

Back-end

Музычук Артём

Примечание

Данное руководство представляет собой учебное пособие по Back-end разработки. Широко рассматривается язык программирования PHP, паттерн SOLID вместе с “чистым кодом”, фреймворк Laravel, а также работа с системой контроля версий GIT. *В будущем планируется добавить CI/CD, Docker, SQL.*

В документе присутствует авторский сленг и возможные неточности. При выявлении ошибки прошу написать мне в гитхаб.

Документация разбита на 4 части:

- 1) Язык программирования PHP
- 2) Git, в частности GitHub
- 3) Фреймворк Laravel (База)
- 4) Фреймворк Laravel (Продвинутый)

В Первой части руководства описывается язык PHP(7-8), его синтаксис, тонкости работы, сильные и слабые стороны. Подробно разобрана работа с функциями, массивами, строками и тд. Вводится понятие ООП, концепции и методы работы, классы, интерфейсы, наследования и другим основные принципы. Разбирается паттерн SOLID и правила написания «Чистого кода».

Во второй части рассматривается работа с системой контроля версий Git, основные задачи, создание, клонирование, скачивание репозитория, решение конфликтов и тд.

В третьей части начинается введение в фреймворк Laravel(8-9), написание сайта(front & back), настройки панели администратора, ознакомление с базовыми понятиями как back-end разработки в целом, так и разработке на Laravel. Большое внимание уделяется правильному написанию кода, концепциям разработке, модели MVC, базам данных и CRUD.

В четвертой части продолжается изучение Laravel, но на более глубоком и продвинутом уровне. Предстоит написать продвинутый современный сайт для блогеров. Продвинутая валидация, очереди, сторонние сервисы, взаимодействие с front частью, отправка сложной формы, загрузка изображений на сайт, оптимизация и многое другое будет рассмотрено.

Оглавление

Часть 1. PHP.....	7
Структура	7
Переменные	8
Типизация и кавычки.....	8
Базовые функции	10
Приведение типов	10
Начало ООП	10
Области видимости	10
Ссылки.....	11
Константы	11
Путь к файлу	12
Операторы.....	12
ЧТЕНИЕ И ЗАПИСЬ В ФАЙЛ	13
ЦИКЛЫ	14
Массивы.....	14
Простые массивы	14
Ассоциативные массивы	15
МНОГОМЕРНЫЕ МАССИВЫ.....	15
ИНТЕРПОЛЯЦИЯ ЭЛЕМЕНТОВ МАССИВА В СТРОКИ	16
КОНСТРУКЦИЯ ЛИСТ	16
ОБХОД МАССИВА ЦИКЛОМ FOR И FOREACH	16
ОБХОД двумерного ассоциативного МАССИВА ЦИКЛОМ FOREACH.....	17
ОБЪЕДИНЕНИЕ И СЛИЯНИЕ.....	18
СРАВНЕНИЕ	18
ПОИСК В МАССИВЕ И ПРОВЕРКА НА СУЩЕСТВОВАНИЕ	18
ЗАДАЧИ	19
Функции	20
ФУНКЦИИ - верблюжий стиль	20
Возвращаемые функции	21
Передача аргументов по ссылке	22
Параметры по умолчанию	23
ПЕРЕДАЧА ЗАРАНЕЕ НЕИЗВЕСТНЫХ ПАРАМЕТРОВ (кол-ва)	23
статические переменные	23
ВАЖНО возврат массива ВАЖНО.....	24
РЕКУРСИЯ, ВЛОЖЕННЫЕ И АНОНИМНЫЕ ФУНКЦИИ.....	24
динамический вызов функции	25
анонимные функции	25
ЗАМЫКАНИЕ ВАЖНО	26
СТРОКИ.....	27
основные функции.....	27
ВСЕ ФОРМЫ НЕОБХОДИМО ОБЕЗОПАСИТЬ	28
ФОРМАТНЫЙ ВВОД-ВЫВОД.....	29
Разбиение на подстроки	29
ПРОТОКОЛЫ ПЕРЕДАЧИ ПАРАМЕТРОВ (GET POST).....	30
GET.....	30
ПАРСИНГ	30
POST.....	31
GET.....	31

ЧЕКБОКСЫ.....	33
ЗАГРУЗКА ФАЙЛА НА СЕРВЕР	33
ФОРМА ОТПРАВКИ ПИСЕМ	33
ООП (МЕТОДЫ) (конструкторы класса).....	34
МЕТОДЫ.....	34
Конструкторы класса	35
НАСЛЕДОВАНИЕ	36
Абстрактные классы и методы	37
Скриншоты.....	38
Классы php.....	49
Сеттеры и Геттеры в классах	50
Конструктор в классах	52
Наследование в php.....	54
Конструктор в классах наследниках.....	55
Абстрактные классы	56
Статические методы и свойства в классах.....	57
Интерфейсы в php.....	58
Уровни доступа в php: private, protected, public	59
S.O.L.I.D	60
Принцип единственной ответственности (single responsibility principle)	60
Принцип открытости/закрытости (open-closed principle).....	64
Принцип подстановки Лисков (Liskov substitution principle)	64
Принцип разделения интерфейса (interface segregation principle).....	65
Принцип инверсии зависимостей (dependency inversion principle).....	65
Часть 2. Git	66
2.1 Какие задачи решает Гит и для чего он нужен?.....	66
2.2 Основные команды в Git	67
2.3 Методология git flow. Наиболее популярный вариант разработки с Git	67
2.4 Хранилища Github и Gitlab	68
2.5 Создаем проект. Инициализируем Git. Первый commit и add в Git	68
2.6 Git clone и .gitignore. Клонируем проект с github	70
2.7 Git clone и .gitignore. Клонируем проект с github и gitlab	72
2.8 Создание задачи(issue) в github и gitlab с веткой и fetch	72
2.9 Pull request в github.....	75
2.10 SSH key для github	78
2.11 Merge conflict в git. Git stash. Git reset.....	78
Часть 3. Laravel (база).....	79
3.1 Введение.....	79
3.1.1 Этапы создания проекта.....	79
3.2 Model-View-Controller.....	80
3.2.1 Этапы:.....	81
3.2.2 Http строка.....	81
3.2.3 Request(route).....	81
3.2.4 Controller.....	82
3.2.5 Database (миграции)	84
3.2.6 Model.....	88
3.2.7 Модель чтения данных из базы	89

3.2.8 Строение Model	92
3.2.8 ELOQUENT	93
3.2.9 DATABASE	93
3.2.10 Методы чтения данных (retrieve).....	94
3.2.11 Создание объекта в бд (добавление)	97
3.2.12 Методы обновления данных(update)	98
3.2.13 Метод удаления данных(delete) и soft delete	99
3.2.14 SOFT DELETE	100
3.2.15 Комбинированные методы создания и обновления данных	101
3.3 Миграции. Редактирование миграций	103
3.3.1 Создание колонок.....	104
3.3.2 Удаление колонок	105
3.3.3 Редактирование колонок	106
3.3.4 ИЗМЕНЕНИЕ ТИПА ДАННЫХ.....	107
3.3.5 УДАЛЕНИЕ ТАБЛИЦЫ	107
3.4 View	108
3.4.1 Знакомство	108
3.4.2 Шаблоны view	109
3.5 Bootstrap в laravel.....	113
3.6 CRUD через интерфейс	114
3.6.1 INDEX	115
3.6.2 CREATE.....	115
3.6.3 STORE.....	116
3.6.4 SHOW.....	118
3.6.5 EDIT / UPDATE	119
3.6.6 DELETE	120
3.7 Отношения один ко многим	120
3.7.1 Модификация crud - категории(один ко многим)	123
3.7.2 Создание:.....	123
3.7.3 Редактирование:	124
3.8 Отношения многие ко многим	125
3.8.1 Модификация crud - Теги(многие ко многим)	127
3.8.2 Создание	128
3.8.3 Редактирование	130
3.9 Модификация crud - обработчик ошибок	131
3.10 Отношения один ко многим и многие ко многим через конвенцию Laravel	132
3.11 Однометодные контроллеры	135
3.12 Класс Request	137
3.13 Класс Service	139
3.14 Классы Factory и Seed	143
3.14.1 Класс seed	144
3.14.2 Factory (фабрика)	145
3.14.3 Совместная работа	145
3.15 Пагинация в Laravel.....	149
3.16 Шаблон Filter, фильтрация данных в Laravel.....	153
3.16.1 НЕИСПОЛЬЗУЕМЫЙ ШАБЛОН	153
3.16.2 Используемый шаблон.....	156
3.17 Admin LTE в Laravel, устанавливаем админку	159
3.18 Авторизация Laravel.....	160

3.19 Класс Middleware в Laravel (Роли авторизации)	162
3.20 Класс Policy	164
3.21 Асинхронный CRUD в Laravel. Приложение Postman.	167
3.22 Restful API. Класс Resource в Laravel, асинхронный ответ с бека.....	170
3.22.1 RESTful API.....	170
3.22.2 Принципы архитектурного стиля REST:.....	172
3.22.2 Как работает RESTful API?	172
3.22.3 HTTP-аутентификация.....	173
3.22.4 OAuth.....	173
3.22.5 Асинхронный ответ в бека	173
3.22.6 ФИЛЬТРАЦИЯ:	178
3.22.7 ПЕРЕЛИСТИВАНИЕ СТРАНИЦ	179
3.23 JWT Token. Асинхронные роуты	182
3.24 CRUD с транзакцией.....	186
Часть 4. Laravel (продвинутый).....	187
4.1 Новый проект – сайт для блогеров	187
Скачиваем шаблон и создаем проект.....	187
4.2 Подключаем bootstrap, auth и базу данных.....	187
4.3 Первоначальный план и создаем миграции	188
4.4 Подгружаем фронт страницы блога. Список постов	190
4.5 Ставим фронт админки, установка Admin Lte	191
4.6 Определяемся со стратегией и создаем задел для CRUD категорий	191
4.7 Страница создания категории(форма создания).....	192
4.8 Добавляем атрибут content для поста, визуальный редактор Summernote.....	194
4.9 Добавляем атрибуты для загрузки изображений	195
4.10 Добавляем возможность привязывать пост к категории	198
4.11 Добавляем возможность привязывать тэги	199
4.12 Реализуем апдейт у постов, форма, отображение изображений.....	201
4.12.1 Как отобразить картинку	201
4.13 Изменяем сообщения об ошибке в классе Request	205
4.14 Полностью добавляем основу CRUDA для пользователей	207
4.15 Внедряем роли для пользователей и улучшаем валидацию для апдейта	212
4.16 Создаем Middleware для админки и auth middleware, делаем кнопку выхода ..	217
4.17 Новая концепция auth.....	218
4.18 Автогенерация пароля, отправка почты. Mailtrap и класс Mailable в Laravel.....	221
4.19 Отправление письма с подтверждением учетной записи Email verification.....	226
4.20 Отправка писем с использованием очередей. Создаем свой Notification класс.	228
4.21 Очереди. Проблематика	228
4.22 Создаем класс Job. Пример работы с очередями любого алгоритма/действия .	234
4.23 Делаем общие косметические изменения по дизайну и редиректам.....	238
4.24 Создаем личный кабинет пользователя.....	238
4.25 Структура в бд для лайков и их отображение в личном кабинете	242
4.26 Структура в бд для комментариев и их отображение в личном кабинете	245
4.27 Вывод ленты постов. Случайные посты. Популярные посты. Пагинация	249
4.28 Carbon даты. Добавляем схожие посты (одиночные страницы для поста)	251
4.29 Реализуем добавление комментариев в посте. Геттеры в моделях Laravel.....	255
4.30 Добавление лайков к постам. Бонусом - вывод постов по категориям	261

Часть 1. PHP

Структура

```
<p><?=Hello php';?></p>
<!-- Для одной строки-->

<?php

echo date(DATE_RSS);
    // Выражение которое что-то делает;
echo '-----';
echo 8
+
4;
echo "text";
    // Так тоже можно конец строки только ;
    # Тоже комментарий
/*and this/


?>
<!-- Несколько строк -->

<?php
if(mt_rand(0,1)){
    ?>
        <p color="green">jksdf</p>
        <?php
} else {
    ?>
        <p color="red">s12212</p>
        <?php
}
?>
<!—HTML код не пишется в блоке PHP-->

<p>dsffd</p>
<?= include_once 'second.php'; ?> <!-- предупреждение -->
<?= require_once 'second.php'; ?> <!-- ошибка -->
<p>sdfsfd&lt;/p>

<?php
```

Переменные

```
$peremen = 5;  
$Peremen = 3;  
$one = $two = $three = 10;  
echo $peremen - 2;
```

Типизация и кавычки

```
// СЛАБОТИПИЗИРОВАННЫЙ (тип данных не указывается)  
// " " - одно и то же (НО {$value} РАБОТАЕТ ТОЛЬКО С "")  
// `` - в них передаются команды операционной системы  
// /$ - экранирование  
// {$car}3 car3 car 3 - вывод без пробела/не рабочий код/с пробелом
```

	СТАТИЧЕСКАЯ/ ДИНАМИЧЕСКАЯ	СИЛЬНАЯ/ СЛАБАЯ	ЯВНАЯ/ НЕЯВНАЯ
ADA	статическая	сильная	явная
BOO	статическая	сильная	неявная
C	статическая	слабая	явная
C#	статическая	сильная	явная
C++	статическая	слабая	явная
COMMON LISP	динамическая	сильная	неявная
D	статическая	сильная	явная
DELPHI	статическая	сильная	явная
GO	статическая	сильная	неявная
HASKELL	статическая	сильная	неявная
JAVA	статическая	сильная	явная
JAVASCRIPT	динамическая	слабая	неявная
PASCAL	статическая	сильная	явная
PERL	динамическая	слабая	неявная
PHP	динамическая	слабая	неявная
PYTHON	динамическая	сильная	неявная
RUBY	динамическая	сильная	неявная
RUST	статическая	сильная	явная
SCALA	статическая	сильная	неявная
SMALLTALK	динамическая	сильная	неявная

Базовые функции

```
// isset(var) - проверка на существование (есть ли значение у
переменной)
// empty(var) - проверка на пустоту
// gettype(var) - узнать тип
// is_...(var) - проверка на тип is_bool(var)
// is_numeric(var) - проверка явл ли переменная числом
// round() - округление
// unset(var) - полное удаление объекта
// mt_rand(1,10) - случайное число
```

Приведение типов

// Явное и неявное приведение типов

```
// "33qwe" - 12 даст 21 несмотря на то что из строки вычитается инт и
выкидывает строки - НЕЯВНОЕ ПРИВЕДЕНИЕ
// все что 0 ("0", 0, 0.0, пустая строка) - false остальное true, тру всегда
неявное преобразовывается в 1 а фолс в пустую строку
// $float = 4.4; (int)$float = 4 -- явное преобразование
```

Начало ООП

```
// ООП
// КЛАСС - собственных сложный тип данных (шаблон для
переменных)
// ОБЪЕКТ - переменные класса
// создаем класс и переменные этого класса которые будут иметь
общий шаблон
// класс нельзя разрывать тегом php
// определить класс 2 раза нельзя
// рекомендуется выносить классы в отдельный файл
```

Области видимости

```
// область видимости позволяет обращаться к переменной из
определенных мест
// спецификатор доступа - указывает какие переменные доступны
снаружи(вне класса) а какие нет (public, private) - закрытие и защита (внутри
класса пользоваться можно)

// статическая переменная класса static - нужна для заранее
установленной переменной (обычные переменные создаются после создания
объекта, а static уже есть в классе заранее) позволяет обращаться к полю как
```

внутри так и снаружи. Для доступа к такой переменной не нужно создавать объект

Ссылки

// при передачи значения объекта передается именно ссылка а не значение (ВАЖНО СКРАНШОТ В ФАЙЛЕ) РАБОТАЕТ ЭТО ПО АНАЛОГИИ С УКАЗАТЕЛЯМИ В СИ

```
$first = 5;  
$second = &$first;  
$second = 3;  
echo $first; echo $second;  
//в ответе будет 5 и 3
```

```
// $obj = new Object;  
// $obj->x = 3;
```

```
// $obj2 = $obj;  
// $obj2->x = 4;  
// echo "x: {$obj->x}, x: {$obj2->x}"  
//ОБА ОБЪЕКТА БУДУТ ХРАНИТЬ X ОТ 4 ТАК КАК ЭТО
```

ССЫЛКИ

//\$obj2 = clone \$obj; - дубликат объекта (теперь уже отдельный объект без ссылки)

```
class Point /*(можно без скобок)*/ {  
    public $x;  
    public $y;  
    public static $num = 120;  
}
```

```
$point = new Point; // объект класса Point  
$point->x = 12;  
$point->y = 10;  
echo $point->x;
```

```
Point::$num; // вывод статической переменной класса
```

Константы

//КОНСТАНТЫ и динамическое создание(скриншот)

```

define('NUMBER', 11); //также возвращает истину или ложь если
можно или нельзя создать константу
echo NUMBER;

echo defined("NUMBER"); // проверка на сущ константы

$num = mt_rand(1,10);
$name = "VALUE($num)"; // Динамическое создание из текста из
числа
define($name, $num);
echo constant($name);

class Myconst {
    const NAMES = "artem";
}

// Myconst::NAMES - получение константы

// сущ стандартные константы типа __FILE__(путь к файлу
скрипта) и тд - ЧИТАТЬ ДОКИ

```

Путь к файлу

//ПУТЬ К ФАЙЛУ

```

//Бывают абсолютные(полная директория) и относительные
пути(относительно файла)
// require_once ../../point.php - относительный
// require_once __DIR__ . '/point.php'

// конкатенация - слияние строк, с помощью неявного
преобразования получим строку из числа
// $num(1) = $num . "+5" (1+5) // $num.="+5" (1+5) // echo $num,
"+5"
//echo '<br>';

```

Операторы

//умнож дел и тд\ выражения - строка до ; \ операнды -
переменные или числа \ инструкция - выражение \ операция - наименьшая
единица выражения

//var++ - постфиксная запись, увеличивает значение но возвращает старое
//++var - префиксная запись, увеличивает значение и возвращает его
//&& - И || - ИЛИ ^ - XOR !x - not << - побитовый сдвиг ($6 << 1 = 12$ ($0110 \rightarrow 1100 = 12$)) --- сдвиг влево - умножение сдвиг вправо - деление

```
// $x = 1;  
// $y = '1';  
  
//$ опущен  
// x < y;  
// x > y;  
// x <= y;  
// x >= y;  
// x == y; - сравнение переменных БЕЗ УЧЕТА ИХ ТИПА (true)  
// x != y - не равно не учитывая тип  
// x === y - ПОЛНОЕ РАВЕНСТВО (FALSE)  
// x !== y  
  
// x <=> y - какая-то хуйня  
  
// 0 == " - true  
// 0 == NULL - true  
// 0 == 'hello' - true (нет числа - значит пустота значит тру)  
// 0 == '8dfw' - false (есть число а строка после числа не важна)  
  
//if elseif else  
  
// if(!(true)) - false  
  
// выражение ? выр-истина : выр-ложь;  
// $var1 = null; $var1 = $var1 ?? 1; --- переинициализировать переменную !!если она пуста или null!!  
  
//switch - ОДИН В ОДИН КАК В СИ switch(true) - проверка как if
```

