основной информации о запросе..... | 272 |
| Объект Response..... | 276 |
| Использование и создание объектов Response в контроллерах | 276 |
| Специализированные типы ответов..... | 277 |
| Laravel и middleware | 283 |
| Вводная информация о middleware | 283 |
| Создание собственного middleware | 284 |

| | |
|--|------------|
| Привязка middleware | 286 |
| Передача параметров middleware..... | 289 |
| Доверенные прокси-серверы..... | 290 |
| Тестирование | 291 |
| Резюме | 292 |
| Глава 11. Контейнер | 293 |
| Вводная информация о внедрении зависимостей | 293 |
| Внедрение зависимостей и Laravel..... | 295 |
| Глобальный хелпер app()..... | 295 |
| Как осуществляется привязка к контейнеру | 296 |
| Привязка классов к контейнеру | 297 |
| Привязка к замыканию..... | 298 |
| Привязка одиночек, псевдонимов и экземпляров..... | 299 |
| Привязка конкретного экземпляра к интерфейсу | 300 |
| Контекстная привязка..... | 300 |
| Внедрение в конструктор в файлах Laravel | 301 |
| Внедрение через метод | 302 |
| Фасады и контейнер | 303 |
| Как работают фасады..... | 304 |
| Фасады реального времени | 305 |
| Сервис-провайдеры | 306 |
| Тестирование | 306 |
| Резюме | 307 |
| Глава 12. Тестирование | 308 |
| Основы тестирования | 309 |
| Именование тестов..... | 313 |
| Среда тестирования | 314 |
| Трейты тестирования..... | 314 |
| RefreshDatabase..... | 315 |
| WithoutMiddleware..... | 315 |
| DatabaseMigrations | 315 |
| DatabaseTransactions..... | 315 |
| Простые модульные тесты | 316 |
| Как осуществляется тестирование приложений | 317 |
| HTTP-тесты..... | 318 |
| Тестирование простых страниц с помощью вызова \$this->get() | |
| и других HTTP-вызовов..... | 318 |
| Тестирование API на базе JSON с помощью вызова \$this->getJson() | |
| и других HTTP-вызовов на базе JSON | 319 |

| | |
|--|------------|
| Утверждения в отношении объекта \$response | 320 |
| Аутентификация ответов | 322 |
| Ряд других настроек HTTP-тестов..... | 323 |
| Обработка исключений в тестах приложений..... | 323 |
| Тесты базы данных..... | 324 |
| Использование фабрик моделей в тестах..... | 325 |
| Заполнение начальными данными в тестах | 325 |
| Тестирование других систем Laravel | 325 |
| Подделка событий | 326 |
| Подделка фасадов Bus и Queue..... | 327 |
| Подделка фасада Mail..... | 328 |
| Подделка фасада Notification..... | 329 |
| Подделка фасада Storage | 330 |
| Имитирование | 331 |
| Вводная информация об имитировании | 331 |
| Вводная информация о Mockery | 331 |
| Подделка других фасадов | 334 |
| Тестирование команд Artisan | 335 |
| Браузерные тесты | 336 |
| Выбор инструмента..... | 337 |
| Тестирование с использованием Dusk | 338 |
| Резюме | 349 |
| Глава 13. Создание API | 350 |
| Базовые сведения о REST-подобных API на базе JSON | 350 |
| Организация контроллеров и возвращаемые JSON-сообщения | 352 |
| Чтение и отправка заголовков | 355 |
| Отправка заголовков ответа в Laravel | 356 |
| Чтение заголовков запроса в Laravel | 356 |
| Разбивка на страницы в Eloquent | 356 |
| Сортировка и фильтрация | 358 |
| Сортировка результатов API | 359 |
| Фильтрация результатов API..... | 360 |
| Преобразование результатов..... | 361 |
| Создание собственного преобразователя | 362 |
| Вложение связей пользовательских преобразователей | 363 |
| Ресурсы API | 365 |
| Создание класса ресурса | 365 |
| Коллекции ресурсов..... | 366 |
| Вложение связей | 368 |

| | |
|--|------------|
| Применение разбивки на страницы к ресурсам API..... | 369 |
| Условное применение атрибутов | 370 |
| Другие настройки для ресурсов API..... | 370 |
| Аутентификация API с помощью Laravel Passport | 370 |
| Вводная информация о OAuth 2.0 | 370 |
| Установка пакета Passport | 371 |
| API пакета Passport | 373 |
| Типы допуска, предлагаемые пакетом Passport | 373 |
| Управление клиентами и токенами с помощью API пакета Passport и компонентов Vue | 382 |
| Области видимости пакета Passport | 384 |
| Развертывание пакета Passport | 386 |
| Аутентификация с помощью токенов API | 386 |
| Настройка ответов с кодом 404 | 387 |
| Тестирование | 388 |
| Резюме | 389 |
| Глава 14. Сохранение и извлечение данных | 390 |
| Локальные и облачные файловые менеджеры..... | 390 |
| Настройка доступа к файлам | 390 |
| Использование фасада Storage..... | 392 |
| Добавление дополнительных провайдеров из пакета Flysystem | 393 |
| Базовые способы загрузки файлов на сервер и манипулирования файлами | 393 |
| Простые способы скачивания файлов | 395 |
| Сессии..... | 395 |
| Получение доступа к сессии | 395 |
| Методы, доступные в экземплярах сессий..... | 396 |
| Флеш-память сессии..... | 398 |
| Кэш | 398 |
| Получение доступа к кэшу | 399 |
| Методы, доступные в экземплярах кэшей | 400 |
| Cookie-файлы..... | 401 |
| Cookie-файлы в Laravel..... | 401 |
| Получение доступа к cookie-файлам..... | 401 |
| Логирование..... | 404 |
| Когда и зачем следует выполнять логирование | 405 |
| Внесение записей в логи..... | 405 |
| Каналы логирования..... | 406 |
| Полнотекстовый поиск с использованием Laravel Scout..... | 409 |
| Установка пакета Scout | 409 |
| Пометка модели для индексирования | 410 |

