
THE
LARAVEL
ГАЙД ПО
ВЫЖИВАНИЮ
NULLS

Nulls

Laravel – гайд по выживанию

http://www.litres.ru/pages/biblio_book/?art=69850618
SelfPub; 2023

Аннотация

Почему эта книга? На самом деле, это не совсем книга. Это большепохоже на руководство – руководство, чтобы уберечь вас и других отпревращения в «зомби-разработчиков». Что такое «зомби-разработчик»? Эторазработчик, похожий на нас, который безраздельно занимается созданиемприложений на PHP, повторяя одни и те же задачи снова и снова. Этиповторяющиеся задачи могут быть утомительными и привести красплавлению мозга. Когда это происходит, разработчики повсюдупревращаются в безмозглых зомби, жаждущих крови и порывающихсяубивать.Однако есть лекарство: фреймворк Laravel, разработанный для быстройразработки приложений. Освоив Laravel, вы сможете заново открыть в себестрасть к кодингу и побороть "зомби". Это руководство призвано сохранитьваше здравомыслие, сделав кодинг снова приятным занятием. И да, этоможет спасти жизнь! Освоив основы Laravel, вы сможете уберечь себя и, возможно, другихот превращения в бездумного зомби-разработчика.

Содержание

Nulls
Laravel – гайд по выживанию

1. Начало работы

В этой первой главе мы рассмотрим:

Настройку локальной среды разработки;
– Composer и установку Laravel;
– Структуру папок Laravel

Давайте сделаем это!

Настройка локальной среды разработки

Знание своего окружения – это ключ к выживанию в зомби-апокалипсисе разработчиков. Чтобы создавать исключительные веб-приложения, нам прежде всего необходимо освоить настройку локальной среды разработчика.

Локальная среда, часто называемая "средой разработки", – это когда вы разрабатываете веб-приложение на своем персональном компьютере. Когда вы будете готовы представить свое творение миру, вы перенесете свой код на другой сервер, известный как "производственная среда".

Ниже приведены инструкции по добавлению локальной среды на ваш компьютер.

Локальная разработка на Mac

Если вы являетесь пользователем Mac, то установить локальную среду разработки на вашей машине будет очень просто. Laravel теперь предлагает собственное приложение под названием Herd. Просто загрузите это приложение здесь, установите его, и вы сможете приступить к разработке.

Для пользователей Mac создание локальной среды разработки не составит труда. Laravel предлагает нативное приложение Herd. Просто загрузите приложение на сайте https://herd.laravel.com, установить его и можно приступать к разработке.

Локальная разработка на Windows

Самым простым решением для Windows-машины является использование Laragon, давно полюбившегося сообществу. Однако есть и другие альтернативы, которые стоит рассмотреть:

https://www.mamp.info/en/mamp-pro/windows/
https://www.wampserver.com/ru/
https://www.apachefriends.org/

Локальная разработка на Ubuntu

Если вы работаете на машине Ubuntu, то можете использовать Xampp, а можете установить все приложения по отдельности. Подробнее о том, как это сделать, можно узнать из этой статьи здесь.

Для изучения других методов настройки локального окружения обратитесь к документации по установке Laravel, там вы найдете множество других вариантов, которые могут подойти для вашего случая.

Важно понимать три основных сервиса, которые необходимы для работы типичного локального окружения:

1. Apache или Nginx (веб-сервер для вашего приложения)
2. MySQL (база данных для вашего приложения)
3. PHP (язык сценариев на стороне сервера для вашего приложения).

Как только эти службы будут установлены на вашем компьютере, все готово! В конечном счете, не существует правильного или неправильного способа создания локальной среды разработки. Найдите тот способ, который подходит

именно вам, и вы в кратчайшие сроки сможете создать несколько потрясающих веб-приложений!

Composer и инсталлятор Laravel

Для управления внешними библиотеками или пакетами Laravel использует Composer. Зависимости вашего приложения определяются в файле composer.json.