ЧТЕНИЕ И ЗАПИСЬ В ФАЙЛ

```
// $homepage = file_get_contents('./text.php'); - считать файл
```

```
// echo $homepage;  
  
// $homepage .= "\n\t123" ('\n\t123' - не сработает)  
// file_put_contents('./text1.txt', $homepage); - запись в файл
```

ЦИКЛЫ

```
// ОДИН В ОДИН КАК В СИ  
//for(;;) - бесконечный цикл
```

Массивы

Простые массивы

```
echo '<br>';  
  
$arr = array('Hello', 'world', "123");  
$arr2 = ['1', 10 => '2', '3']; // указываем для второго элемента  
индекс 10, для 3 элемента индекс будет 11 (на 1 больше максимального)  
$arr[10] = '!!!'; // пустые скобки [] идут по порядку  
  
$text = "hello";  
$arr3 = (array) $text; // массив с [0] = "hello"  
  
echo $arr; //выводит 'array'  
echo $arr[0]; // 'Hello'  
  
echo '<br>';  
echo '<br>';  
echo '<pre>';  
    print_r($arr); //Выводит весь массив  
echo '</pre>';  
  
/*Array  
(  
    [0] => Hello  
    [1] => world  
    [2] => 123  
)*/  
  
/*Array  
(  
    [0] => 1  
    [10] => 2
```

```
[11] => 3
)*/
```

Ассоциативные массивы

```
//Массивы у которых индексом могут быть строки
// его индексы - КЛЮЧИ
// его элементы - ЗНАЧЕНИЯ
// РЕГИСТР ИМЕЕТ ЗНАЧЕНИЕ

$arr4 = [
    'one' => '1',
    'two' => '2',
    'three' => 3
];

$arr4['one'] = 123;

echo '<pre>';
print_r($arr4);
echo '<pre>';
```

МНОГОМЕРНЫЕ МАССИВЫ

```
//создаются как и одномерные массивы

$transport = [
    "auto" => ['lada', 'mers', 'bmw'], // это все один элемент
    'plain' => ['airbus777']
];

echo '<pre>';
print_r($transport);
echo '<pre>';

/*Array
(
    [auto] => Array
        (
            [0] => lada
            [1] => mers
            [2] => bmw
```

```

    )
)*/  

echo $transport['auto'][2]; //bmw

```

ИНТЕРПОЛЯЦИЯ ЭЛЕМЕНТОВ МАССИВА В СТРОКИ

```

$arr5[0] = 12;  

$arr6['one'] = 'one';  

$arr7[0][0] = 123;  

echo '<br>'. "Array[0] = $arr6[one]"; //Array[0] = one; (" не нужны)  

echo '<br>'. "Array[0] = {$arr7[0][0]} и еще текст";  

echo '<br>'. "Array[0] = " . $arr7[0][0] . " и еще текст";

```

КОНСТРУКЦИЯ ЛИСТ

```

// индексы должны начинаться с 0 и не быть ассоциативными

$arrays = [1,2,3];
list($one1, $two2, $three3) = $arrays;
echo '<br>'. $one1; // 1

// так можно менять значения переменных без 3 переменной

$xx = 1;
$yy = 2;
list($yy, $xx) = [$xx, $yy];

```

ОБХОД МАССИВА ЦИКЛОМ FOR И FOREACH

```

echo '<br>' . '_____'. '<br>';
  

$numbers = [1,2,3,4,5,6, true, 'hello'];
for($i = 0; $i <= count($numbers); ++$i){
    echo $numbers[$i] . '<br>';
}
  

echo '<br>' . '_____'. '<br>';
  

$number2 = [

```

```

'one' => 1,
'two' => false,
'three' => "mamba"
];

foreach($number2 as $key => $value){
    echo "$key : $value <br>";      // ВЫВОД КЛЮЧ -
ЗНАЧЕНИЕ
}

echo '<br>' . '_____'. '<br>';

$number3 = [
    'one' => 1,
    'two' => false,
    'three' => "mamba"
];

foreach($number2 as $value){
    echo "$value <br>";      // ВЫВОД ТОЛЬКО КЛЮЧА
}

```

ОБХОД двумерного ассоциативного МАССИВА ЦИКЛОМ FOREACH

```

echo '<br>' . '_____'. '<br>';

$names = [
    'man' => ['artem','vlad','ivan'],
    'girl' => ['alina', 'polina', 'irina'],
    'neutra' => ['none']
];

foreach ($names as $key => $arrays) {
    echo "<b>{$key}</b>";
    foreach($arrays as $value){
        echo "<li>{$value}</li>";
    }
}

/*for($i = 0; $i < count($arrays); $i++){
    echo "<li>". $arrays[$i] . "</li>";
}*/
}
```

ОБЪЕДИНЕНИЕ И СЛИЯНИЕ

```
$n1 = ['1' => 1, '2' => 2];
$n2 = ['1' => 3, 4 => 4];
$n3 = $n1+$n2; // 1 2 4 (n2 '1' - не запишется так как n1 левее от
плюса и главное)
echo '<pre>';
    print_r($n3);
echo '</pre>';

/*Array
(
    [1] => 1
    [3] => 2
    [4] => 4
)*/           //индексы соединяются

$n4 = [1, 2];
$n5 = [3, 4];
$n6 = $n4+$n5; // 1 2 так как у второго массива индексы как и у
первого а первый главное
$n6 = array_merge($n4, $n5); // 1 2 3 4 -- функция СЛИВАЕТ
МАССИВЫ
```

СРАВНЕНИЕ

// == != === !== --- БУДУТ РАВНЫ ТОЛЬКО ЕСЛИ ОНИ
ПОЛНОСТЬЮ СОВПАДАЮТ (типы данных у ключей не важны)

ПОИСК В МАССИВЕ И ПРОВЕРКА НА СУЩЕСТВОВАНИЕ

```
// проверка на сущ функцией isset();
// in_array(2, $arr, true/false) - проверка есть ли элемент 2 в
массиве ТИП ВАЖЕН
// array_key_exists('car', $array) - проверка есть ли ключ
// array_search(2, array) - поиск ключа по значению (выводит
индекс)
// unset($array[2]/$array) - удалить элемент массива или
полностью массив
```

ЗАДАЧИ

```
// ВЫВЕСТИ СЛУЧАЙНЫЙ ЭЛЕМЕНТ МАССИВА

$arrand = [1,2,3,4,5,6];
$rand = array_rand($arrand);
echo $rand;
//echo arrand(rand(0, count($arrand) - 1));

$arrand2 = [
    100 => 'one',
    200 => 'two',
    300 => 'АБОВА'
];

$rand2 = array_rand($arrand2);
echo '<pre>';
    print_r($rand2);
echo '</pre>';

// создать массив с кол-вом от 5 до 10 и знач от 0 до 100 и
отсортировать

$len = rand(5,10);
for($i = 0; $i <= $len; $i++){
    $arand[$i] = rand(0,100);
}
sort($arand);
echo '<pre>';
    print_r($arand);
echo '</pre>';

//Считать данные из файла и сделать его массивом - ФУНКЦИЯ
FILE      ВАЖНО
$text = file('text.txt');
echo '<pre>';
    print_r($text);
echo '</pre>';
```

Функции

ФУНКЦИИ - верблюжий стиль

```
function myFunction($a,$b){ //параметры
    $sum = $a+$b;
    return $sum;
    //echo $sum;
}

echo myFunction(20,4); //аргументы
// myFunction();

//      Указание типа возвращаемого значения

function myFunction2(int $a, int $b) : int { //параметры
    return $a+$b;
    //echo $sum;
    //return $sum
}

echo '<br>'.myFunction2(20.0,4.0); //аргументы
echo gettype(myFunction(20.0,4.0));
// myFunction();
```

Иногда относят к типу данных

Объявление = придумывание название переменной которая явл функцией

В ней **данные** могут явл **действием**



```
function sayHello() {
    echo 'hello';
    echo ' lalalal';
    $name = 'Vasya';
    echo $name;
}
```

Сигнатура функции(название и скобки)
Усы - тело функции



```
sayHello();
```

Вызов

ФУНКЦИЯ - ДЕЙСТВИЕ И НАЗЫВАТЬ ЕЕ НАДО КАК ДЕЙСТВИЕ (ВЕРБ НАТАЦИЯ)

Нужны они для посторения кода в разных местах и илучшение читабельности

```
function sum($a, $b) {  
    $sum = $a + $b;  
    echo $sum|
```

Функция с аргументами (значения передаются в эти переменные (еще не знаем что там будет))

Возвращаемые функции

Функции могут что-то возвращать и не возвращать

ПЕРЕД ВОЗВРАЩАЕМЫМИ ФУНКЦИЯМИ ВСЕГДА НУЖНО ПИСАТЬ КЛЮЧЕВОЕ СЛОВО GET (GetSum)

```
function getSum($a, $b) {  
    $sum = $a + $b;  
    return $sum;  
}
```

Теперь наша функция(переменная) будет хранить какое-нибудь получившееся значение

```
function getSum($a, $b) {  
    $sum = $a + $b;  
    return $sum;  
}  
getSum( 55, 10);  
//echo $name . "\n";  
//echo $age . "\n";  
//echo $hobby . "\n";  
//echo $isMarried . "\n";  
//echo $pet . "\n";
```

Run: index.php
D:\wamp64\bin\php\php7.4.0\php.exe D:\00work\course\php\index.php
Process finished with exit code 0

Так мы переменную не получим

```

function getSum($a, $b) {
    $sum = $a + $b;
    return $sum;
}
$result = getSum(a: 55, b: 10);
echo $result;
//echo $name . "\n";
//echo $age . "\n";

```

А так получим тк ФУНКЦИЯ ХРАНИТ ЗНАЧЕНИЕ

```

7     echo $name;
8 }
9
10    function getSum($a, $b) {
11        $sum = $a + $b;      I
12        echo 'Hello';
13        return $sum;
14    }
15    $num = getSum(a: 55, b: 10);
16    echo $num + 20;
17 //echo $name . "\n";
18 //echo $age . "\n";
19 //echo $hobby . "\n";
20 //echo $isMarried . "\n";
getSum()

```

D:\wamp64\bin\php\php7.4.0\php.exe D:\00work\course\php\index.php
Hello85
Process finished with exit code 0

Если функция что-то возвращает и печатает то при ее вызове она напичтает

То есть функция остается функцией и помимо возврата значения она еще и произведет действия внутри скобок

Передача аргументов по ссылке

// По умолочанию передается значение (переменная не изменится)
 // Передача по ссылке меняет переменную
 // ОБЪЕКТЫ И МАССИВЫ ПО УМОЛОЧАНИЮ
ПЕРЕДАЮТСЯ ПО ССЫЛКЕ
 // Переменные существуют только в пределах функций
 // чтобы переменная стала глобальной нужно написать
 ключевое слово global \$var; и ее можно менять через функцию везде (не рекоменд)

```

$arg = 10;
echo '<br>' . $arg; //10
function funcArg(&$arg){
    $arg += $arg;
    return $arg;
}

```

```
echo '<br>'.fincArg($arg); //20
```

Параметры по умолчанию

```
function param($b, $a = 10){ // указать сначала а потом б  
нельзя тк первым идет значение по умолочанию (без инициализации)  
    return $a + $b;  
}
```

```
echo '<br>' . param(10, /*100*/); // второй аргумент можно не  
указывать тк мы его уже задали, но его можно перезадать
```

ПЕРЕДАЧА ЗАРАНЕЕ НЕИЗВЕСТНЫХ ПАРАМЕТРОВ (кол-ва)

```
function neiz(...$items){ // $items рассматривается как массив  
и мы можем передать в функцию любое кол-во переменных  
    foreach($items as $values){  
        echo $values . "<br>";  
    }  
}  
  
neiz(1,2,3,4,4,5);
```

// ТАК МОЖНО ДЕЛАТЬ И ОБРАТНЫЕ
МАНИПУЛЯЦИИ

```
function neiz2($a,$b,$c,$d){  
    echo '<br>' . $a . '<br>';  
    echo '<br>' . $b . '<br>';  
    echo '<br>' . $c . '<br>';  
    echo '<br>' . $d . '<br>';  
}
```

```
$itm = [1,2,3,4];  
echo neiz2(...$itm);
```

статические переменные

```
function my(){  
    static $i = 0;
```

```

        return ++$i;
    }

echo my();
echo my();
echo my();

//выведет 123 тк static существует все время выполнения
программы и сохраняет значение
// так можно хранить значения

```

ВАЖНО возврат массива ВАЖНО

```

$bytes = 123456789;
function myArrays($bytes){
    $k = $bytes / 1024;
    $m = $bytes / 1024;
    $b = $bytes / 1024;
    return [$k,$m,$b];
}

list($a,$b,$c) = myArrays($bytes);
echo $a . '<br>';
echo $b . '<br>';
echo $c . '<br>';

// print_r(myArrays($bytes));

```

РЕКУРСИЯ, ВЛОЖЕННЫЕ И АНОНИМНЫЕ ФУНКЦИИ

```

// РЕКУРСИЯ (чем меньше тем лучше) (нужно обязательно
придумать выход) (любая задача может быть решена без рекурсии (почти))
function rec($count){
    if($count < 10){
        echo '<br>' . $count++;
        rec($count);
    } else return;
}

rec(0);

```

```
// вложенные функции

function ar1(){
    function ar2(){
        echo 'Hello';
    }
}

ar1();
ar2(); // невозможно достучаться до вложенной функции без
вызыва основной
```

динамический вызов функции

```
function first(){
    echo "first";
}

function second(){
    echo 'second';
}

$myFunc = rand(0,1) ? 'first' : 'second';

echo $myFunc();
```

анонимные функции

```
// пример 1 на скриншоте

class Points{
    public $x;
    public $y;
};

$points1 = new Points;
$points1 -> x = 1;
$points1 -> y = 2;

$points2 = new Points;
$points2 -> x = 3;
$points2 -> y = 4;
```

```

$arr = [$points1, $points2];

usort($arr, function($a,$b){ // анонимная функция
которая возвращает 0 или 1
    $dist_a = sqrt($a -> x ** 2 + $a -> y **2); // 0
    $dist_b = sqrt($b -> x ** 2 + $b -> y **2); // 1
    return $dist_a <=> $dist_b; // 0 или 1
});

echo '<pre>';
echo print_r($arr);
echo '</pre>';

// (
// [0] => Points Object
// (
//     [x] => 1
//     [y] => 2
// )

// [1] => Points Object
// (
//     [x] => 3
//     [y] => 4
// )

// )

```

ЗАМЫКАНИЕ ВАЖНО

```

// это функция, которая запоминает состояние
окружения в момент своего создания
// если состояние переменных изменится после
функции, то замыкание продолжит содержать первоначальное состояние при
ее вызове
// применяются только к анонимным функциям

// используется как замена глобальных переменных

$message = "text text text";
$usage = function() use ($message){

```

```
$message = "none"; // создаем дубликат и
запоминаем состояние (работаем с переменной не меняя значение)
    return $message;
}; // ОБЯЗАТЕЛЬНО ;

echo $usage();
echo '<br>';
echo $message;
```

СТРОКИ

```
// работают строки так же как и массивы

$string = "aboba";
$string[0]; // a

// в английском utf-8 буква занимает 1 байт, а в русском 2
байта. Поэтому 0 символ строки не даст 1 букву слова
// для решения этой проблемы существует
расширение mb-string

//strltn - функция длины строки (байт) (1 русский
символ = 2 байта)
//mb_strlen - функция которая решает проблему 2 байт
русской буквы
```

основные функции

**//1. Вывод посимвольно - либо цикл for либо
функция**

```
// chr(ascii) - принимает код и возвращает букву
// ord(char) - наоборот
```

//2. Поиск в подстроке

```
// substr(переменная, начало, кол-во символов);
```

**//3. Индекс первого вхождения подстроки в
строку**

```
//strpos($str, 'hello')
```

```
$strr = "я артем музырчук";
```

```
$str2 = mb_substr($str, mb_strpos($str, 'арт'));
echo '<br>' . $str2;
```

//4. Замена подстроки / удаление - пустая строка

```
$str = "[b]Это жирный текст[/b]";
//$str = str_replace('[b]', '<b>');
$str = str_replace(['[b]', '[/b]'], ["<b>", "</b>"],
$str/*, $num - кол-во замен*/); // замена массивов
echo $str;
```

//5. Удаление начальных и конечных пробелов

```
// trim($str, 'Н' // удаление символов Н);
```

//6. Преобразование \n в br для HTML

```
//nl2br($str); $str = 'dfd\n fd\n';
```

//7. Удаление html тегов

```
//strip_tags($str);
```

// ОЧЕНЬ СУКА

ВАЖНО ЧТОБЯ НЕ ВЗЛОМАЛИ

ВСЕ ФОРМЫ НЕОБХОДИМО ОБЕЗОПАСИТЬ

// **htmlspecialchars**(POST['ИМЯ ФОРМЫ'])

- преобразует код в безопасное представление

// применение этой функции гарантирует что любой введенный в форму код не будет выполнен

// ВСЕГДА ПРИМЕНЯТЬ ПРИ ОБРАБОТКЕ

ТЕКСТА ПОЛЬЗОВАТЕЛЯ

ФОРМАТНЫЙ ВВОД-ВЫВОД

// ФУНКЦИЯ printf точно так же как и в си

Разбиение на подстроки

```
$str12 = "name, fmale, adress";
echo "<pre>";
    print_r(explode(',', $str12/*, 2 - кол-во
```

элементов массива*/);

```
echo "</pre>";
```

```
/*Array
(
    [0] => name
    [1] => fmale
    [2] => adress,
)*/
```

// функция implode(' ', \$str12); делает наоборот

```
$fds = ['a','v','b'];
```

```
echo "<pre>";
    print_r(implode(' ', $fds));
echo "</pre>";
```

// a,v,b

```
//json --- скриншот
//json_encode($array);
```

ПРОТОКОЛЫ ПЕРЕДАЧИ ПАРАМЕТРОВ (GET POST)

