



Фреймворк Laravel: быстрый старт для PHP-разработчика




PHP разработчик
Давыдов Роман



Что такое Framework

И чем он нам так сильно поможет в разработке?

- Набор инструментов, библиотек и модулей
- Методология



Преимущества использования Framework

- Framework содержит основные компоненты.
- Использование методологии.
- Код структурирован, вследствие чего понятен другим программистам.
- Простая расширяемость.
- Обновления и доработки, сообщество.

Установка Laravel через Composer



- Устанавливаем установщик Laravel:
`composer global require "laravel/installer"`
- Делаем alias для установщика (Linux):
`alias laravel=~/.composer/vendor/bin/laravel`
- Устанавливаем Laravel:
`laravel new blog`



Настройка Laravel

- Основные настройки Laravel хранятся в файле «среды» `.env`
- Остальные настройки можно найти в папке `config`.



Консольный помощник Artisan

Artisan – это незаменимый помощник при разработке на Laravel:

- Запуск/«откат» миграций проекта (структура БД)
- Автогенерация классов и модулей
- Вывод информации о версии, среде и маршрутах.
- Обновление кеша приложения

И многое другое..



Laravel & MVC

Framework Laravel – типичный представитель MVC-паттерна:
В нём присутствует разделение на модели, контроллеры и представления.

Модели: `app/`

Контроллеры: `app/Http/Controllers/`

Представления: `resources/views/`




С ЧЕГО НАЧАТЬ?

Вторым параметром в маршрут передается его действие:
функция замыкание или передача управления в
контроллер.

Внутри роутера уже есть как пример самый простой вариант:

```
Route::get('/', function() {  
    return view('welcome');  
});
```

Все запросы от пользователей попадают в маршрутизатор

Все запросы должны быть зарегистрированы в
`routes/web.php`

При этом класс `Route` позволяет регистрировать любые
варианты запросов.

`Route::get()` – регистрация маршрута с методом GET

`Route::post()` – регистрация маршрута с методом POST

При этом первый параметр – это `uri`, а второй – действие.

Поддерживаются также любые другие методы, а также
параметры в запросах.



Миграции

Зачем нужны:

- Быстро создать структуру БД
- Иметь возможность хранить структуру внутри кода
- Обновлять структуру БД

Что это?

- Классы, содержащие описание структуры БД
- Каждый класс содержит 2 зеркальных метода: `up&down`
- Миграции хранятся в `database/migrations/`



Создание и запуск миграций

`artisan make:migration название_миграции`

поддерживает также флаг `-table=название_таблицы`

Запуск миграций:

`artisan migrate`

Откат миграций:

`artisan migrate:rollback`



Контроллеры

Контроллеры – это классы, содержащие логику приложения.

Методы контроллера – они же «экшены» – функции, которые выполняются при определенных запросах пользователей.

Один запрос – один «экшин». То есть каждый метод отвечает за свою функцию.



Создание контроллера

`artisan make:controller`

- Создается класс внутри папки `app/Http/Controllers/` который уже наследуется от базового контроллера и имеет нужный namespace

Ресурс-контроллер

```
artisan make:controller --resource --model=User
```

- Создается класс внутри папки `app/Http/Controllers/` который уже наследуется от базового контроллера и имеет нужный namespace а также имеет 7 методов по REST.

Каждая сущность, которую нужно

редактировать/просматривать/удалять/обновлять – это ресурс.

Для таких ресурсов больше всего и подходят ресурс-контроллеры

Пример ресурс-контроллера для posts

Глагол	Ссылка (uri)	Действие (метод)	Маршрут (имя)
GET	/posts	index	posts.index
GET	/posts/create	create	posts.create
POST	/posts	store	posts.store
GET	/posts/{photo}	show	posts.show
GET	/posts/{photo}/edit	edit	posts.edit
PUT/PATCH	/posts/{photo}	update	posts.update
DELETE	/posts/{photo}	destroy	posts.destroy



Ресурс-контроллер в роутере

Для того, чтобы реализовать связку от запросов к ресурс-контроллеру, в роутере нужно зарегистрировать ресурс:

```
Route::resource('posts', 'PostsController');
```




Модель

Модель – это класс, описывающий некоторую сущность.

Модель работает с базой и позволяет нам абстрагироваться от способа хранения этой информации в базе данных.



Laravel: Модель Eloquent

Создание модели:

```
artisan make:model название_модели
```


Автоматически попадают в app/

При этом нужно придерживаться правила:

Название модели в единственном числе CamelCase

Название таблицы для модели – то же название, но во множественном числе snake_case

Первичный ключ – id а ссылка на внешнюю таблицу (связь) название_сущности_id (user_id, post_id и т.д.)




Дополнительные возможности маршрутизации

Передача управления из роутера в контроллер (метод):

```
Route::get('/', 'PageController@index');
```

Именованное маршрутов (для возможности их вызова из кода или представления):

```
Route::get('/', 'PageController@index')->name('index-page');
```



Дополнительные возможности маршрутизации

Группировка свойств маршрутов:

```
Route::group(['middleware' => 'auth'], function () {  
    ...  
});
```

Параметры в маршрутах:

```
Route::get('{category}/{post}', 'PageController@post');
```



Связи между сущностями

Для того, чтобы через модель реализовать связь один ко многим (или один к одному) есть несколько методов внутри Eloquent:

`belongsTo ()`

`belongsToMany()`

`hasOne()`

`hasMany()`



Связи между сущностями

Есть 2 таблицы: categories и posts

У таблицы posts есть поле category_id, которое указывает, к какой категории принадлежит пост:

Post будет иметь метод, возвращающий категорию через
`$this->belongsTo('App\Category');`

Category будет иметь доступ ко всем связанным постам
через

`$this->hasMany('App\Post')`



Админка из «коробки» Laravel

- Генерация авторизации одной строчкой:

`artisan make:auth`



Представления

В Laravel есть встроенный шаблонизатор: Blade

Blade позволяет:

Работать

- Отображать переменные
- Работать с циклами, условными конструкциями и вложениями

Все шаблоны хранятся в `resources/views`

Вызов представления

Вызов шаблона происходит в конце метода контроллера через хелпер `view`.

Первым параметром является название, вторым – параметры.

```
return view('posts', ['list' => $posts]);
```

-будет вызвано представление по пути

`resources/views/posts.blade.php` в которое будут

переданные данные из переменной `$posts`. Доступны эти данные будут в представлении внутри переменной `$list`.



Конструкции, доступные в Blade

Отображение переменной:

```
{{ $var }}
```

Циклы:

```
@foreach()
```

```
@for()
```

Условия

```
@if()
```

```
@isset()
```

```
@auth
```



Конструкции, доступные в Blade

Подключение подшаблона:

`@include()`

`@yield.... @extends.... @section`



Работа с запросом

Для того, чтобы работать с запросом (передаваемыми данными) в Laravel есть несколько способов. Одн из них – подключение класса Request как аргумент метода контроллера.

Из этого класса как свойства созданного объекта будут доступны все переменные из запроса.

```
$request->title;
```

Или можно воспользоваться методом `all` и взять массив со всеми данными из запроса:

```
$request->all();
```



Валидация запроса

В Laravel есть встроенный механизм для валидации передаваемых полей:

```
$this->validate($request, [  
    "title" => "required|min:3",  
    "slug" => "required|min:3|unique:categories,slug",  
]);
```

Полный список правил валидации можно посмотреть в официальной документации.



Git репозиторий с кодом из занятия

<https://github.com/warlight/open-lesson-laravel.git>