Composer

Если вы еще не знакомы с концепцией Composer и его функциональными возможностями, не волнуйтесь. Давайте упростим это с помощью забавной аналогии.

Понимание Composer с помощью аналогии с пиццей

Представьте себе Composer как команду для приготовления пиццы. Если бы вы заказывали пиццу с помощью команды, то это выглядело бы примерно так:

$ composer make pizza

По умолчанию эта команда дает нам пиццу "пепперони". Но что, если мы хотим получить пиццу другого типа, например, пиццу для любителей мяса? Мы укажем желаемые начинки следующим образом:

```
{
"toppings" : [
"pepperoni",
"ham",
"bacon",
"beef",
"sausage"
];
}
```

Чтобы настроить заказ пиццы, мы сохраним этот список в файле с именем 'composer.json' в нашем текущем каталоге. Выполняем команду еще раз:

$ composer make pizza

Вуаля! Вместо стандартной пиццы с пепперони у нас теперь есть пицца для любителей мяса!

Composer, по сути, помогает управлять компонентами (или начинками), необходимыми для создания наших приложений.

Composer уже установлен, если вы использовали Herd или Laragon; однако если вам необходимо установить его вручную, вы можете сделать это по адресу https://getcomposer.org/download/.

Инсталлятор Laravel

Laravel installer – это инструмент, позволяющий разработчикам быстро создать новый проект Laravel из командной строки. Чтобы создать новый проект Laravel с помощью программы установки Laravel, выполните следующие действия:

laravel new project-name

Замените project-name на желаемое имя нового проекта. Эта команда создаст каталог с указанным именем и установит в него свежее приложение Laravel.

Установка программы установки Laravel

Если вы уже установили Herd или Laragon, пропустите этот шаг.

После настройки Composer настало время интегрировать Laravel Installer. Для этого выполните следующую команду:

$ composer global require "laravel/installer"

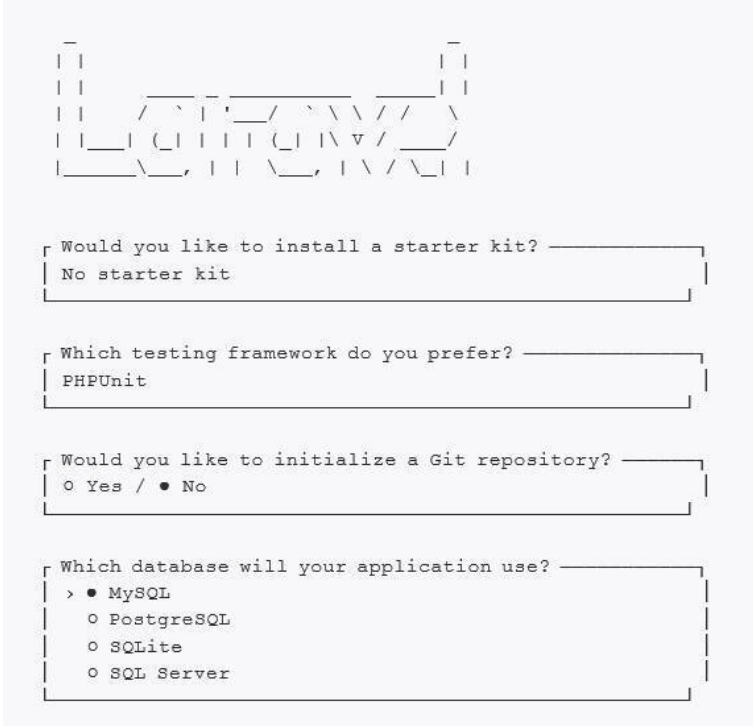
Использование Laravel Installer

Чтобы воспользоваться программой установки Laravel, откройте командную строку и введите следующую команду:

$ laravel new folder_name

При выполнении этой команды вы столкнетесь с несколькими запросами; выберите No starter kit, PHPUnit и No

соответственно. Также на вопрос о базе данных выберите MySQL.