GET

```
// передача параметров в строке запроса  
// передаются в строке запроса после /? -  
все что после - параметры метода гет  
  
// параметры этого метода автоматом  
помещаются в суперглобальный массив $_GET - ИМЕНА ПАРМЕТРОВ -  
КЛЮЧИ МАССИВА  
  
// https://vk.com/?id=1221  
// $_GET['id'] = 1221;  
  
// для разделения параметров  
используется знак &  
  
// https://vk.com/?id=1221&log=artem  
// $_GET['id'] = 1221;  
// $_GET['log'] = artem;  
  
// print_r($_GET);  
  
//  
https://vk.com/?id[]=1221&id[]=444&log=artem  
  
// print_r($_GET); // ПОЛУЧИМ  
МАССИВ ID ИЗ ДВУХ ЭЛЕМЕНТОВ 1221 444  
  
// ПРОБЕЛЫ И РУССКИЕ СИМВОЛЫ  
НЕ ДОПУСТИМЫ ТК ПОЙДЕТ ПРЕОБРАЗОВАНИЕ В БЕЗОПАСНЫЙ  
ФОРМАТ  
// но можно преобразовать в  
безопасный формат urlencode('абоба');
```

ПАРСИНГ

```
$url =  
"https://vk.com/name:123/pass:21";
```

```

        echo '<pre>';
print_r(parse_url($url/*,PHP_URL_...*/));
        echo '</pre>';

/*Array
(
    [scheme] => https
    [host] => vk.com
    [path] => /name:123/pass:21
)*/
```

POST

// передача параметров в теле http
документа

// ДАЛЕЕ ИДУТ ФОРМУ ГЕТ И ПОСТ
?>

GET

ФОРМЫ

```

<form action="" method="GET">
    <!-- Метод пол умолчанию - гет    акцион - ссылка на
обрабатывающий док -->
    <label for="one">Введите имя: </label>
    <input id="one" type="text" name ='login' placeholder="Имя:">
    <label for="two">Введите пароль</label>
    <input id="two" type="text" name="pass" placeholder="Пароль:>
    <button type="submit">Отправить</button>
</form>
<?php
    if(!empty($_GET['login']) && !empty($_GET['pass'])){
        echo '<pre>';
        print_r($_GET);
        echo '</pre>';
        // через name мы получаем ключ к значению гет
    } else {
```

```

        // прекращает выполнение скрипта
        echo "Поля не заполнены";
    }

//http://php/main.php?login=sdf&pass=sdfdf --- гет протокол

?>

<!-- ПОСТ -->

<form action="" method="POST">
    <label for="nam">Введите имя: </label>
    <input type="text" name="name" id="nam" value="<?=
htmlspecialchars($_POST['name'], ENT_QUOTES);?>">
    <br>
    <button type="submit">ОТПРАВИТЬ</button>
</form>

<?php

$errors2 = [];
if (!empty($_POST)) {
    if (empty($_POST['name'])) {
        $errors2[] = 'dflkj';
    }

    if (empty($errors2)) {
        echo htmlspecialchars($_POST['name']);
        exit();
    }
}
if (!empty($errors2)) {
    foreach($errors2 as $err){
        echo $err;
    }
}

?>

```

ЧЕКБОКСЫ

```
<form action="check.php" method="POST">
    <input type="checkbox" name="html" id="html">Я знаю хтмл <br>
    <input type="checkbox" name="css" id="html">Я знаю css <br>
    <input type="checkbox" value="1" name="php" id="html">Я знаю php
<br>
    <button type="submit">Отправить</button>
    <!-- По умолочанию значения отправятся как "on" value="..." меняет его
-->
    <!-- ЕСЛИ У НАС МНОЖЕСТВЕННЫЙ ВЫБОР ТО ИМЯ
ПЕРЕДАЕМ КАК МАССИВ -->
    <!-- ЧЕКБОКС СЕЛЕКТ И РАДИО МОЖНО ОБРАБАТЫВАТЬ ПО
РАЗНОМУ -->
</form>
```

ЗАГРУЗКА ФАЙЛА НА СЕРВЕР

```
<form action="file.php" method="POST" enctype="multipart/form-data">
    <input type="file" name="filename" id="file">
</form>
```

ФОРМА ОТПРАВКИ ПИСЕМ

```
<form action="sandmail.php" method="POST">
    <select name="subject">
        <option disabled selected>Тема письма</option>
        <option value="1">Вопрос по уроку</option>
        <option value="2">Спросить как дела</option>
        <option value="3">Просто</option>
    </select>
    <input type="email" name="email" id="email" placeholder="Введите майл:
" maxlength="50" required>
    <textarea name="text" placeholder="Введите ваше сообщение:
" maxlength="100" cols="30" rows="10" required></textarea>
    
    <input type="text" name="capcha" id="" placeholder="Введите код с
картинки: " maxlength="5" required>
    <button type="submit">Отправить</button>
```

```
</form>
```

ООП (МЕТОДЫ) (конструкторы класса)

```
<?php
```

МЕТОДЫ

функции объявленные в пределах класса и могут работать только с переменными класса

```
// $this - выбор переменной текущего класса
// НЕ СТАТИЧЕСКИЕ МЕТОДЫ нельзя вызвать до создания объекта
(создаём объект класса и через него вызываем методы) (при указании static
методу можно вызывать его через класс имяКласса::названиеМетода )
// чтобы сослаться на статистическую переменную класса внутри
класса нужно написать ключевое слово self::имяКласса (можно указывать и
имя класса но если оно изменится ничего работать не будет) -- селф -
универсальность
// Хороший тон использовать self вместо имени класса

class Hello{
    public function printText(){
        return "Hello world!";
    }
}

$hel1 = new Hello;
echo $hel1 -> printText();

class Hello2{
    private $x;
    private $y;

    public function setX($x){
        $this->x = $x;           // таким образом можно менять
        приватные переменные класса
    }                           // Но переменные
защищены и их нельзя изменять напрямую (инкапсуляция)
    public function setY($y){
```

```

        $this->y = $y;
    }

    public function getX(){
        return $this->x;           // таким образом можно вернуть
значения приватных переменных
    }
    public function getY(){
        return $this->y;
    }

    public function dist(){
        return sqrt($this->getX() ** 2 + $this->getY() ** 2); //
получает x и y
    }
}

$p1 = new Hello2;
$p1->setX(4);
$p1->setY(2);
echo $p1->dist();

```

Конструкторы класса

```

// При создании класса и объекта существуют спец методы которые
скрыты от разраба и запускаются они неявно
// Эти методы можно переопределить

```

// Конструктор - специальный метод класса который автоматом выполняется при создании объекта (до вызова всех остальных НЕ СТАТИЧ методов класса)

// Конструктором можно управлять передавая в класс аргументы если в конструкторе они есть

```

class People{
    private $name;
    private $x;
    private $y;
    public function __construct($x=0, $y=0){ // если задать значения по
умолчанию можно избежать ошибок
        echo 'Конструктор вызван';
        $this->name = 'артем';
    }
}

```

```

        $this->x = $x;
        $this->y = $y;
    }
}

$artem = new People(1,2);
$artem2 = new People(1,2);
$artem3 = new People(1,2);

// ТАКИМ ОБРАЗОМ МОЖНО ЗАДАВАТЬ МНОГО ТОЧЕК
print_r($artem);

```

```

// Конструктор вызванPeople Object
// (
//   [name:People:private] => artem
//   [x:People:private] => 1
//   [y:People:private] => 2
// )

```

// метод __toString() - преобразует класс в строку (СКРИНШОТ)

НАСЛЕДОВАНИЕ

```

// дочерний класс будет иметь такие же переменные и методы как и у
родительского А ТАКЖЕ СОДЕРЖАТЬ СВОИ УНИКАЛЬНЫЕ
// спецификатор доступа static не будет доступен и дочерних классов
// спецификатор protected работает как статик но доступен при
наследовании (снаружи его вызвать нельзя)
// переменные указываем протектед а методы паблик
// при наследовании у дочерних классов можно ПЕРЕОПРЕДЕЛЯТЬ
методы и называется это ПЕРЕГРУЗКА (у родителя и ребенка есть
одинаковые по названию методы но работают они по разному) //
parent::info() - обращение к родительскому методу

```

```

class Animals{
    public $legs = 4;
    public function info(){
        echo "У животных {$this->legs} ноги";
    }
}

class Dog extends Animals{

```

```

public $name = 'Собака';
public function voice(){
    echo " я {$this->name} и я издаю звук гав-гав";
}
}

$dog = new Dog;
echo $dog->info();
echo $dog->voice();

```

Абстрактные классы и методы

// абстрактные классы работают точно так же как и обычные, но от них нельзя создавать объекты (используется при наследовании чтобы обезопасить класс) - это класс который создан только для наследования от него

```

abstract class Animals2{
    public $legs = 4;
    public function info(){
        echo "У животных {$this->legs} ноги";
    }
}

abstract public function color(); // абстрактный метод
ГАРАНТИЯ ТОГО ЧТО ЭТЫЙ МЕТОД ОБЯЗАТЕЛЬНО БУДЕТ В
ДОЧЕРНИХ КЛАССАХ (должен быть определен) (обязательная перегрузка)
}

class Dog2 extends Animals2{
    public $name = 'Собака';
    public function voice(){
        echo " я {$this->name} и я издаю звук гав-гав";
    }
}

public function color(){
    echo "Red"; // перегрузили абстрактный класс
}
}

$dog2 = new Dog2;
echo $dog2->info();
echo $dog2->voice();
echo $dog2->color();

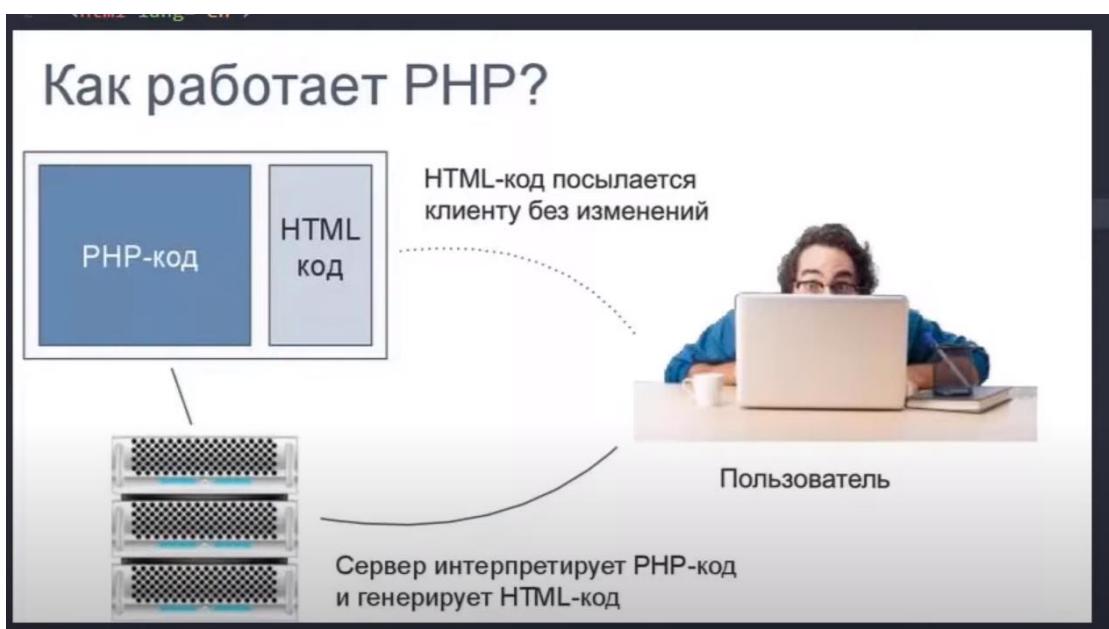
```

```
// ключевое слово final - обратное классу abstract (нельзя  
перегружаться и иметь наследников)
```

```
// $dog instanceof Dog - проверка на принадлежность
```

```
?>  
</body>  
</html>
```

Скриншоты



Типы данных в PHP

В языке PHP используются восемь типов данных:

- **boolean** - логический тип, содержит значения TRUE или FALSE;
- **integer** - целое число;
- **float** - вещественное число;
- **string** - текст произвольной длины;
- **array** - массив;
- **object** - объект;
- **resource** - ресурс (например, файл);
- **NULL** - значение NULL.

Значение	Описание
\$var = (int) \$var;	Приведение к целому типу int
\$var = (integer) \$var;	Приведение к целому типу int
\$var = (bool) \$var;	Приведение к логическому типу boolean
\$var = (boolean) \$var;	Приведение к логическому типу boolean
\$var = (float) \$var;	Приведение к вещественному типу double
\$var = (double) \$var;	Приведение к вещественному типу double
\$var = (real) \$var;	Приведение к вещественному типу double
\$var = (string) \$var;	Приведение к строковому типу string
\$var = (array) \$var;	Приведение к массиву

```
9:45 • Явное приведение к объекту
'42' - число
1337 - число
1337 - число
1337 - число
1337 - число
1337.0 - число
'0x539' - НЕ число
'02471' - число
'0b10100111001' - НЕ число
'1337e0' - число
'not numeric' - НЕ число
array () - НЕ число
9.1 - число
NULL - НЕ число
```

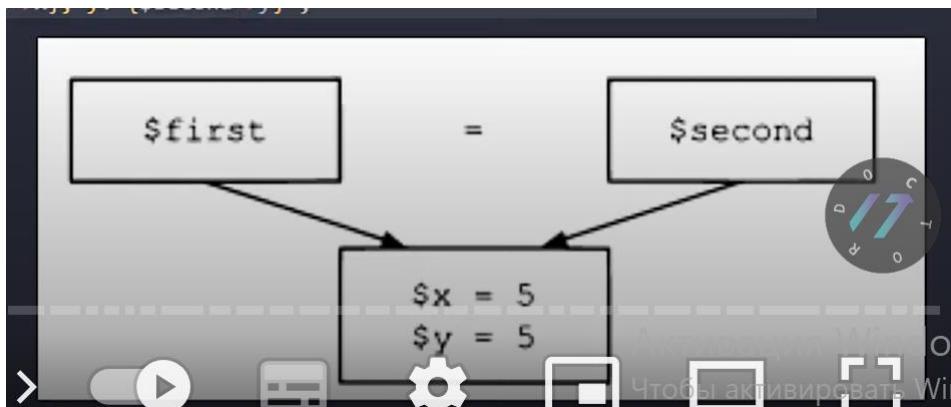
```
 php
$first = new Point;
$first-&gt;x=3;
$first-&gt;y=3;

$second = $first;
$second-&gt;x=5;
$second-&gt;y=5;

echo "x: {$first-&gt;x}, y: {$first-&gt;y}";
?&gt;</pre

```

выводит 5 5 в обоих переменных



```

1 <html lang="en">
2   <head>
3     <meta charset="UTF-8">
4     <meta name="viewport" content='width=device-width,
5       initial-scale=1.0'>
6     <title>Урок 12</title>
7   </head>
8   <body>
9     <?php
10    require 'point.php';
11
12    $p1 = new Point;
13    $p1->x=3;
14    $p1->y=5;
15
16    $p2 = new Point;
17    $p2->x=7;
18    $p2->y=12;
19
20    $distance = [pow((($p2->x-$p1->x), 2)+pow
21      [($p2->y-$p1->y), 2])
22    ??
23  </body>
24 </html>

```

```

1 <html lang="en">
2   <head>
3     <meta charset="UTF-8">
4     <meta name="viewport" content='width=device-width,
5       initial-scale=1.0'>
6     <title>Урок 14</title>
7   </head>
8   <body>
9     <?php
10    $num = mt_rand(1,10);
11    $name = VALUE($num);
12    define($name, $num);
13    echo constant($name);
14    ?>
15  </body>
16 </html>

```

Не защищено | phlessons/lesson18/

Поразрядное пересечение - И (AND)			Поразрядное объединение - ИЛИ (OR)			Поразрядное исключающее ИЛИ (XOR)		
x1	x2	x1 & x2	x1	x2	x1 x2	x1	x2	x1 ^ x2
0	0	0	0	0	0	0	0	0
0	1	0	0	1	1	0	1	1
1	0	0	1	0	1	1	0	1
1	1	1	1	1	1	1	0	0

Поразрядное отрицание (NOT)

x	$\sim x$
0	1
1	0

Поразрядное пересечение - И (AND)

x1	x2	x1 & x2
0	0	0
0	1	0
1	0	0
1	1	1

Поразрядное объединение - ИЛИ (OR)

x1	x2	x1 x2
0	0	0
0	1	1
1	0	1
1	1	1

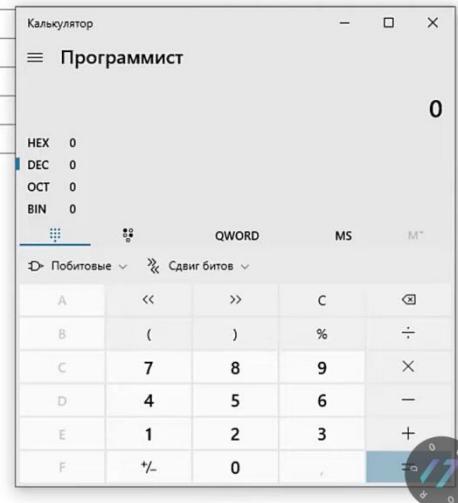
Поразрядное исключающее ИЛИ (XOR)

x1	x2	x1 ^ x2
0	0	0
0	1	1
1	0	1
1	1	0

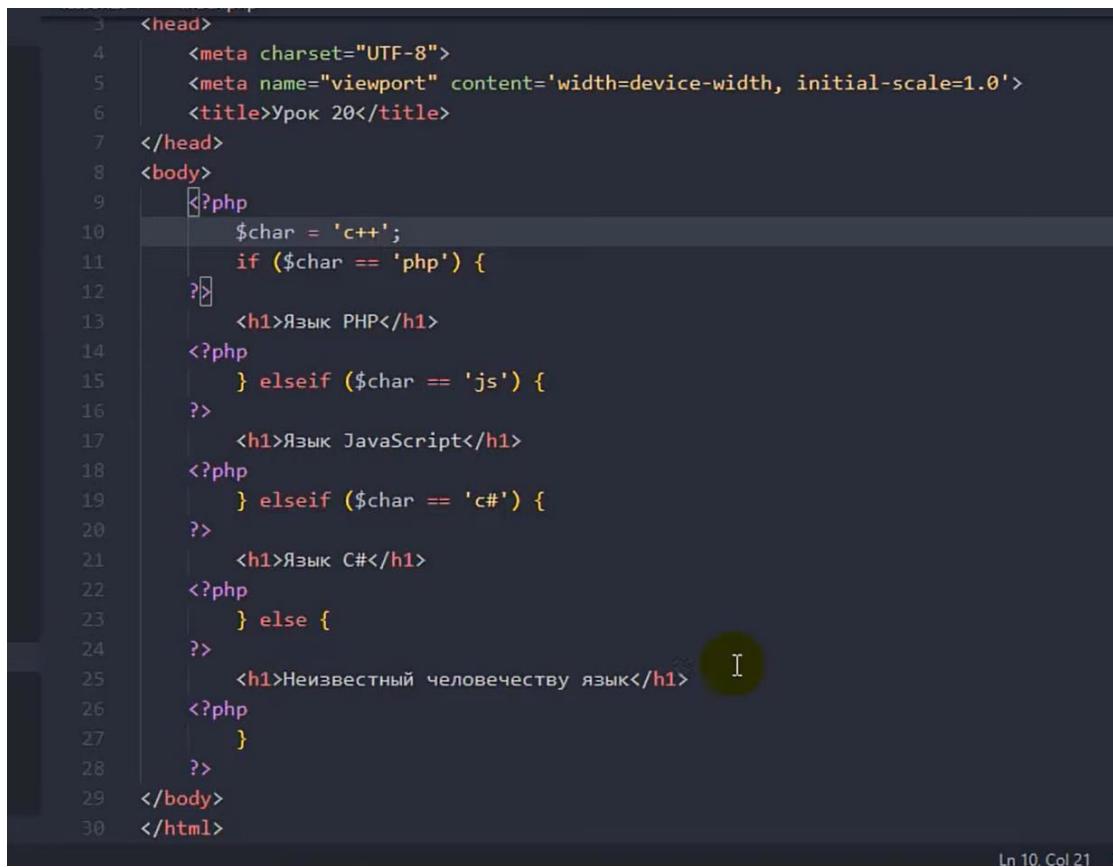
Поразрядное отрицание (NOT)