| | |
|--|-----|
| Поиск по вашему индексу | 410 |
| Очереди и Scout..... | 410 |
| Выполнение операций без индексирования..... | 411 |
| Условное индексирование моделей | 411 |
| Запуск индексирования вручную с помощью кода | 411 |
| Запуск индексирования вручную с помощью интерфейса командной строки | 412 |
| Тестирование | 412 |
| Сохранение файлов..... | 412 |
| Сессия | 414 |
| Кэш..... | 415 |
| Cookie-файлы | 415 |
| Логирование | 416 |
| Scout | 417 |
| Резюме..... | 417 |
| Глава 15. Почта и уведомления | 418 |
| Почта..... | 418 |
| Классическая электронная почта | 419 |
| Простейший способ использования отправлений | 419 |
| Шаблоны писем..... | 421 |
| Методы, доступные в build() | 422 |
| Прикрепленные файлы и встроенные изображения | 423 |
| Markdown-отправления..... | 424 |
| Визуализация отправлений в браузере | 426 |
| Очереди | 426 |
| Локальная разработка | 427 |
| Уведомления | 428 |
| Определение метода via() для уведомляемых объектов | 431 |
| Отправка уведомлений | 432 |
| Помещение уведомлений в очередь | 432 |
| Предлагаемые по умолчанию типы уведомлений | 433 |
| Тестирование | 437 |
| Электронная почта | 437 |
| Уведомления | 438 |
| Резюме..... | 438 |
| Глава 16. Очереди, задания, события, трансляция и планировщик | 439 |
| Очереди..... | 439 |
| Зачем нужны очереди..... | 440 |
| Базовая конфигурация очередей..... | 440 |

| | |
|---|------------|
| Задания в очереди..... | 441 |
| Запуск обработчика очередей..... | 444 |
| Обработка ошибок..... | 445 |
| Управление очередью..... | 447 |
| Очереди для поддержки других функций..... | 448 |
| Laravel Horizon | 448 |
| События | 449 |
| Запуск события..... | 449 |
| Прослушивание события | 451 |
| Трансляция событий посредством веб-сокетов и Laravel Echo | 454 |
| Конфигурация и настройка..... | 455 |
| Трансляция события..... | 455 |
| Получение сообщения..... | 458 |
| Продвинутые инструменты трансляции | 460 |
| Laravel Echo (сторона JavaScript-кода)..... | 464 |
| Планировщик..... | 469 |
| Доступные типы задач..... | 470 |
| Доступные временные интервалы | 470 |
| Определение часовых поясов для запланированных задач | 472 |
| Блокирование и наложение..... | 472 |
| Обработка выходных данных задачи | 473 |
| Перехват задач | 474 |
| Тестирование | 474 |
| Резюме | 476 |
| Глава 17. Хелперы и коллекции | 477 |
| Хелперы | 477 |
| Массивы | 477 |
| Строки..... | 479 |
| Пути приложения | 481 |
| URL-адреса | 482 |
| Прочее..... | 484 |
| Коллекции..... | 486 |
| Базовые сведения..... | 487 |
| Некоторые методы..... | 489 |
| Резюме | 493 |
| Глава 18. Экосистема инструментов Laravel | 494 |
| Инструменты, рассмотренные в книге..... | 494 |
| Valet | 494 |
| Homestead | 494 |

| | |
|---|-----|
| Установщик Laravel | 495 |
| Mix..... | 495 |
| Dusk | 495 |
| Passport | 495 |
| Horizon..... | 495 |
| Echo..... | 496 |
| Инструменты, не рассмотренные в книге | 496 |
| Forge | 496 |
| Envoyer | 496 |
| Cashier | 497 |
| Socialite | 498 |
| Nova..... | 498 |
| Spark | 498 |
| Lumen | 498 |
| Envoy | 499 |
| Telescope..... | 499 |
| Другие ресурсы | 499 |
| Глоссарий..... | 501 |
| Об авторе..... | 511 |
| Об обложке..... | 512 |

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, обслуживающий 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` (хранищий 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 минимальную 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 среди 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.