Теперь у вас будет новое приложение в указанной вами папке с именем folder_name. Перейдите в эту папку с помо-

щью команды cd folder_name, а затем запустите:

$ php artisan serve

При этом запускается локальный сервер по адресу http://localhost:8000/. При обращении к этому URL отображается экран приветствия Laravel.

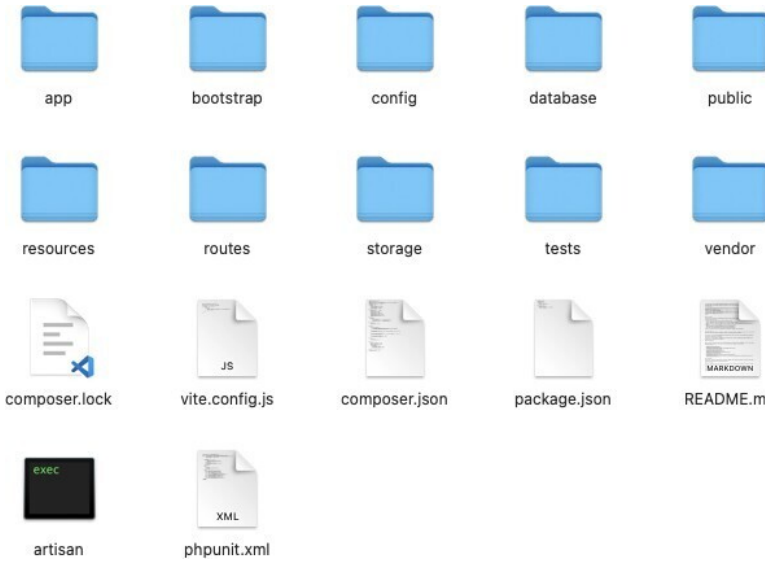
Примечание: Если вы используете Laravel Herd, ваши новые приложения Laravel будут использовать домен .test. Например, по адресу **https://folder_name.test** будет отображаться страница приветствия.

Поздравляем! Теперь вы готовы приступить к созданию замечательных приложений. Простота Laravel гарантирует, что вы сможете поднять новый проект за считанные мгновения.

Прежде чем погрузиться в код, давайте кратко познакомимся со структурой папок Laravel.

Структура папок Laravel

В новом проекте Laravel вы будете иметь следующую структуру кода:



Вы встретите 10 директорий:

1. app
2. bootstrap
3. config
4. база данных
5. public
6. ресурсы

7. маршруты
8. хранилище
9. тесты
10. vendor

Мы не будем подробно рассматривать все папки, однако важно иметь краткое представление о каждой из них.

App

Это каталог, в котором хранится вся логика нашего приложения. В эту папку мы поместим все наши модели, контроллеры, сервисы и многие другие классы.

Bootstrap

Эта папка используется для загрузки laravel (startup laravel).

Config

Эта папка будет содержать множество глобальных настроек для нашего приложения.

Database

Эта папка содержит наши элементы базы данных, такие как миграции и сиды (seeders).

Public

Эта папка Public содержит многие активы приложения, такие как изображения, таблицы стилей и скрипты.

Resources

В эту папку мы поместим наши представления. Представления – это страницы, которые видит пользователь.

Routes

Эта папка содержит все маршруты для нашего приложения.

Storage

Laravel использует эту папку для хранения сессий, кэша и журналов.

Test

В этой папке находятся файлы, которые мы используем для тестирования логики работы нашего приложения.

Vendor

В этой папке хранятся наши зависимости. Когда вы добавляете в приложение новые библиотеки (как в примере с пиццей «Добавки»), именно эта папка будет содержать эти библиотеки.