x	$\sim x$
0	1
1	0

6 = 0110
10 = 1010
0010

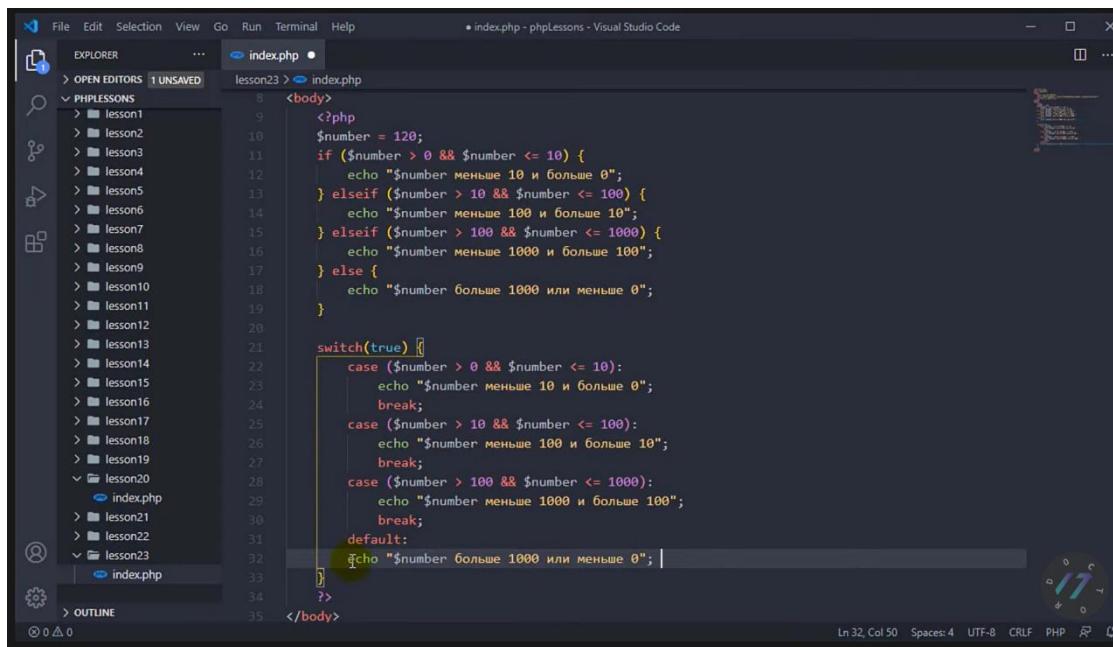


2



```
3 <head>
4     <meta charset="UTF-8">
5     <meta name="viewport" content='width=device-width, initial-scale=1.0'>
6     <title>Урок 20</title>
7 </head>
8 <body>
9     <?php
10    $char = 'c++';
11    if ($char == 'php') {
12        ?>
13        <h1>Язык PHP</h1>
14    <?php
15    } elseif ($char == 'js') {
16        ?>
17        <h1>Язык JavaScript</h1>
18    <?php
19    } elseif ($char == 'c#') {
20        ?>
21        <h1>Язык C#</h1>
22    <?php
23    } else {
24        ?>
25        <h1>Неизвестный человеческому языку</h1>
26    <?php
27    }
28    ?>
29 </body>
30 </html>
```

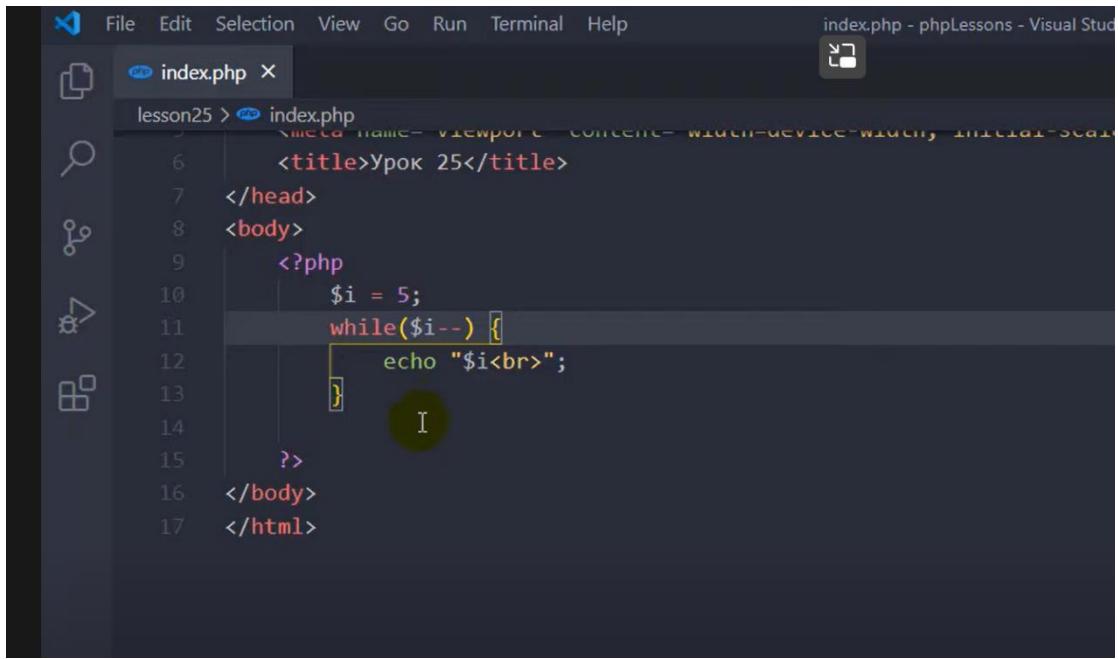
Ln 10, Col 21



```
<?php
if ($number > 0 && $number <= 10) {
    echo "$number меньше 10 и больше 0";
} elseif ($number > 10 && $number <= 100) {
    echo "$number меньше 100 и больше 10";
} elseif ($number > 100 && $number <= 1000) {
    echo "$number меньше 1000 и больше 100";
} else {
    echo "$number больше 1000 или меньше 0";
}

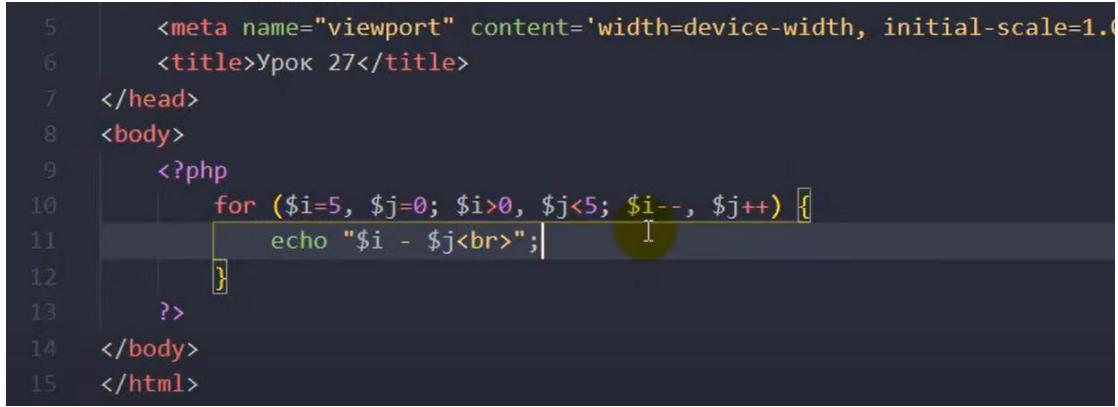
switch(true) {
    case ($number > 0 && $number <= 10):
        echo "$number меньше 10 и больше 0";
        break;
    case ($number > 10 && $number <= 100):
        echo "$number меньше 100 и больше 10";
        break;
    case ($number > 100 && $number <= 1000):
        echo "$number меньше 1000 и больше 100";
        break;
    default:
        echo "$number больше 1000 или меньше 0";
}
```

Ln 32, Col 50 Spaces:4 UTF-8 CRLF PHP



A screenshot of the Visual Studio Code interface. The title bar shows "File Edit Selection View Go Run Terminal Help" and "index.php - phpLessons - Visual Studio". The left sidebar has icons for file, search, and refresh. The main editor window displays the following PHP code:

```
5 <meta name="viewport" content='width=device-width, initial-scale=1.0'>
6 <title>Упок 25</title>
7 </head>
8 <body>
9 <?php
10 $i = 5;
11 while($i--) {
12     echo "$i<br>";
13 }
14 ?>
15 </body>
16 </html>
```



A screenshot of the Visual Studio Code interface. The title bar shows "File Edit Selection View Go Run Terminal Help" and "index.php - phpLessons - Visual Studio". The left sidebar has icons for file, search, and refresh. The main editor window displays the following PHP code:

```
5 <meta name="viewport" content='width=device-width, initial-scale=1.0'>
6 <title>Упок 27</title>
7 </head>
8 <body>
9 <?php
10 for ($i=5, $j=0; $i>0, $j<5; $i--, $j++) {
11     echo "$i - $j<br>";
12 }
13 ?>
14 </body>
15 </html>
```

```
3 <head>
4     <meta charset="UTF-8">
5     <meta name="viewport" content='width=device-width, initial-scale=1.0'>
6     <title>Урок 27</title>
7 </head>
8 <body>
9     <?php
10
11    for ($i=2; $i<100; $i++) {
12        for($j=2; $j<$i; $j++) {
13            if(($i % $j) != 0) {
14                continue;
15            } else {
16                $flag = true;
17                break;
18            }
19        }
20        if(!$flag) echo "$i ";
21        $flag = false;
22    }
23 ?>
24 </body>
25 </html>
```

```
File Edit Selection View Go Run Terminal Help
index.php - phpLessons - Visual Studio Code
index.php X
lesson39 > index.php > html > body
4 <head>
5     <meta charset="UTF-8">
6     <meta name="viewport" content='width=device-width, initial-scale=1.0'>
7     <title>Урок 39</title>
8 </head>
9
10 <body>
11     <?php
12         function formatSize($bytes) {
13             $kbytes = $bytes / 1024;
14             $mbytes = $kbytes / 1024;
15             $gbytes = $mbytes / 1024;
16             return [$bytes, $kbytes, $mbytes, $gbytes];
17         }
18
19         echo '<pre>';
20         print_r(formatSize(54989777));
21         echo '</pre>';
22
23 ?>
```

index.php

```
lesson39 > index.php > html > body
  3
  4     <head>
  5         <meta charset="UTF-8">
  6         <meta name="viewport" content='width=device-width, initial-scale=1.0'>
  7         <title>Урок 39</title>
  8     </head>
  9
 10    <body>
 11        <?php
 12            function formatSize($bytes) {
 13                $kbytes = $bytes / 1024;
 14                $mbytes = $kbytes / 1024;
 15                $gbytes = $mbytes / 1024;
 16                return [$bytes, $kbytes, $mbytes, $gbytes];
 17            }
 18
 19
 20            list($bytes, $kbytes, $mbytes, $gbytes) = formatSize(54989777);
 21            echo '</pre>';
 22
 23        ?>
```

Активация Windows

index.php

```
lesson40 > index.php > html > body > first
  1
  2     <meta charset="UTF-8">
  3     <title>Урок 40</title>
  4
  5     </head>
  6
  7     <body>
  8         <?php
  9             function first() {
 10                 return "First function";
 11             }
 12             function second() {
 13                 return "Second function";
 14             }
 15             $newFunction = rand(0,1) ? 'first' : 'second';
 16
 17             echo $newFunction();
 18             ?>
 19         </body>
 20
 21     </html>
```

Активация Windows

АНОНИМНЫЕ ФУНКЦИИ

Сортировка точек

File Edit Selection View Go Run Terminal Help

index.php - phpLessons - Visual Studio Code

lesson40 > index.php > html > body

```
15     }
16     $fst = new Point;
17     $fst->x = 12;
18     $fst->y = 5;
19
20     $snd = new Point;
21     $snd->x = 1;
22     $snd->y = 1;
23
24     $thd = new Point;
25     $thd->x = 4;
26     $thd->y = 10;
27
28     $arr = [$fst, $snd, $thd];
29
30     usort($arr, function($a, $b) {
31         $dist_a = sqrt($a -> x ** 2 + $a -> y ** 2);
32         $dist_b = sqrt($b -> x ** 2 + $b -> y ** 2);
33         return $dist_a <= $dist_b;
34     });
35 }
```

echo <pre>;</pre> 5:41:15 / 7:49:45 · Рекур... >

Активация Windows

Ln 34, Col 8 Spaces: 4 UTF-8 Справка раздел "Параметры".

JSON

File Edit Selection View Go Run Terminal Help

index.php - phpLessons - Visual Studio Code

lesson46 > index.php > html > body

```
6     <meta name="viewport" content='width=device-width, initial-scale=1.0'>
7     <title>Урок 46</title>
8   </head>
9
10  <body>
11    <?php
12      $arr = [
13          'name' => 'Petr',
14          'phones' => [
15              '888',
16              '777'
17          ],
18      ];
19      $encodeArr = json_encode($arr);
20      echo $encodeArr;
21      // echo '<pre>';
22      // print_r(json_decode($encodeArr, true));
23      // echo '</pre>';
24    </?>
25  </body>
```

```
Array
(
    [scheme] => http
    [host] => www.site.ru
    [user] => user
    [pass] => pass
    [path] => /path/index.php
    [query] => par=value
    [fragment] => andh
)
```

```
index.php script.php
lesson50 > script.php
1 <?php
2 echo 'Исходное имя файла - ' . $_FILES['filename']['name'] . '<br>';
3 echo 'Размер файла в байтах - ' . $_FILES['filename']['size'] . '<br>';
4 echo 'MIME тип файла - ' . $_FILES['filename']['type'] . '<br>';
5 echo 'Временный файл, в котором сохранен загруженный файл - ' . $_FILES['filename']
6 ['tmp_name'] . '<br>';
7
8 ?>
```

```
index.php script.php
lesson50 > script.php
1 <?php
2 if($_FILES['filename']['size'] > 3*1024*1024)
3 //     return 'Привет, мир!';
4 // }
5 // }
6 // echo Hello::printText();
7
8 class Page {
9     static $content = 'Тело сайта<br>';
10    public static function footer() {
11        return 'Это подвал сайта<br>';
12    }
13    public static function header() {
14        return 'Это шапка сайта<br>';
15    }
16    public static function site() {
17        echo self::header() .
18             self::$content .
19             self::footer();
20    }
21 }
```

```
2 // echo "</pre> ";
3
4 class Point {
5     private $x;
6     private $y;
7     public function __construct($x=0, $y=0) {
8         $this->x = $x;
9         $this->y = $y;
10    }
11    public function __toString()
12    {
13        return "({$this->x}, {$this->y})";
14    }
15 }
16 $obj = new Point(10,12);
17 echo "{$obj}";
```

The screenshot shows a code editor window for a file named `index.php`. The file path is listed as `lesson54 > index.php > html > body`. The code itself defines a `Animal` class with an `info()` method that outputs a message about the number of legs. It then defines a `Dog` class that extends `Animal`, setting its name to "Собака" and overriding the `info()` method to output a bark sound. Finally, it creates a `$dog` object and calls both `info()` and `voice()` methods on it.

```
File Edit Selection View Go Run Terminal Help
index.php - phpLessons - Visual Studio Code
index.php X
lesson54 > index.php > html > body
10 <body>
11 <?php
12 class Animal {
13     public $legs = 4;
14     public function info() {
15         echo "У меня {$this->legs} лапы.";
16     }
17 }
18
19 class Dog extends Animal {
20     public $name = "Собака";
21     public function voice() {
22         echo "{$this->name} издает звук гав-гав.";
23     }
24 }
25
26 $dog = new Dog();
27 $dog->info();
28 $dog->voice();
```

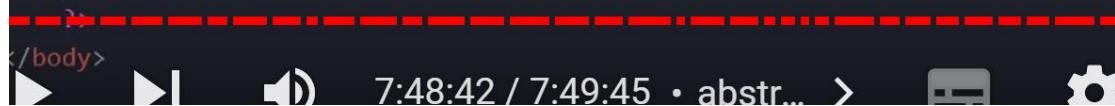
```
        }

        $dog = new Dog();
        $cat = new Cat();

        if($dog instanceof Animal) {
            echo 'Dog является экземпляром класса Animal';
        }
        if($dog instanceof Dog) {
            echo 'Dog является экземпляром класса Dog';
        }
        if($dog instanceof Cat) {
            echo 'Dog является экземпляром класса Cat';
        }

        // $dog->info();
        // $dog->voice();
        // $dog->parentInfo();


```



Классы php

Весь код на php всегда сводится к ООП - а это классы и объекты

Классы нужны для “сохранения в одну переменную блок кода переменных действий и тд”

Создаем информацию и сохраняем ее куда-то (в класс)

В класс можно сохранить и переменные и функции и все что душе угодно

**ПЕРЕМЕННЫЕ КЛАССА - СВОЙСТВА
ФУНКЦИИ КЛАССА - МЕТОДЫ**

Класс - удобное хранение чего-то

Именование - верб

```
class Person{
    public $name = 'victor';
}
```

Чтобы задать свойство или метод класса нужно дать ему МОДИФИКАТОР ДОСТУПА

Нужны они для задания области видимости элементов класса в программе

Паблик - доступно везде (можно вызвать везде)

Протектед - доступно только у наследников и у класса

Приват - только в классе

Название свойств и методов - верб

```
jerz / desktop / Desktop / test.php / ...
```

```
<?php

class Person{
    public $name = 'victor';
    public function getHello($name){
        echo "Привет {$name}";
    }
}

$person = new Person();
```

Чтобы получить все из класса нужно создать ОБЪЕКТ

```
<?php

class Person{
    public function getHello($name){
        echo "Привет {$name}";
    }
}

$person = new Person();
$name = "Artem";
echo $person->getHello($name);
```

И далее мы уже обращаемся К ОБЪЕКТУ А НЕ К КОАССУ и у него ВЫЗЫВАЕМ методы или свойства через (->) (заходим в дом через дверь а не пробив стену)

Достучаться к внутренностям КЛАССА можно только через ЕГО ОБЪЕКТ посредством ВЫЗОВА У ОБЪЕКТА МЕТОДОВ

Переменные без доллара вызываются

Сеттеры и Геттеры в классах

Нужны для изменения свойств класса

```
<?php

class Person{
    public $name = 'artem';
    public function getHello($name){
        echo "Hello {$name}";
    }

    public function setName($name){
        $this->name = $name;
    }
}

$person = new Person;
$person2 = new Person;
$person3 = new Person;

echo $person->name . "\n";
echo $person2->name. "\n";
echo $person3->name. "\n";
```

Ключевое слово `$this` говорит о том что мы будем менять что-то у ОБЪЕКТА класса(пока что мы не знаем какое у него имя и есть ли он) у которого мы вызываем метод тк все действия мы оперируем именно с объектом а не с самим классом

Далее мы говорим `$this->name` у этого объекта поменяй свойство name на то что мы передадим методу

```
rs > admin > Desktop > test.php > Person > s
<?php

class Person{
    public $name = 'artem';
    public function getHello($name){
        echo "Hello {$name}";
    }

    public function setName($name){
        $this->name = $name;
    }
}

$person = new Person;
$person2 = new Person;
$person3 = new Person;

$person->setName('vlad');
echo $person->name . "\n";
```

При создании объекта все что было в классе по сути комикуется в объект

Нейминг - верб (второе слово - сво-во)

```
        public function getName(){
            return $this->name;
        }
    }

$person = new Person;
$person2 = new Person;
$person3 = new Person;

$person->setName('vlad');
echo $person->getName() . "\n";
```

Геттер

Конструктор в классах

Для того чтобы не прописывать 100110 разных геттеров и сеттеров
придумали конструктор

```
C: > Users > admin > Desktop > test.php > ...
1  <?php
2
3  class Person{
4      public $name;
5      public $age;
6      public $hobby;
7
8      public function setProperty($name, $age, $hobby){
9          $this->name = $name;
10         $this->age = $age;
11         $this->hobby = $hobby;
12     }
13
14 }
15
16 $person = new Person;
17
18 $person ->setProperty('artem', 19, 'gym');
19 print_r($person);
```

ВЫХОДНЫЕ ДАННЫЕ КОНСОЛЬ ОТЛАДКИ ТЕРМИНАЛ

Person Object
(
 [name] => artem
 [age] => 19
 [hobby] => gym
)

Универсальный сеттер

```
$person = new Person();
$person->setProperty( name: 'Victor', age: 20, hobby: 'swimming');
$person2 = new Person();
$person2->setProperty( name: 'Victor', age: 20, hobby: 'swimming');
$person3 = new Person();
$person3->setProperty( name: 'Victor', age: 20, hobby: 'swimming');
```

Но теперь нам нужно вот так выводить для каждого объекта а если у нас их 1000 мы просто не сможем

Именно для таких вещей и придумали конструктор

```
class Person{
    public $name;
    public $age;
    public $hobby;

    public function __construct($name, $age, $hobby){
        $this->name = $name;
        $this->age = $age;
        $this->hobby = $hobby;
    }
}
```

Вот так выглядит конструктор

Отличие от обычного сеттера в том что он запускается АВТОМАТИЧЕСКИ при создании объекта (можно проверить добавив echo)

```
<?php

class Person{
    public $name;
    public $age;
    public $hobby;

    public function __construct($name, $age, $hobby){
        $this->name = $name;
        $this->age = $age;
        $this->hobby = $hobby;
    }
}

$person = new Person('artem', 19, 'gym');
print_r($person);
```

Теперь все выглядит вот так И У КЛАССА ПОЯВИЛИСЬ КРУГЛЫЕ СКОБКИ

```

1  <?php
2
3  class Magazine{
4      public $goods = [];
5      public $price;
6
7      public function __construct($goods, $price)
8      {
9          $this->goods = $goods;
10         $this->price = $price;
11     }
12 }
13
14 $shava = new Magazine(['one'=>'1', 2, 'm100'], 100);
15 print_r($shava);

```

ВЫХОДНЫЕ ДАННЫЕ КОНСОЛЬ ОТЛАДКИ ТЕРМИНАЛ

```

Magazine Object
(
    [goods] => Array
        (
            [one] => 1
            [0] => 2
            [1] => m100
        )

    [price] => 100
)

```

Интересный пример

```

13
14 $shava = new Magazine(['one'=>'1', 2, 'm100'], 100);
15 foreach($shava->goods as $good){
16     echo $good . "\n";
17 }

```

ВЫХОДНЫЕ ДАННЫЕ КОНСОЛЬ ОТЛАДКИ ТЕРМИНАЛ

```

1
2
m100

```

Наследование в php

```

<?php
class person{
    public $name;
    public $age;
    public $job;
}

class Developer{
    public function work(){
        echo 'yes - dev';
    }
}

class Manager{
    public function work(){
        echo 'yes - man';
    }
}

class Director{
    public function work(){
        echo 'yes - dir';
    }
}

```

Представим что у нас есть 1000 классов в которых все св-ва одинаковые и чтобы не копировать 1000 раз придумали НАСЛЕДОВАНИЕ

```
<?php

class Person{
    public $name;
    public $age;
    public $job;

    public function __construct($name, $age, $job)
    {
        $this->name = $name;
        $this->age = $age;
        $this->job = $job;
    }
}

class Developer extends Person{
    public function work(){
        echo 'yes - dev';
    }
}

class Manager{
    public function work(){
        echo 'yes - man';
    }
}

$develop = new Developer('artem',19,'dev');
print_r($develop);
```

Теперь благодаря этому ключевому слову у нас есть все что и в классе Person

Конструктор в классах наследниках

Представим ситуацию когда нам нужен конструктор в определенном объекте когда у родительского класса уже есть конструктор

```

3  class Person{
4      public $name;
5      public $age;
6      public $job;
7
8      public function __construct($name, $age, $job)
9      {
10         $this->name = $name;
11         $this->age = $age;
12         $this->job = $job;
13     }
14 }
15
16 class Manager extends Person{
17     public $time;
18
19     public function work(){
20         echo 'yes - man';
21     }
22
23     public function __construct($name, $age, $job, $time){
24         parent::__construct($name, $age, $job);
25         $this->time = $time;
26     }
}

```

ВЫХОДНЫЕ ДАННЫЕ КОНСОЛЬ ОТЛАДКИ ТЕРМИНАЛ

```

Manager Object
(
    [time] => 20
    [name] => vlad
    [age] => 19
    [job] => stone
)

```

```

    public function __construct($name, $age, $job, $time){
        parent::__construct($name, $age, $job);
        $this->time = $time;
    }
}

$manager = new Manager('vlad', 19, 'stone', 20);
print_r($manager);

```

Так мы создаем конструктор внутри конструктора родителя

Абстрактные классы

Нужны абстрактные классы для того чтобы показать что класс только для того чтобы от него что-то наследовали

Можно запретить создание объектов от этого класса, только КЛАССЫ НАСЛЕДНИКИ

```

class Person
{
    public $name;
    public $age;
    public $job;
}

```

Просто оставлять свойства так нельзя это неправильно

Если у класса есть св-ва и методы которым ничего не назначено - его нужно делать абстрактным

```
abstract class Person{
    public $name;
    public $age;
    public $job;

    public function __construct($name, $age, $job)
    {
        $this->name = $name;
        $this->age = $age;
        $this->job = $job;
    }
}
```

Так мы сделали класс абстрактным и теперь мы обязали делать у него наследников и запретили создавать объекты и инициализировать св-ва

Статические методы и свойства в классах

Представим что нам не нужны никакие объекты и никакое наследование от нашего класса, мы просто хотим использовать функции класса в разных местах проекта

Для этого нам нужно дать возможность обращение к элементом класса без создания объектов

И для этого существует **static**

```
.. > Users > admin > Desktop > test.php > ...
1  <?php
2
3  abstract class Person{
4      public $name;
5      public $age;
6      public $job;
7
8      public static function getSum($a,$b){
9          $sum = $a+$b;
10         return $sum;
11     }
12 }
13
14 echo Person::getSum(1,2);
```

ВЫХОДНЫЕ ДАННЫЕ КОНСОЛЬ ОТЛАДКИ ТЕРМИНАЛ

3

ПРИ УКАЗАНИИ КЛЮЧЕВОГО СЛОВО СТАТИК МЫ МОЖЕМ ОБРАЩАТЬСЯ К МЕТОДУ КЛАССА НЕПОСРЕДСТВЕННО ЧЕРЕЗ КЛАСС А НЕ ЧЕРЕЗ ОБЪЕКТ

```
1 <?php
2
3 class Person{
4     public static $name;
5     public $age;
6     public $job;
7
8     public static function getSum($a,$b){
9         $sum = $a+$b;
10        return $sum;
11    }
12
13    public static function setName($name){
14        self::$name = $name;
15    }
16 }
17
18 echo Person::getSum(1,2);
19 echo Person::setName('vlad');
20 echo Person::$name;
```

ВЫХОДНЫЕ ДАННЫЕ КОНСОЛЬ ОТЛАДКИ ТЕРМИНАЛ
Vlad

Self::\$name - обращение к переменной САМОГО КЛАССА (\$this->name не работает тк по сути объекта то у нас нет (класс и есть объект))

```
$person = new Person();
$person::setName(name: 'Gleb');
echo Person::$name;
```

Вот такой конструкцией мы вызываем у \$person класс Person и у этого класса метод
(если указать абстракт то и объекты создавать нельзя будет никак даже такие липовые)

Интерфейсы в php

Интерфейсы в php - **список** методов которые необходимо реализовать

По сути так мы обязуем реализовать какие-либо методы

```
<?php

interface Spans{
    public function run();
    public function jump();
}

'Gymnast' does not implement methods 'run', 'jump', 'stop' intelephense(1037)
Просмотреть проблему Исправления недоступны

class Gymnast implements Spans{

}
```

Implements - реализует

Когда мы создали класс который реализует наши интерфейсы он будет выдавать ошибку если интерфейсных методов в классе нет

Чтобы ошибки не было нужно просто реализовать методы интерфейса

Интерфейсы нужны для того чтобы убрать неопределенность и дать понять программисту какие методы точно должны быть реализованы

Уровни доступа в php: private, protected, public

Эти уровни доступа предназначены для других классов наследников либо других файловых php

Это просто уровни доступа между классами

Нужно это для регулирования удобства (какие методы будут высвечиваться какие не будут что нужно а что нет в других классах)

```
class Rosa extends Plant
{
}

$rosa = new Rosa(title: 'Rosa', color: 'Red', season: 'spring');

echo $rosa->;
```

В классе наши св-ва PUBLIC

```
echo $rosa->;
title
color
```

Здесь у нас модификатор PRIVATE

PRIVATE - доступен только в классе (не в объектах не в наследниках нигде больше)

PUBLIC - доступен везде (и в объектах и в наследниках и в др файлах)

PROTECTED - доступен только в классе наследнике и в самом классе (не в объекте классах)

S.O.L.I.D

Принцип единственной ответственности (single responsibility principle)

/*При создании программных систем использование принципов SOLID способствует созданию такой системы, которую будет легко поддерживать и расширять в течение долгого времени[3]. Принципы SOLID – это руководства, которые также могут применяться во время работы над существующим программным обеспечением для его улучшения, например, для удаления «дурно пахнущего кода». */

/* принцип ООП, обозначающий, что каждый объект должен иметь одну ответственность и эта ответственность должна быть полностью инкапсулирована в класс. Все его поведения должны быть направлены исключительно на обеспечение этой ответственности. */

```
//class Worker{
//    protected $old;
//    protected $name;

//    public function __construct($old, $name)
//    {
//        $this -> old = $old;
//        $this -> name = $name;
//        echo "работник создан \n";
//    }

//    public function getName(){
//        return $this -> name;
//    }

//    public function getOld(){
//        return $this -> old;
//    }
// }

// class deleteWorker{
//     удалить работника
// };

// $man = new Worker(20, 'артем');
```

```
// echo $man -> getName() . " - ";
// echo $man -> getOld();

// -----
// class Worker {
//     public $name;
//     public $old;
//     public $cost;

//     public function getName(){
//         echo $this->name;
//     }

//     public function getOld(){
//         echo $this->old;
//     }

//     public function getCost(){
//         echo $this->cost;
//     }
// };

// $ivan = new Worker();

// $ivan -> name = "ivan";
// $ivan -> old = 22;
// $ivan -> cost = 1000;
```

```
// $ivan->getName();
// echo "\n";
// $ivan->getOld();
// echo "\n";
// $ivan->getCost();

// _____

// class Worker {
//     private $name;
//     private $old;

//     public function setName($name){
//         $this -> name = $name;
//     }

//     public function getName(){
//         return $this -> name;
//     }

//     public function setOld($old){
//         $this -> old = $old;
//     }

//     public function getOld(){
//         return $this -> old;
//     }
// }
```

```

// $man = new Worker();

// // $man -> name = "ivan"; не работает тк private

// $man -> setName("artem"); // работает
// echo $man -> getName();

// -----
// class People {
//     public function __construct(protected $lags, protected $hands, private $ace){
//         echo "Человек создан и у него {$lags} ноги, {$hands} руки, {$ace} лицо";
//     }
// }

// $artem = new People(lags: 2, hands: 2, ace: 1); // начиная с php 8

// -----

```

Принцип открытости/закрытости (open-closed principle)

```

// Принцип открытости/закрытости (open-closed principle)
// «программные сущности ... должны быть открыты для расширения, но закрыты для модификации»
// простыми словами – нужно не изменять работающий класс а дополнять его

/*открыты для расширения: означает, что поведение сущности может быть расширено путём создания новых типов
сущностей.закрыты для изменения: в результате расширения поведения сущности, не должны вноситься изменения
в код, который эту сущность использует. */

/* Идея была в том, что однажды разработанная реализация класса в дальнейшем требует только
исправления ошибок, а новые или изменённые функции требуют создания нового класса.
Этот новый класс может переиспользовать код исходного класса через механизм наследования.
Производный подкласс может реализовывать или не реализовывать интерфейс исходного класса.

Определение Мейера поддерживает идею наследования реализации. Реализация может быть переопределена
через наследование, но спецификации интерфейса могут измениться. Существующая реализация
должна быть закрыта для изменений, но новые реализации не обязаны использовать
существующий интерфейс. */

// простыми словами – не изменяем старую сущность а добавляем новую которая расширит старую
// также есть и полиморфный признак открытости, когда мы выносим в метод в интерфейс и ПЕРЕОПРЕДЕЛЯЕМ его
// есть класс оружие, интерфейс атак и 3 класса – нож ствол и лук, а не 1 класс оружие
// СОЗДАТЬ КЛАСС ЛУК УНАСЛЕДОВАТЬ ЕГО ОТ КЛАССА ОРУЖИЕ И ПЕРЕОПРЕДЕЛИТЬ МЕТОД интерфейса

```

Принцип подстановки Лисков (Liskov substitution principle)

```
/* Принцип подстановки Лисков (Liskov substitution principle)
«функции, которые используют базовый тип, должны иметь возможность использовать подтипы базового типа не зная об этом». См. также контрактное программирование. */

//наследуемый класс должен дополнять а не замещать поведение базового класса (контракты)

//если у нас есть функция, которая работает с объектом базового класса и от этого класса
//мы наследуем другой класс, то функция должна работать с объектом нового класса точно так же
//как и с базовым классом

//если в базовом классе есть метод который не может быть в наследнике – это ошибка!!

//есть класс персонаж – от него идет класс кодер и если в классе герой есть метод спать
//а кодер спать не может – ошибка!

//переопределять метод и ложить в него NULL НЕ НАДО тк мы ожидаем совсем другое поведение

//при правильном ПРОЕКТИРОВАНИИ такой метод не нужно держать в базовом классе а прописать в
//наследнике и если эти методы есть в других классах наследовать теперь от него

// ДУМАТЬ НАДО НА ЭТАПЕ ПРОЕКТИРОВАНИЯ

//Пример: есть класс базы данных и 2 класса реляц и нереляц, которые наследуют базовый класс
//но отличаются по методам, и уже от них наследуются другие классы
```

Принцип разделения интерфейса (interface segregation principle)

```
/*Принцип разделения интерфейса (interface segregation principle)
«много интерфейсов, специально предназначенных для клиентов, лучше, чем один интерфейс общего назначения»*/,
//нельзя заставлять клиента реализовывать интерфейс который ему не нужен. Лучше сделать
//другой интерфейс
```

Принцип инверсии зависимостей (dependency inversion principle)

```
/* Принцип инверсии зависимостей (dependency inversion principle)

/*
принцип объектно-ориентированного программирования, суть которого состоит в том,
что классы должны зависеть от абстракций, а не от конкретных деталей[1].
Используется для минимизации зацепления в компьютерных программах. */

// модули более высокого уровня не должны зависеть от методов более низкого

//из-за одной детали не нужно менять рабочих и менять электричество

//нужна промежуточная абстракция которая не зависит от станка и пульт управления чтобы
//не менять работника

//нужно добавлять как можно больше модулей

//решение: сделать общий интерфейс для набора классов которые будут его реализовывать
// А ЕЩЕ ЛУЧШЕ ВВЕСТИ АБСТРАКЦИЮ ПРОМЕЖУТОЧНУЮ ГДЕ С ПОМОЩЬЮ МЕТОДА МЫ ВСЕГДА МОЖЕМ ПОМЕНЯТЬ
// ТО КАК БУДЕТ РАБОТАТЬ ПРОГРАММА (пуль) (конструктор сервис в ларавел)
```

Часть 2. Git

2.1 Какие задачи решает Гит и для чего он нужен?