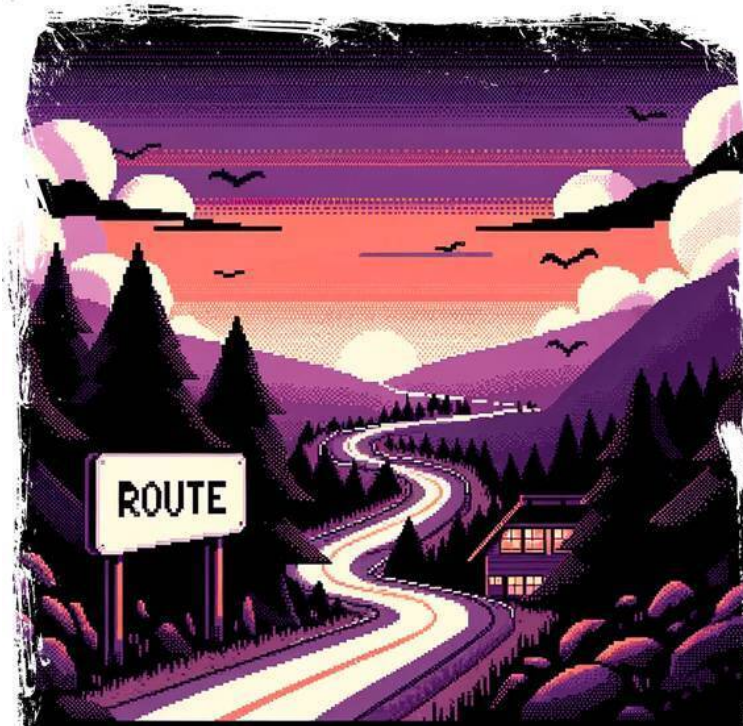
Узнаете ли вы composer.json file на изображении выше?

Помните, что именно здесь мы определяем зависимости (начинки для пиццы) для нашего приложения. Другим важным элементом является .env, который содержит все наши переменные окружения, такие как режим отладки и учетные данные базы данных.

Такова основная структура приложения Laravel. По мере дальнейшей работы с Laravel все это станет для вас более привычным.

Отличная работа! Вам понравится работать с Laravel. Засучим рукава и углубимся в работу с кодом.

2. Маршрутизация



Надежная система маршрутизации имеет решающее значение для выживания. Как опытный выживальщик находит верный путь к безопасности, так и мощная система маршрутизации гарантирует, что пользователь никогда не заблудится и всегда достигнет желаемого пункта назначения.

Обзор маршрутизации

Чтобы убедиться в том, что все находятся на одной волне, давайте сначала разберемся, что такое маршрутизация приложений.

Маршрут можно представить себе, как подобие дороги. Например, "Мы ехали по дороге (маршруту), чтобы добраться до кладбища". Маршрут определяет, как вы добираетесь из одного места в другое. Когда вы набираете URL-адрес веб-сайта, например, **site.com/graveyard**, вы сообщаете браузеру, что кладбище – это маршрут, по которому вы хотите проехать. Затем приложение говорит: "Хорошо, вы хотите отправиться на "кладбище"? Вот вывод, который я получил для маршрута "**graveyard**".

Создание маршрута с помощью Laravel довольно простое:

<?php

```
Route::get('graveyard', function(){
echo 'Добро пожаловать на кладбище!';
});
```

Здесь создается маршрут для страницы "кладбище". Приложение, получив запрос на "get" маршрута "graveyard", выполнит функцию и выдаст сообщение "Добро пожаловать на кладбище!".

Маршрутизация в Laravel

Наша маршрутизация в Laravel находится по адресу **routes\web.php**. Именно здесь мы будем добавлять все маршруты для нашего приложения.

В принципе, у нас есть четыре типа маршрутов: **POST**, **GET**, **PUT** и **DELETE**. Они выглядят следующим образом:

```
<?php

Route::post('/zombie', function () {
echo "We want to create a new zombie";
});
```

```
Route::get('/zombie', function () {
echo 'We want to read or view a zombie';
});

Route::put('/zombie', function () {
echo "We want to update an existing zombie";
});

Route::delete('/zombie', function () {
echo "We want to destroy a zombie";
});
```

Эти методы – POST, GET, PUT и DELETE – являются частью архитектуры **RESTful**, каждый из которых отражает определенное действие:

POST: Создание сущности (Create).
GET: Чтение сущности или нескольких сущностей (Read).
PUT: Обновить сущность (Update).
DELETE: удаление сущности (Delete).