Ситуация 1:

Уже есть какой-то готовый проект и я хочу его улучшить, но есть опасения что все сломаю. Можно взять и скопировать проект а можно создать **систему версий**

Ситуация 2:

Над проектом работают несколько людей. Можно выкладывать в гугл диск, но если 1 файл меняли несколько людей одновременно или работают много людей? Начинается каша

Решение:

Для решения этих ситуаций и был придуман гит. В системе контроля версий можно создавать версии проектов, выкладывать изменения, контролировать работу в команде и избавиться от критических ошибок. И создание альтернативных проектов не занимает места и времени. Также можно контролировать изменения, новые строки, что добавлять а что

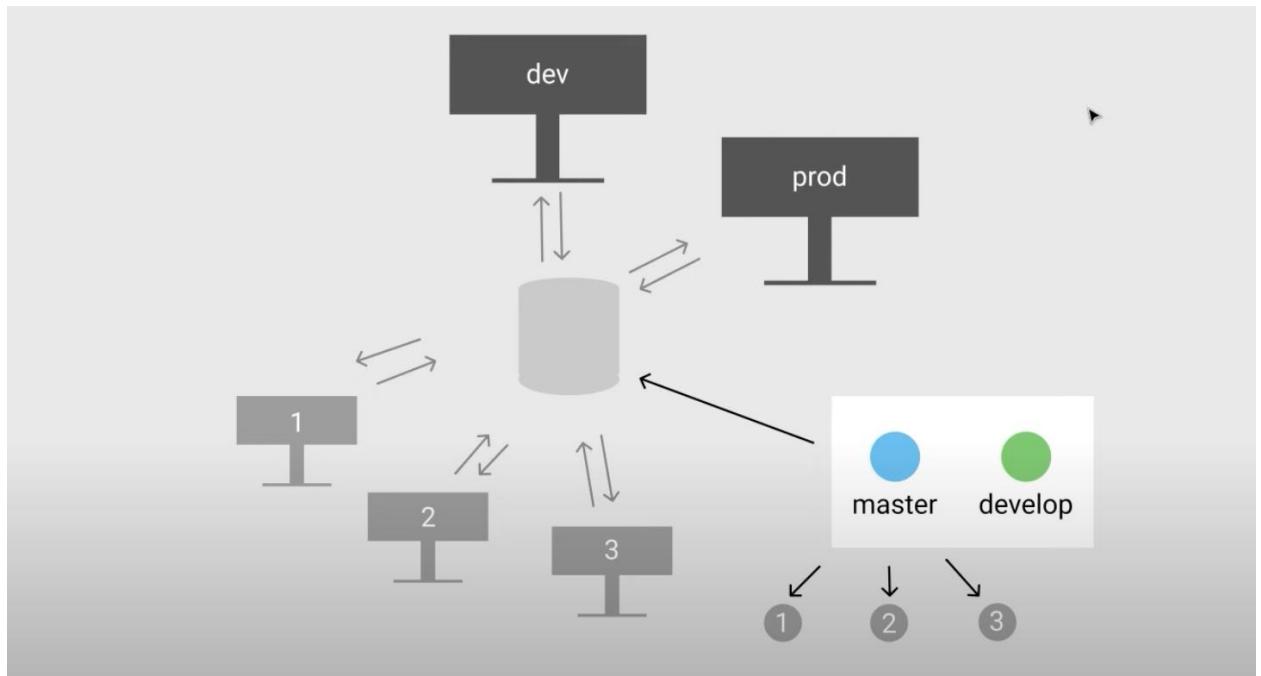
убирать. В общем гит – создание альтернативных вариантов проекта и новых версий.

2.2 Основные команды в Git

Терминология:

1. Git clone – скачивание проекта себе
2. Git add – добавить изменения, определение изменений (к себе)
3. Git commit – зафиксировать изменения
4. Git push – отправить изменения, выложить в основу
5. Git pull – скачать проект с изменениями (то чего нет у нас)
6. Git merge – объединить изменения одного файла из двух источников (смержить, соединить, слить, объединить изменения файла). Выбираем в окне что оставить а что убрать.
7. Git branch – создать ветку, аналог нынешнего проекта, альтернативу (исходный проект – ветка мастер). Ветка нужна для работы, изменения чего-то для последующего слияния с мастер веткой.
8. Git checkout – переключение между ветками (отображение варианта проекта)
9. Master ветка – основной исходный проект (оригинал), только она постоянна, остальные ветки заливаются в мастер, устойчивое состояние. Тут всегда все стабильно и тут не работают а только заливают.

2.3 Методология git flow. Наиболее популярный вариант разработки с Git



Обычно при разработке в команде используется такая методология и такой подход к разработке, механизм разработки

Строение:

1. Цилиндр – это хранилище(гитхаб гитлаб) где присутствуют 2 ветки (2 варианта одного проекта)
2. Прод комп – продакшен, основной сайт, который работает, в него заходят люди, он работает (здесь отображена ветка мастер)
3. Дев комп – абсолютная копия прода, но обычные пользователи туда зайти не могут, только разработчики (здесь будет ветка девелоп). И всегда прежде чем выложить изменения на прод нужно протестить на деве совокупность изменений. После теста все идет на мастер ветку и потом уже на прод
4. Маленькие компы – разработчики. Прежде чем выполнить задачу разрабы создают для себя свою ветку от мастера и работают с ней. После локальных тестов заливают в хранилище на ветку дева и далее ветка идет в дев среду. Под каждую новую задачу разрабы создают новую ветку, после работы и тестов в себя выкладывают в дев и после тестов на дев изменения идут на прод. После выполнения своих изменений ветка удаляется
5. Ветка девелоп создается в начале создания проекта от ветки мастер

2.4 Хранилища Github и Gitlab

Это хранилища проектов, их нужно заполнить по советам чтобы быть крутым.

Также это децентрализованные хранилища

2.5 Создаем проект. Инициализируем Git. Первый commit и add в Git

Цель: создать проект -> загрузить в хранилище(с ветками мастер и дев) -> поменять проект -> залить изменения

1. Инициализируем гит

```
saint@MacBook-Pro-Artem git % git init
hint: Using 'master' as the name for the initial branch. This default branch nam
e
hint: is subject to change. To configure the initial branch name to use in all
hint: of your new repositories, which will suppress this warning, call:
hint:
hint:   git config --global init.defaultBranch <name>
hint:
hint: Names commonly chosen instead of 'master' are 'main', 'trunk' and
hint: 'development'. The just-created branch can be renamed via this command:
hint:
hint:   git branch -m <name>
Initialized empty Git repository in /Users/saint/Desktop/git/.git/
saint@MacBook-Pro-Artem git %
```

Создается папка .git которая следит за всем что есть в папке

2. Команда git status проверяет текущее состояние гита проекта (добавлены ли папки в гит, внесены ли изменения)

3. Коммитим изменения

```
saint@MacBook-Pro-Artem git % git commit -m "first commit"
[master (root-commit) d36a942] first commit
  Committer: Artem Muzychuk <saint@MacBook-Pro-Artem.local>
Your name and email address were configured automatically based
on your username and hostname. Please check that they are accurate.
You can suppress this message by setting them explicitly. Run the
following command and follow the instructions in your editor to edit
your configuration file:

  git config --global --edit

After doing this, you may fix the identity used for this commit with:

  git commit --amend --reset-author

  1 file changed, 0 insertions(+), 0 deletions(-)
  create mode 100644 README.md
saint@MacBook-Pro-Artem git %
```

4. Пушим в гитхаб

Сначала создадим токен

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
<https://docs.github.com/en/authentication/keeping-your-account-and-data-secure/creating-a-personal-access-token>

<https://stackoverflow.com/questions/47465644/github-remote-permission-denied> -
сменна аккаунта

Добавим репозиторий командой или клонируем готовый

git remote add origin <https://github.com/ArtemMusic/test.git>
git clone https://github.com/ArtemMusic/test.git

Далее запишем на гитхаб

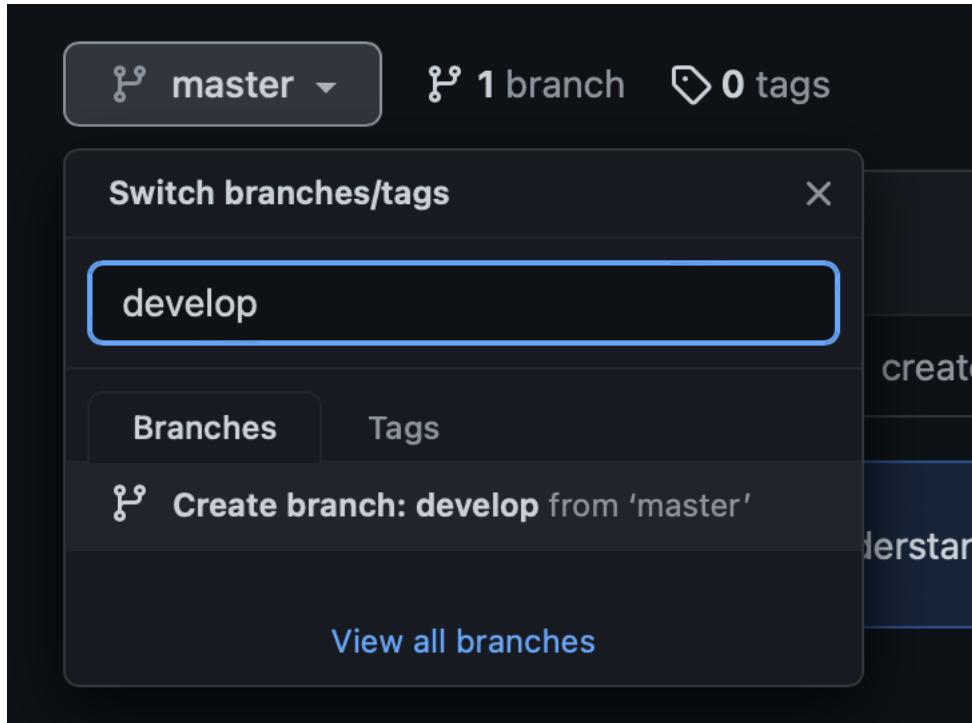
git push -u origin master

...or create a new repository on the command line

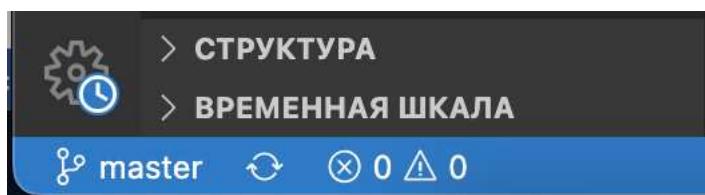
```
echo "# test" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M master
git remote add origin https://github.com/ArtemMusic/test.git
git push -u origin master
```

2.6 Git clone и .gitignore. Клонируем проект с github

Теперь любые изменения веток, такие как создание новой ветки, слияние веток и тд нужно делать НЕ ЧЕРЕЗ КОНСОЛЬ А ЧЕРЕЗ ХАБ



Вот тут (создать ветку дев от мастера)



Vs code ветки (появятся после гит fetch) (после обновления синхронизации .git)

- saint@MacBook-Pro-Artem test % git branch
* master
- saint@MacBook-Pro-Artem test %

Так можно проверить ветки

- saint@MacBook-Pro-Artem test % git fetch
Из https://github.com/ArtemMusic/test
* [новая ветка] develop -> origin/develop
- saint@MacBook-Pro-Artem test % git branch
* master

Так можно скачать ветки и проверить их

Но как можно заметить новой ветки не появилось ТК ВЫТЕЯНУЛИСЬ
УДАЛЕННЫЕ ВЕТКИ И НАША ЗАДАЧА СДЕЛАТЬ ВЕТКИ
ЛОКАЛЬНЫМИ

- saint@MacBook-Pro-Artem test % git branch -r
 - origin/develop
 - origin/master
- saint@MacBook-Pro-Artem test % █

Вот наши удаленные ветки, нужно их стянуть и сделать локальными

- saint@MacBook-Pro-Artem test % git checkout -b develop origin/develop
branch 'develop' set up to track 'origin/develop'.
Переключились на новую ветку «develop»
- saint@MacBook-Pro-Artem test % █

Этой командой мы создали локальную ветку develop из удаленной ветки develop в проекте origin

- saint@MacBook-Pro-Artem test % git branch
 - * develop
 - master

Теперь ветки есть

- saint@MacBook-Pro-Artem test % git checkout master
Переключились на ветку «master»
Эта ветка соответствует «origin/master».

Переключение между ветками

- saint@MacBook-Pro-Artem test % git remote -v
origin https://github.com/ArtemMusic/test.git (fetch)
origin https://github.com/ArtemMusic/test.git (push)

Так можно посмотреть какие есть имена УДАЛЕННЫХ ХРАНИЛИЩ



Тут можно сделать fetch через vscode

ЕСЛИ МЕНЯТЬ ЧТО-ТО НА ЛОКАЛЬНЫХ ВЕТКАХ, ТО В УДАЛЕННЫХ
НИЧЕГО НЕ ИЗМЕНИТЬСЯ

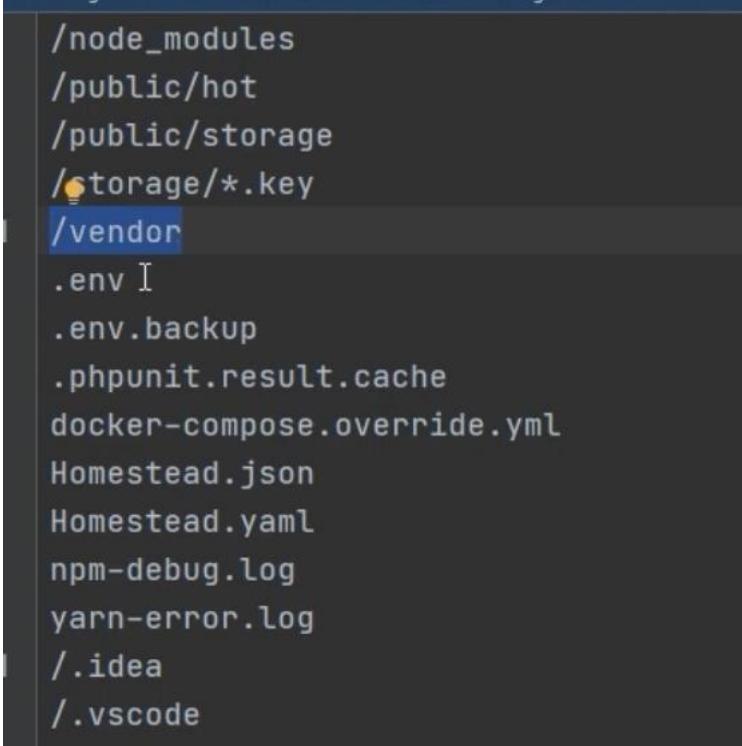
Чтобы это изменить нужно сделать пуш

2.7 Git clone и .gitignore. Клонируем проект с github и gitlab

Проиграем ситуацию прихода нового сотрудника

Для начала склонируем проект или запулим изменения

Gitignore – файл с конфигурациями игнорирования



```
/node_modules
/public/hot
/public/storage
/storage/*.key
/vendor
.env
.env.backup
.phpunit.result.cache
docker-compose.override.yml
Homestead.json
Homestead.yaml
npm-debug.log
yarn-error.log
.idea
.vscode
```

ЕСЛИ МЫ КЛОНИРУЕМ Laravel проект, то прежде нужно добавить все эти файлы в папку исключения а потом уже после скачивания проекта выполнить команду composer install тк в композере храняться все необходимые расширения далее сделать npm install для node
Далее нужно на основе .envexample создать свой .env

2.8 Создание задачи(issue) в github и gitlab с веткой и fetch

Клонировали репозиторий – от ветки МАСТЕР создали свою ветку и работаем в ней – отправляем изменения на хранилище на ветку ДЕВ - если все работает хорошо то отправляем на мастер

При этом ветки сначала создаем в хранилище и потом вытягиваем в локальную машину

Задание:

Представим что нам дали задачу создать какой-то контроллер

Чтобы дать задание:

The screenshot shows the GitHub web interface for creating a new issue. At the top, there is a navigation bar with links for Issues, Pull requests, Actions, Projects, Wiki, Security, Insights, and Settings. A prominent banner at the top says "Label issues and pull requests for new contributors" with a "Dismiss" button. Below the banner, there are filters set to "is:issue is:open". On the right, there are buttons for Labels (9), Milestones (0), and New issue. The main area is titled "Create controller" and contains a text input field with the placeholder "create posts controller". There are "Write" and "Preview" tabs above the input field. Below the input field, there is a note about attaching files and a "Submit new issue" button. A small note at the bottom reminds users to follow GitHub Community Guidelines.

Так можно создать задачу себе

This screenshot shows the sidebar of the GitHub issue creation interface. It includes sections for Assignees (with a note to assign yourself), Labels (none yet), Projects (none yet), and Milestone (no milestone). Below these, there is a section for "Development" which shows branches and pull requests linked to the issue. At the bottom, there is a "Helpful resources" section linking to the GitHub Community Guidelines.

Тут можно передать задачу кому-то или указать авторство

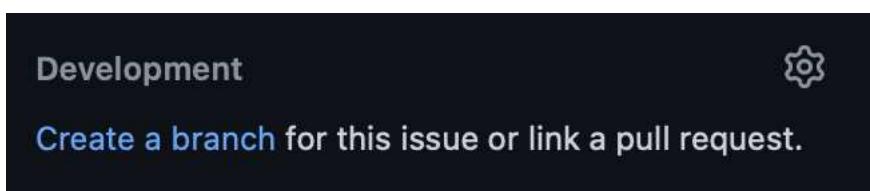
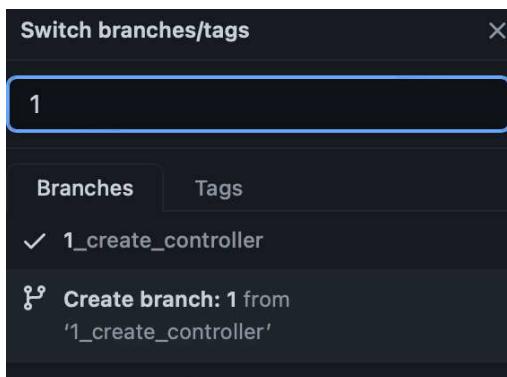
Create controller #1

 Open ArtemMusic opened this issue now · 0 comments

Задача открыта

Приступим к выполнению

1. Создадим ветку с именем задачи



Либо можно создать ветку от этой задачи (нинада)

2. Получаем удаленную ветку

```
● saint@MacBook-Pro-Artem test % git fetch
  Из https://github.com/ArtemMusic/test
    * [новая ветка]      1_create_controller -> origin/1_create_controller
○ saint@MacBook-Pro-Artem test %
```

3. Делаем ветку локальной путем переключения на нее

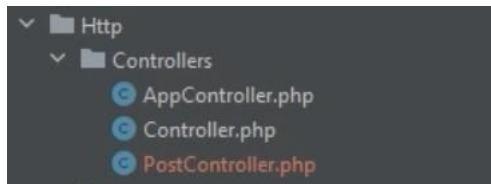
```
○ saint@MacBook-Pro-Artem test % git checkout -b 1_create_controller origin/1_create_controller
```

Либо через vs code просто переключением на нее

```
● saint@MacBook-Pro-Artem test % git branch
* 1_create_controller
  develop
  master
```

4. Выполняем задачу

...



Добавим в индекс и закоммитим

5. Коммитим и отправляем либо через консоль либо через vscode

2.9 Pull request в github

6. Создадим пул реквест для закрытия задачи

Label issues and pull requests for new contributors
Now, GitHub will help potential first-time contributors discover issues labeled with [good first issue](#) Dismiss

1_create_controller had recent pushes 6 minutes ago Compare & pull request

Filters is:pr is:open Labels 9 Milestones 0 New pull request

0 Open 0 Closed Author Label Projects Milestones Reviews Assignee Sort

Далее новый пул реквест

Compare and review just about anything
Branches, tags, commit ranges, and time ranges. In the same repository and across forks.

Example comparisons

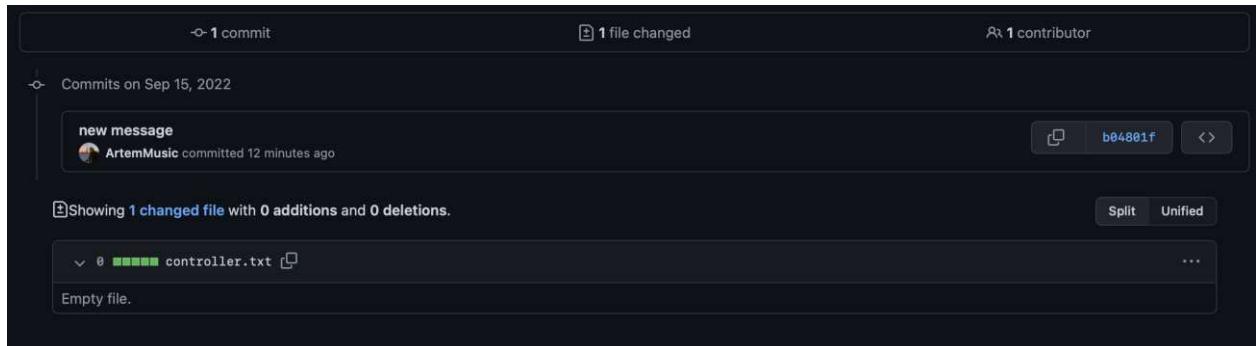
Comparison	Time Ago
1_create_controller	8 minutes ago
develop	1 hour ago
master@{1day}...master	24 hours ago

Выбираем ветку задачи

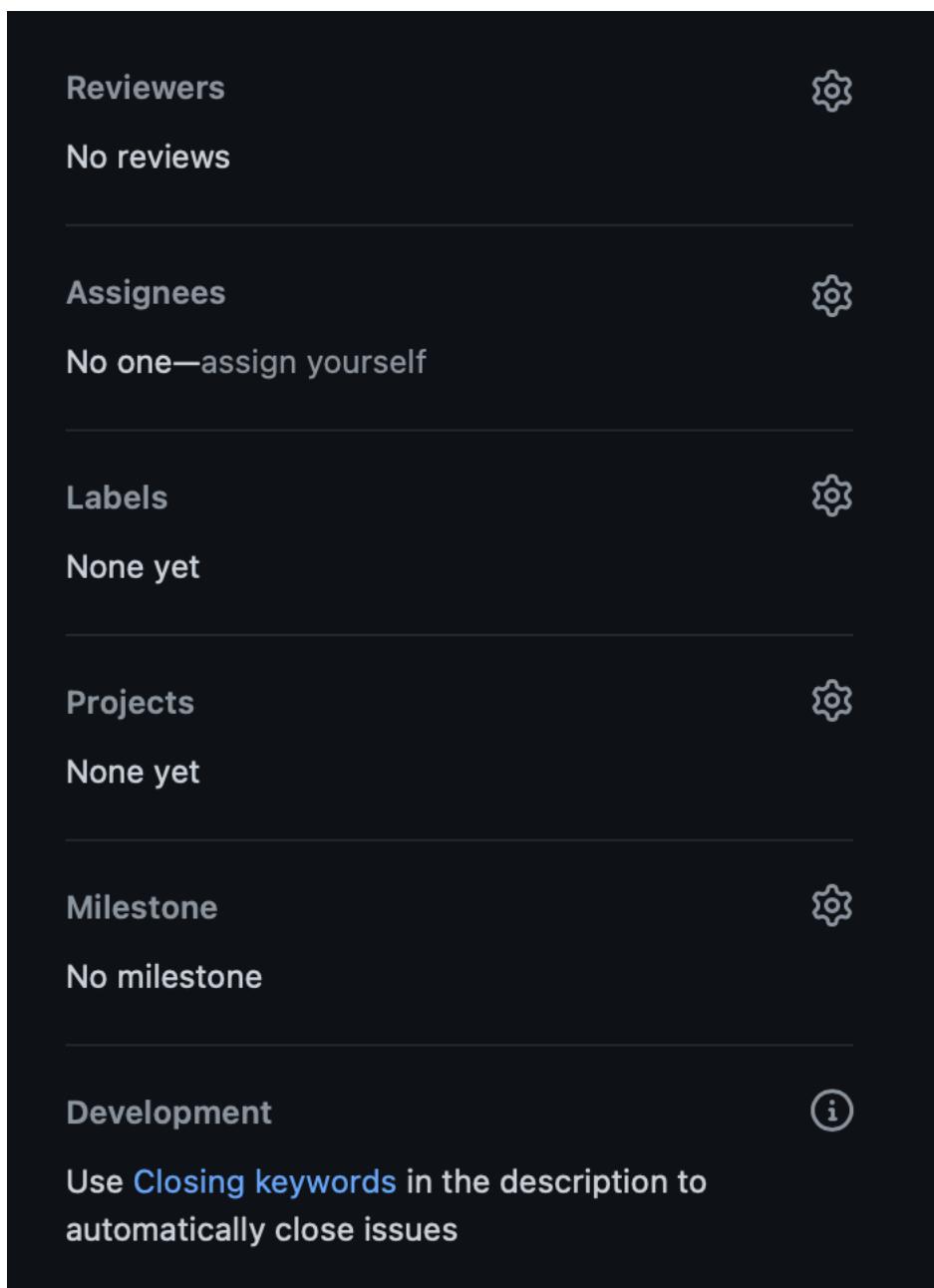
base: develop compare: 1_create_controller Able to merge. These branches can be automatically merged.

База – ветка куда сливаются компэйр – сливаемая (не забываем что сначала на девелоп)

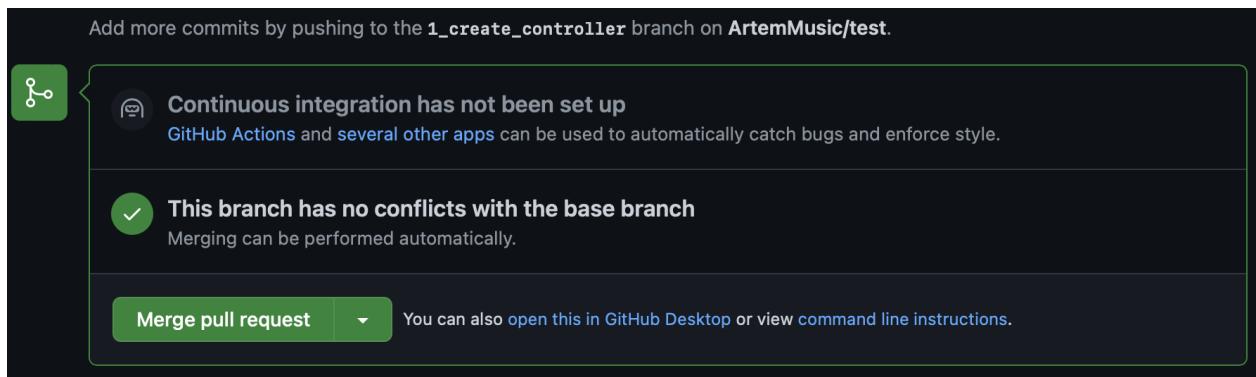
Галочка значит что конфликтов слияния нет



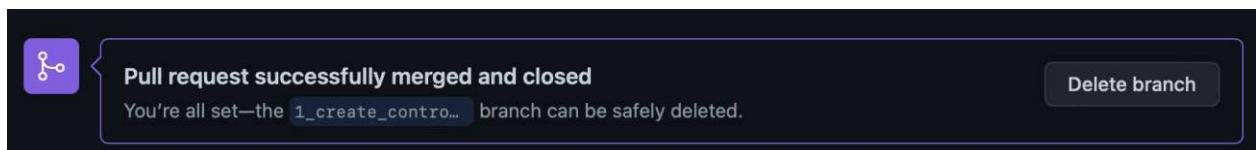
Тут находятся изменения



Также можно выюрать проверяющего и автора



Далее можем смержить



Сообщение о готовности

7. После тестов на деве можно мерджить с мастером

Тут уже как пункт 6 только мастер – девелоп

8. Закрываем задачу с прикреплением реквеста

скудею сщтекшддукуюче #2

Open ArtemMusic opened this issue 11 minutes ago · 0 comments · Fixed by #3

ArtemMusic commented 11 minutes ago

No description provided.

ArtemMusic linked a pull request now that will close this issue

vse ok #3 Merged

Write Preview

Leave a comment

Attach files by dragging & dropping, selecting or pasting them.

Close issue Comment

Remember, contributions to this repository should follow our GitHub Community Guidelines.

Assignees: None yet

Labels: None yet

Projects: None yet

Milestone: None

Development: Successfully merging a pull request may close this issue.

vse ok ArtemMusic/test

Notifications: Customize Unsubscribe

You're receiving notifications because you're watching this repository.

2.10 SSH key для github

Принцип: создаем уникальный ключ машины, привязываем его к гиту, и когда мы что-то делаем система будет проверять правда ли что мой ключ совпадает с ключом гита и если да то пропускает

Просто читаем доки как сгенерировать ssh ключ

Удаляем remote пути(удаленные хранилища)

git remote remove gitlab

git remote remove github

Создаем новые remote

git remote add github тут_ssh_ссылка_c_github

git remote add gitlab тут_ssh_ссылка_c_gitlab

Далее открываем git bash

Создаем ключ в git bash

ssh-keygen -t ed25519

Нажимаем enter

Далее вносим секретную фразу

Повторяем секретную фразу

Добавляем ssh ключ в github

В phpstorm делаем фетч

Затем переходим на ветку мастер с github

Gitlab при этом отдаст ошибку, что нет доступа для фетча

Делаем git pull

Затем добавляем контроллер

Делаем commit и push

Затем добавляем ssh ключ в gitlab

В phpstorm делаем фетч

Затем переходим на ветку мастер с gitlab

Делаем git pull

Затем добавляем контроллер

Делаем commit и push

2.11 Merge conflict в git. Git stash. Git reset

1. Получаем хэш-код коммита, к которому хотим вернуться.
2. Заходим в папку репозитория и пишем в консоль:

```
$ git reset --hard a3775a5485af0af20375cedf46112db5f813322a  
$ git push --force
```

Часть 3. Laravel (база)

3.1 Введение

Laravel - фреймворк. Система, которая облегчит разработку. Это уже разработанная архитектура программы с готовыми решениями многих проблем. Инструмент для эффективной разработки.

Composer - пакетный менеджер для php. Нужен для подключения и обновления пакетов и плагинов. Это менеджер зависимостей. (библиотеки, улучшающие разработку)

3.1.1 Этапы создания проекта

1. Качаем и устанавливаем composer (перед этим php и т.д. и т.п.)

2. Переходим в папку, где будет проект и пишем

```
composer create-project laravel/laravel two (two - имя)
```

3. Устанавливаем dbal

```
composer require doctrine/dbal
```

4. Коннектимся к бд



5. Убираем защиту

A screenshot of a code editor showing the RouteServiceProvider.php file from a Laravel project named 'two'. The file contains PHP code for a service provider. The code includes imports for RouteServiceProvider, RouteServiceProvider, and RouteServiceProvider. It defines a public const HOME = '/home'; and a protected \$namespace = 'App\\Http\\Controllers';. The code editor interface shows the file structure on the left and the code on the right.

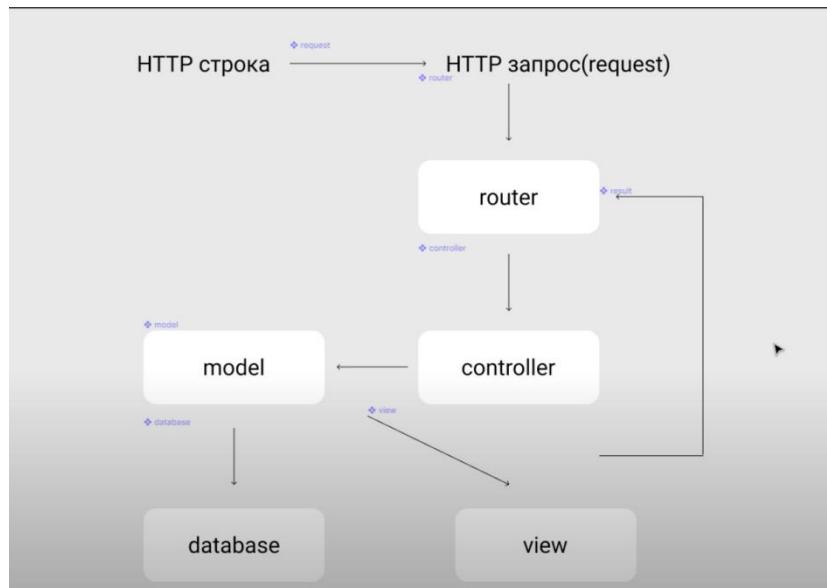
3.2 Model-View-Controller

Model-View-Controller — схема разделения (шаблон проектирования) данных приложения и управляющей логики на три отдельных компонента: модель, представление и контроллер — таким образом, что модификация каждого компонента может осуществляться независимо. Модель предоставляет данные и реагирует на команды контроллера, изменения своё состояние.

MVC паттерн – повторимая конструкция решения конкретной проблемы проектирования

Такой пример – работа абсолютно любого сайта основанного на фреймворках. Именно так принято разрабатывать сайты.

Такой паттерн – победа над сложностью за счет разделения труда



Модель – ПРЕДСТАВЛЕНИЕ одного объекта для интерфейса php. Когда мы берем какой-то пост из бд, то аргументы сохраняются в результирующем объекте. И из одной модели можно будет через синтаксис php обращаться к аргументам объекта (название цена и т.д. и т.п.). У НАС БУДЕТ МОДЕЛЬКА СО СВ-МИ КОТОРЫЕ МОЖНО БРАТЬ

Все это – классы. И в разном классе разная логика

Контроллер – Мастер обработки http запросов.

Модель – реализация действий с объектами бд.

Роутер – список запросов и соответствий.

3.2.1 Этапы:

1. **HTTP строка → HTTP запрос (request).** Тут наш сайт по протоколу обращается к нашему серверу и говорит ему что нужно что-то достать
2. **HTTP запрос (request) → router(маршрутизатор).** Маршрутизатор работает как администратор в отеле, который при запросе ключа от комнаты ищет его на полках и дает ключ. Тут же наш роутер обращается к контроллеру, у которого просит принести ему этот ключ
3. **router(маршрутизатор) → контроллер.** Контроллер по приказу роутера идет к модели и просит уже его достать этот ключ в любом виде из базы данных.
4. **Контроллер – модель.** Модель обращается в базу данных за определенным ключом и выдает его контроллеру
5. **Контроллер - view.** На этом этапе контроллер подготавливает данных в пригодный вид и выдает результат роутеру
6. **Router – result.** Роутер выводит результат на сайт

Все это можно сделать и в 1 файле без такого разделения, но это ГЕМОРОЙ.
То же самое что жить всей семьей в одной комнате, можно, но не нужно.
ТАКАЯ МОДЕЛЬ ПРАВИЛЬНАЯ И ЭФФЕКТИВНАЯ

3.2.2 Http строка

HTTP — протокол прикладного уровня передачи данных (отправка запросов на сервер и возврат страниц)

3.2.3 Request(route)

```
MacBook-Pro-artem:first_proj muzica$ php artisan serve
Starting Laravel development server: http://127.0.0.1:8000
[Mon Jul 25 17:03:26 2022] PHP 8.1.8 Development Server (http://127.0.0.1:8000) started
[Mon Jul 25 17:03:32 2022] 127.0.0.1:50777 Accepted
[Mon Jul 25 17:03:32 2022] 127.0.0.1:50777 Closing
```

Данная команда запускает Laravel сервер и выдает окно прослушки действий

```
<?php

use Illuminate\Support\Facades\Route;

/*
|--------------------------------------------------------------------------
| Web Routes
|--------------------------------------------------------------------------
|
| Here is where you can register web routes for your application. These
| routes are loaded by the RouteServiceProvider within a group which
| contains the "web" middleware group. Now create something great!
|
*/

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

Routers-web.php – Список маршрутизаторов (все что мы вбиваем в командную строку и какой результат получим на запрос)

```
Route::get('/my_page', function(){
    return 'this is my page';
}); // класс route с приватным методом get(тип запроса) и аргументами uri адрес и
//анонимной функцией
```

В этот файл мы можем внести свой список маршрутизаторов при помощи данного класса

this is my page

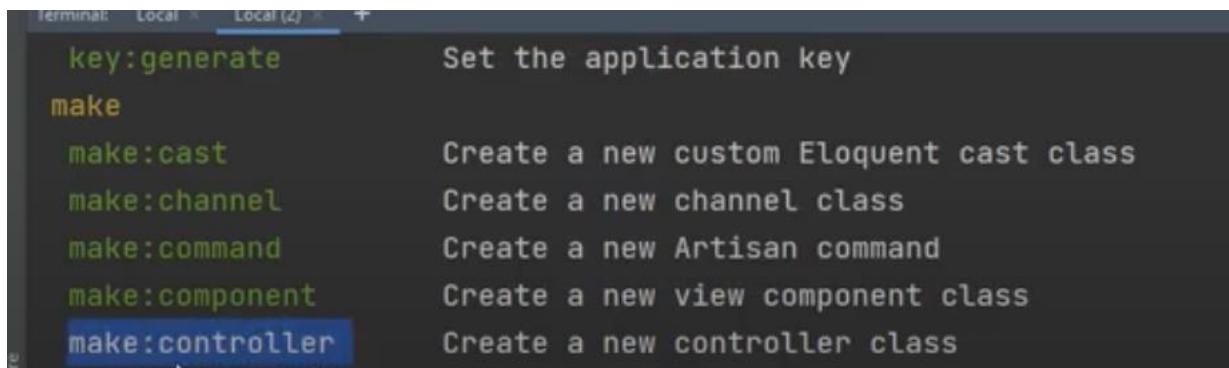
Роутер лишь маршрутизатор, который перенаправляет запросы.

3.2.4 Controller

Все что находится внутри проекта Laravel – сайт, где все взаимосвязано. Каждая папка и каждый класс. ЭТО ПРАВИЛЬНАЯ СТРУКТУРА САЙТА.

Контроллер – Мастер обработки http запросов.

Контроллер – это класс.



При помощи artisan можно создать контроллеры.