Эта техника также называется CRUD (**Create**, **Read**, **Update**, **Delete**).

Чаще всего мы будем использовать метод GET, но есть

также маршрут, который можно использовать для получения любого метода:

```
<?php

Route::any('/zombie', function () {
echo "Any request from this zombie route";
});
```

Отлично!

Итак, как мы инициируем наши маршруты из браузера? В большинстве случаев мы используем GET-запрос. Введя **site.com/zombie**, мы получаем функцию GET. Но как насчет передачи данных?

Легко! Для этого подойдет HTML-форма, подобная этой:

```
<form method="POST" action="/zombie">
@csrf
@method('PATCH')
…
<input type="submit">
</form>
```

При нажатии кнопки submit на этой форме данные будут

отправлены на POST-маршрут **site.com/zombie**.

Обратите внимание на передачу @csrf и дополнительного указания метода. Указание метода необходимо из-за того, что в HTTP не существует метода 'PATCH'. Таким образом мы даем понять Laravel, какое именно действие нужно совершить с данными из этой формы.

Пример быстрой маршрутизации

Представьте, что перед вами стоит задача убить (удалить) зомби-изгоя! Сначала необходимо создать форму:

```
<form method="POST" action="/zombie">
@csrf
@method(DELETE)
<input type="hidden" name="id" value="2">
<input type="submit" value="Destroy">
</form>
```

На ней отображается кнопка "Уничтожить". Для простоты мы жестко задали идентификатор 2, который, как правило, зависит от конкретного зомби.

Далее, давайте составим маршрут:

```
<?php
```

```
use Illuminate\Http\Request;

Route::delete('/zombie', function(Request $request){
$id = $request->id; Zombie::destroy($id);
});
```

И вот уже нет проблемного зомби с идентификатором 2! Обратите внимание на включение класса Request из Laravel, который перехватывает данные запроса. Вам придется не забыть объявить пространство имен, когда вы захотите использовать объект запроса.

Внимание! *Этот пример пока не будет полностью рабочим, так как наша база данных и модели еще находятся в процессе создания. Мы займемся этим в ближайшее время.*

Мы использовали закрытие маршрутов. Далее обсудим разницу между закрытием маршрутов и контроллерами маршрутов.

Закрытие маршрута и действия контроллера маршрута

Закрытие маршрута – это непосредственная функция, содержащая код, как показано здесь:

```
Route::get('/zombie', function(){
echo 'Greetings from the Zombie Page!';
});
```

Для действия контроллера маршрута мы указываем, какой метод контроллера следует вызвать:

```
Route::get('/zombie', [ZombieController::class, 'index']);
```

Обращение к **/zombie** вызывает метод **index** в **ZombieController**.

Более подробно мы рассмотрим контроллеры в ближайшее время. Запомните эти различия, и они станут более понятными.

Параметры маршрута

Иногда маршруты требуют параметров.

Например, для просмотра конкретного зомби по адресу **site.com/zombie/5** требуется включить в маршрут параметр:

```
Route::get('/zombie/{id}', function($id){
echo "You've encountered a zombie with ID: " . $id;
});
```

Если наши модели и база данных работают, то это позволит получить и отобразить информацию о конкретном зомби:

```
Route::get('/zombie/{id}', function($id){
$zombie = Zombie::find($id);
echo 'Name: ' . $zombie->name . '<br />';
echo 'Strength: ' . $zombie->strength . '<br />'; echo 'Health: ' . $zombie->health . '<br />';
});
```

Напоминаем, что наша настройка не завершена, поэтому данный пример пока не будет работать идеально. С учетом сказанного, в следующей теме мы рассмотрим все это вместе.

3. Модели

Зомби-разработчики часто используют сложные запросы, которые могут привести к плохому и зараженному коду. Как разработчик Laravel, мы должны поддерживать наши запросы сильными и здоровыми.

Что же такое модели?

В Laravel модель – это PHP-класс, который управляет взаимодействием между кодом вашего приложения и базой данных. Расширение класса Laravel Eloquent Model позволяет сделать эти взаимодействия простыми и понятными.

Модель **Zombie**

Возьмем, к примеру, модель Zombie, которая будет размещена по адресу **/app/Models/Zombie.php**:

```
<?php

namespace App\Models;
use Illuminate\Database\Eloquent\Model;

class Zombie extends Model {
protected $table = 'zombies';
}
```

Этот код сообщает Laravel, что класс Zombie соответствует таблице zombies в вашей базе данных. Гипотетическая таблица zombies может выглядеть следующим образом:

zombies table

Field	Type	Length
id	INT	11
name	VARCHAR	50
strength	VARCHAR	20
health	INT	3
updated at	TIMESTAMP	
created at	TIMESTAMP	

Совет: Laravel автоматически управляет полями updated_at и created_at, если они существуют, регистрируя временные метки для новых добавленных строк и любых обновлений. Данные поля не нужно создавать. При написании миграции достаточно добавить **timestamps()**, но об этом позже.

Предположим, что у нас есть эта таблица в нашей базе данных. В одной из следующих глав мы обсудим миграции, которые позволяют нам легко создавать таблицы базы дан-

ных через наш код.

Теперь, когда вышеупомянутая таблица создана, мы можем взаимодействовать с базой данных с помощью Eloquent.

Eloquent: ORM в Laravel

Eloquent, ORM (Object-Relational Mapper) в Laravel, упрощает и украшает работу с базой данных. Вспомним код из предыдущего раздела:

```
<?php
use App\Models\Zombie;
Route::get('/zombie/{id}', function($id){
$zombie = Zombie::find($id);
echo 'Name: ' . $zombie->name . '<br />';
echo 'Strength: ' . $zombie->strength . '<br />'; echo
'Health: ' . $zombie->health . '<br />';
});
```

Если раньше наше приложение не могло найти класс Zombie, то с появлением модели мы можем обращаться к нему без проблем.

Стоит отметить, что, вызывая Zombie, мы обращаемся

именно к классу Zombie, расположенному по адресу **App \Models\Zombie**. Это понятие известно, как пространство имен, которое мы рассмотрим в одной из следующих глав.

Тем не менее, препятствие все еще существует.

Не имея в базе данных ни одного зомби, мы не можем получить доступ к упомянутому выше маршруту. Поэтому давайте создадим нового зомби по приведенному ниже маршруту:

```
<?php

Route::get('/admin/zombies/create', function(){
echo '<form method="POST" action="/admin/zombies/
create">
<input type="text" name="name"
placeholder="Name"><br>
<input type="text" name="strength"
placeholder="Strength"><br>
<input type="text" name="health"
placeholder="Health"><br>
<input type="hidden" name="_token"
value="' . csrf_token() . '">
<input type="submit" value="Create New Zombie">
</form>';
```

});

При посещении этого маршрута в браузере (site.com/admin/zombies/create) отображается простая форма.

При отправке формы данные публикуются в файл **site.com/admin/zombies/create** POST-маршрут, который должен выглядеть следующим образом:

```
<?php

Route::post('/admin/zombies/create', function () {
// создаем нового зомби
```

```
});
```

Добавив следующую реализацию:

```
<?php

use App\Models\Zombie;
use Illuminate\Http\Request;

Route::post('/admin/zombies/create', function(Request $request){

// instantiate a new zombie
$zombie = new Zombie();
$zombie->name = $request->name;
$zombie->strength = $request->strength;
$zombie->health = $request->health;
$zombie->save();

echo 'Zombie Created';
});
```

И затем отправить форму со следующими данными:

Имя: Johnny Bullet Holes
Strength: Сильный

Здоровье: 70

Вы получите сообщение '**Zombie Created**'. При просмотре нашей базы данных обнаруживается новая запись.

id	name	strength	health	updated_at	created_at
1	Johnny Bullet Holes	Strong	70	2015-09-08 14:35:56	2015-09-08 14:35:56

Впечатляет, не правда ли? Однако вместо того, чтобы вручную указывать имя, силу и здоровье, Laravel позволяет использовать более лаконичный подход:

```
<?php

use App\Zombie;
use Illuminate\Http\Request;

Route::post('/admin/zombies/create', function(Request $request){

// instantiate a new zombie using posted data
$zombie = Zombie::create($request->all());

echo 'Zombie Created';
```

```
});
```

При попытке использовать этот маршрут может возникнуть ошибка 'MassAssignmentException'. Это означает, что мы пытаемся выполнить массовое назначение классу "Зомби", не указав допустимые поля. В Laravel такая защита предусмотрена по умолчанию.

Чтобы разрешить массовое присвоение для атрибутов имени, силы и здоровья в нашем классе Zombie, просто добавьте:

```
protected $fillable = ['name', 'strength', 'health'];
```

Переделанный класс выглядит следующим образом:

```
<?php namespace App\Models;

use Illuminate\Database\Eloquent\Model; class Zombie extends Model {
protected $table = 'zombies';
protected $fillable = ['name', 'strength', 'health'];
}
```

Теперь, без ошибки массового присвоения, можно без труда создать еще одного зомби без особых усилий.

Предположим, что мы создали еще одного зомби:

Имя: Ted Manwalking
Strength: Слабый
Здоровье: 90

Тогда в нашей базе данных появятся следующие записи:

id	name	strength	health	updated_at	created_at
1	Johnny Bullet Holes	Strong	70	2015-09-08 14:35:56	2015-09-08 14:35:56
3	Ted Manwalking	Weak	90	2015-09-08 15:17:53	2015-09-08 15:17:53

Eloquent значительно облегчает процессы создания, чтения, обновления и удаления записей в базе данных. Далее мы рассмотрим отношения, облегчающие связывание данных между таблицами базы данных.

4. Модельные отношения

ОТНОШЕНИЯ (МОДЕЛИ)

Разработчик-зомби борется с отношениями, но разработчик Laravel превосходно пользуется преимуществами отношений в базе данных.

Зомби не хватает интеллекта для создания значимых отношений –связей между таблицами. В отличие от них, класс Eloquent в Laravel позволяет без особых усилий устанавливать и использовать отношения между таблицами.

Модельные отношения

Отношения связывают данные между таблицами. Представьте, что вы ведете блог с таблицами '**posts**' и '**comments**'.

Эти таблицы взаимосвязаны. У поста может быть МНОГО КОММЕНТАРИЕВ, в то время как комментарий всегда будет относиться к конкретному посту. Это называется отношениями.

Давайте создадим еще одну таблицу с именем weapons:

Таблица weapons:

Field	Type	Length
id	INT	11
zombie id	INT	11
name	VARCHAR	50

Обратите внимание на столбец 'zombie_id'. Он ссылается на столбец 'id' в таблице Zombies. Эта связь, известная как внешний ключ (Foreign Key), возникает, когда строка одной таблицы однозначно идентифицирует строку другой таблицы. Этот внешний ключ обеспечивает надежную связь между таблицами Weapons и Zombies.

Рассмотрим два вида оружия в нашей базе данных, связанных с зомби:

id	zombie_id	name
1	2	Axe
2	1	Shot Gun

Выше вы видите, что мы включили "Топор" для зомби с идентификатором 2 и "Дробовик" для зомби с идентификатором 1.

Теперь сформулируем модель оружия для связи с таблицей оружия. Путь к ней – **app/Models/Weapon.php**:

```
<?php namespace App\Models;

use Illuminate\Database\Eloquent\Model; class Weapon extends Model {
protected $table = 'weapons';
}
```

Для отображения информации о зомби, включая его оружие, мы можем использовать этот код:

Конец ознакомительного фрагмента.