```
MacBook-Pro-artem:first_proj muzica$ php artisan make:controller -help
Description:
    Create a new controller class

Usage:
    make:controller [options] [--] <name>

Arguments:
    name           The name of the class

Options:
    --api          Exclude the create and edit methods from the controller.
    --type=TYPE    Manually specify the controller stub file to use.
    --force        Create the class even if the controller already exists
    -i, --invokable Generate a single method, invokable controller class.
```

Документация.

```
php artisan make:controller MyPageController
```

ИМЕНА контроллеров должны быть с большой буквы и верблюжьей нотацией. Это конвенция (правило, договор)

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class MyPageController extends Controller
{
    public function index(){
        return "My page";
    }
}
```

Вносим в наш новый контроллер(класс) новый метод, который мы сможем вызвать через роутер

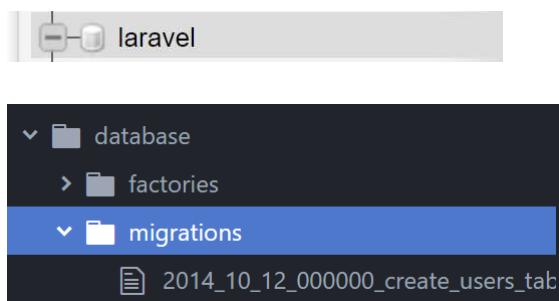
```
Route::get('/my_page', 'MyPageController@index');
```

3.2.5 Database (миграции)

Для работы с бд в laravel нужно создать миграцию для работы с контроллером

Миграция -> обычная таблица бд с колонками и типами данных

Атрибут - колонка. Все что хранится внутри таблицы - объекты, каждая колонка - атрибут объекта (колонки и атрибут синонимы)



Миграции находятся здесь

```
PS C:\OpenServer\domains\laravel\first-app> php artisan make:migration -help
```

Description:

Create a new migration file

Usage:

```
make:migration [options] [--] <name>
```

Arguments:

name	The name of the migration
------	---------------------------

Команда для создания миграций [--] опции можно не писать

Для того чтобы не сойти с ума при создании миграции нужно сразу создавать модель, связанную с миграцией и ларавел создаст все по конвенции

```
PS C:\OpenServer\domains\laravel\first-app> php artisan make:model Post -m
```

Создание модели сразу с миграцией (имя с большой буквы и при связывании в конце Model не писать и обязательно в ед. числе)

```
public function up()
{
    Schema::create('posts', function (Blueprint $table) {
        $table->id();
        $table->timestamps();
    });
}
```

Получаем миграцию (ключевое слово `create` в названии означает создание таблицы) и модель (ниже) и мы получаем даже класс с нашим именем

```
<?php
namespace App\Models;
use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;
class Post extends Model
{
    use HasFactory;
}
```

```

        */
    public function up()
    {
        Schema::create('posts', function (Blueprint $table) {
            $table->id();
            $table->timestamps();
        });
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::dropIfExists('posts');
    }
}

```

В нашей миграции в классе есть два метода up and down Нужны они для **накатывания и откатывания** миграций (так мы получаем и извлекаем данные)

При накатывании изменений все происходит в методе up
А при откате в методе down

- Аналогия ктрл з ктрл у
- Аналогия бекапов (по сути, это они и есть)

```

public function up()
{
    Schema::create('posts', function (Blueprint $table) {
        $table->id();
        $table->timestamps();
    });
}

```

РАЗБЕРЕМ МЕТОД АП

Здесь schema - класс фасад (удобная оболочка) далее обращаемся к методу класса create и аргументы функции (название таблицы и функция кол-бек)

далее создаём айди (ключ в базе для отличия элементов) и две колонки (время занесения объекта в базу и время изменения элемента)

```
public function up()
{
    Schema::create('posts', function (Blueprint $table) {
        $table->id();
        $table->string('title');
        $table->timestamps();
    });
}
```

В миграциях мы можем создавать свои колонки (нужно указать тип)

```
public function up()
{
    Schema::create('posts', function (Blueprint $table) {
        $table->id();
        $table->string('title');
        $table->text('content');
        $table->unsignedBigInteger('likes')->nullable();
        $table->boolean('is_published')->default(1);
        $table->string('images')->nullable();
        $table->timestamps();
    });
}
```

ТУТ МЫ СОЗДАЛИ ТАБЛИЦУ

```
PDO::__construct()
PS C:\OpenServer\domains\laravel\first-app> php artisan migrate
```

Мигрируем таблицу (будет ошибка подключения к бд). Для этого в файле env (окружение (сторонние используемые программы)) подключаемся к бд

The screenshot shows the MySQL Workbench interface. On the left, there's a tree view of databases: 'information_schema', 'laravel' (which contains 'Новая', 'failed_jobs', 'migrations', 'password_resets', 'personal_access_tokens', 'posts', 'users', 'launcher', 'launcher2', 'mysql', and 'migrations'). On the right, there's a main panel with a table titled 'migrations'. The table has columns: id, migration, and batch. There are five rows in the table:

	id	migration	batch
<input type="checkbox"/>	1	2014_10_12_000000_create_users_table	1
<input type="checkbox"/>	2	2014_10_12_100000_create_password_resets_table	1
<input type="checkbox"/>	3	2019_08_19_000000_create_failed_jobs_table	1
<input type="checkbox"/>	4	2019_12_14_000001_create_personal_access_tokens_ta...	1
<input type="checkbox"/>	5	2022_07_25_112250_create_posts_table	2

Получаем вот такую таблицу и бд. Тут есть таблица миграции, которая отслеживает все созданные таблицы для предотвращения повторной миграции

```
D:\Lessons\first_project>php artisan migrate:rollback
```

Откат таблиц (поэтапно (нужно много раз откатывать все срачу он не откатит))

3.2.6 Model

```
PS C:\OpenServer\domains\laravel\first-app> php artisan make:model Post -m
```

Создание модели сразу с миграцией (имя с большой буквы и при связывании в конце Model не писать и обязательно в ед числе)

С данными можно делать действия CRUD

```
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Post extends Model
{
    use HasFactory;
}
```

Структура модели

Наш класс Пост наследник класса модель (у него есть все св-ва и метода класса модель)

В классе модель описаны все с-ва и методы которые помогают работать с данными в таблице. Теперь мы можем работать с данными посредством модели (методами и св-ми класса пост который наследуется от класса модель)

Класс пост является МОДЕЛЬЮ, связанной с таблицей пост
ХЕЛПЕРЫ

Хелперы - зарезервированные методы и функции в ларавел которые упрощают нам жизнь

```
class PostsController extends Controller
{
    public function posts(){
        $var = 'string';
        var_dump($var);
    }
}
```

(показывает какой тип данных и кол-ва символов)

string(6) "string" -- Функция php по дампингу (не хелпер)

```
public function posts(){
    $var = 'string';
    // var_dump($var);
    dd($var);
}
```

-- Хелпер (вывод данных и остановка выполнения)

"string" dd - dump die

3.2.7 Модель чтения данных из базы

Существует множество способов чтения данных из таблицы, но мы пока что рассмотрим только 1

Метод FIND

Для чтения через этот метод мы сначала обращаемся к классу и его статическому методу find(id)

```
$post = Post::find(3); // айди элемента в таблице
```

Dd(\$post)

Переменная называется так же, как и класс

```
App\Models\Post {#620 ▾
  #connection: "mysql"
  #table: "posts"
  #primaryKey: "id"
  #keyType: "int"
  +incrementing: true
  #with: []
  #withCount: []
  +preventsLazyLoading: false
  #perPage: 15
  +exists: true
  +wasRecentlyCreated: false
  #escapeWhenCastingToString: false
  #attributes: array:8 [▶]
  #original: array:8 [▶]
  #changes: []
  #casts: []
  #classCastCache: []
  #attributeCastCache: []
  #dates: []
  #dateFormat: null
  #appends: []
  #dispatchesEvents: []
  #observables: []
  #relations: []
  #touches: []
  +timestamps: true
  #hidden: []
  #visible: []
  #fillable: []
  #guarded: array:1 [▶]
}
```

Так дампится ОБЪЕКТ

Метод find создает ОБЪЕКТ МОДЕЛИ на основании класса

```
class PostController extends Controller
{
    public function index() {
        $post = new Post();
        dd($post);
    }
}
```

По сути, это аналог вот такого создания метода, но круче

```
#attributes: array:8 [▼
  "id" => 3
  "title" => "firstpost"
  "content" => "content"
  "likes" => 10
  "is_published" => 1
  "images" => "img"
  "created_at" => null
  "updated_at" => null
]
```

Здесь хранятся наши атрибуты (колонки) таблицы (сохранились в объект в ассоциативном массиве)

```
$post = Post::find(3); // айди элемента в таблице
dd($post->title);
```

Также можно обратиться к СВОЙСТВУ через объект (НАЗВАНИЕ КОЛОНК)

```
"firstpost"
```

РАЗБЕРЕМ СВ-ВА нашего объекта

```
App\Models\Post {#620 ▾
  #connection: "mysql"
  #table: "posts"
  #primaryKey: "id"
  #keyType: "int"
  +incrementing: true
  #with: []
  #withCount: []
  +preventsLazyLoading: false
  #perPage: 15
  +exists: true
  +wasRecentlyCreated: false
  #escapeWhenCastingToString: false
  #attributes: array:8 [▶]
  #original: array:8 [▶]
  #changes: []
  #casts: []
  #classCastCache: []
  #attributeCastCache: []
  #dates: []
  #dateFormat: null
  #appends: []
  #dispatchesEvents: []
  #observables: []
  #relations: []
  #touches: []
  +timestamps: true
  #hidden: []
  #visible: []
  #fillable: []
  #guarded: array:1 [▶]
}
```

Connection - тип соединения

Table - название таблицы

Primarykey - первичный ключ(айди)

Keytype - тип ключа

Incrementing - авто инкрементирование

```
class Post extends Model
{
    use HasFactory;
    protected $table = 'posts';
}
```

Явное указание типа таблицы (ЛУЧШЕ ДОПОЛНИТЕЛЬНО УКАЗАТЬ НАЗВАНИЕ ХОТЬ ОНО УЖЕ И СТОИТ)

Дз сделать миграции таблицы и тд и тп и вытянуть посредством find

ЭТАПЫ РАБОТЫ С БД (создание и чтение)

Создаем контроллер(для обращения к моделям) -> создаем функцию контроллера(пока пустую) -> создаем модель с миграцией(чтобы работать с базой данных) -> В миграцию создаём колонки таблицы -> миграируем(переносим в бд) -> В контроллере в функции создаем переменную с именем таблицы и обращаемся к классу созданной модели через метод модели find -> обращаемся к роутеру для вывода

3.2.8 Строение Model

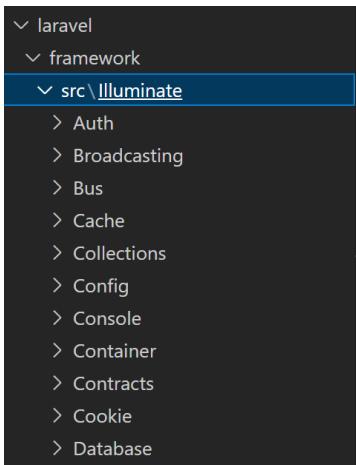
Когда мы создаем модель то мы создаем какой-то класс наследник класса модель

Сама же модель тоже наследует кучу классов

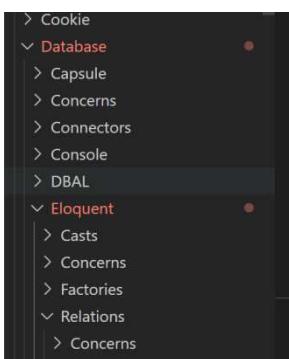
```
use ArrayAccess;
use Illuminate\Contracts\Broadcasting\HasBroadcastChannel;
use Illuminate\Contracts\Queue\QueueableCollection;
use Illuminate\Contracts\Queue\QueueableEntity;
use Illuminate\Contracts\Routing\UrlRoutable;
use Illuminate\Contracts\Support\Arrayable;
use Illuminate\Contracts\Support\CanBeEscapedWhenCastToString;
use Illuminate\Contracts\Support\Jsonable;
use Illuminate\Database\ConnectionResolverInterface as Resolver;
use Illuminate\Database\Eloquent\Collection as EloquentCollection;
use Illuminate\Database\Eloquent\Relations\BelongsToMany;
use Illuminate\Database\Eloquent\Relations\Concerns\AsPivot;
use Illuminate\Database\Eloquent\Relations\HasManyThrough;
use Illuminate\Database\Eloquent\Relations\Pivot;
use Illuminate\Support\Arr;
use Illuminate\Support\Collection as BaseCollection;
use Illuminate\Support\Str;
use Illuminate\Support\Traits\ForwardsCalls;
use JsonSerializable;
use LogicException;
```

Синие - самые интересные они работают с бд

Все эти классы хранятся в папке vendor (там лежит содержимое всех зависимостей и отдельных модулей)



Отсюда мы уже и подтягиваем методы типа find



3.2.8 ELOQUENT

Это тоже самое что и database, но он содержит красивые sql запросы (более простые и естественные, и удобные в использовании) (они приближены к php)

Также он содержит модели, которые уже имеют привязку к таблице (в отличии от database)

Вместо

```
SELECT * FROM first_project.posts WHERE id = 1;
```

Будет

```
$post = Post::find(1);
```

3.2.9 DATABASE

Это уже более приближенное к sql (более низкоуровневый)

```
$user = DB::table('users')->find(3);
```

3.2.10 Методы чтения данных (retrieve)

Все действия с бд можно прочитать в документации в разделе eloquent - getting start

```
class PostsController extends Controller
{
    public function posts () {
        $table = 'posts';
        $posts = Post::all();
        dd($posts);
    }
}

class Post extends Model
{
    use HasFactory;
    protected $table = 'posts';
}
```

(имя указывать как тут, а не как выше)

Вытаскиваем не 1 объект, а все

```
Illuminate\Database\Eloquent\Collection {#980 ▼
  #items: array:2 [▼
    0 => App\Mode...\Post {#1227 ▶}
    1 => App\Mode...\Post {#1228 ▶}
  ]
  #escapeWhenCastingToString: false
}
```

! Collection - аналог массивов НО В КОНТЕКСТЕ ЛАРАВЕЛ (альтернатива массивам) Объекты тут устроены определенном образом с которыми можно работать множеством методов

```
contains
diff
except
find
fresh
intersect
load
loadMissing
modelKeys
makeVisible
makeHidden
only
toQuery
unique
```

(это не все)

Некоторые компании сразу говорят, что будет идти работа с коллекциями и методами коллекций

Если у массивов PHP есть методы, то аналог таких же методов есть и у коллекций

```
$table->string('name');  
$table->unsignedInteger('old')->nullable();
```

ЭТО коллекции (можно вызывать по цепочке через стрелочку)

ИСПОЛЬЗОВАНИЕ КОЛЛЕКЦИЙ И ХОРОШО И ПЛОХО каждая компания выставляет свои требования

Выборка элементов

```
class PostsController extends Controller  
{  
    public function posts () {  
        $table = 'posts';  
        $posts = Post::all();  
        foreach($posts as $post){  
            dump($post -> title);  
        }  
        // dd($posts);  
    }  
}
```

Пройдемся по заголовкам наших атрибутов

```
"firstpost"
```

```
"ивавив"
```

```
class PostsController extends Controller  
{  
    public function posts () [  
        $table = 'posts';  
        $posts = Post::where('status', 1)->get();  
        foreach($posts as $post){  
            dump($post -> title);  
        }  
        // dd($posts);  
    }  
}
```

Вернет все атрибуты

МЕТОД ГЕТ или ПОСТ ВОЗВРАЩАЕТ КОЛЛЕКЦИЮ

SELECT `title` FROM `posts` WHERE 1 - SQL

```
public function posts () {
    $table = 'posts';
    $post = Post::where('status', 1)->first();
    dump($post->title);
    // dd($posts);
}
```

Выбор только первой встречи

```
class FilmsController extends Controller
{
    public function first_films(){
        $table = 'films';
        $films = Film::where('status', 1)->first();
        foreach($films as $film){
            dump($film -> title);
        }
    }

    public function all_films(){
        $table = 'films';
        $films = Film::where('status', 1)->get();
        foreach($films as $film){
            dump($film->title);
        }
        // $table = 'films';
        // $films = Film::all();
        // foreach($films as $film){
        //     dump($film->title);
        // }
    }
}
```

```

class FriendsController extends Controller
{
    public function all_friends () {
        $table = 'friends';
        $friends = Friend::all();
        foreach($friends as $friend){
            dump($friend -> name);
        } //Вывод всех друзей
    }

    public function yung_friends () {
        $table = 'friends';
        $friends = Friend::where('old', '<', 30)->get();
        foreach($friends as $friend){
            dump($friend -> name);
        } // вывод друзей которым меньше 30
    }

    public function first_friend(){
        $table = 'friends';
        $friend = Friend::where('old', 18)->first();
        dump($friend->name); // вывод первого друга которому 18
    }
}

```

3.2.11 Создание объекта в бд (добавление)

Создание происходит посредством обращения к модели и ее методу

```

class Post extends Model
{
    use HasFactory;
    protected $table = 'posts';
    protected $fillable = ['title','discription','likes','status']; // разрешение внесения добавлений
    // protected $fillable = ['title']; // разрешает изменения только названия(того что укажем)

    protected $guarded = []; // разрешение внесения добавлений (разрешает полное редактирование)
}

```

Для начала в модели нужно дать разрешение на внесение изменений в таблицу

```
Route::get('/posts/create', 'PostsController@create');
```

Далее создаем роутер (страницу для создания объектов таблицы)

```

class PostsController extends Controller
{
    public function create(){
        $postsArr = [
            [
                'title' => 'Пост про работу',
                'description' => 'работа супер',
                'likes' => '20',
                'status' => '1'
            ],
            [
                'title' => 'Пост про письки',
                'description' => 'люблю письки', // создаем объект таблицы
                'likes' => '0',
                'status' => '0'
            ]
        ];
        foreach($postsArr as $item){
            dump($item);
            Post::create($item);
        } // занесение в таблицу колонок из массива (циклом по ключу)
    }
}

```

Создаем action (метод) create у контроллера

Далее можно создать двумерный ассоциативный массив, где ключи - атрибуты таблицы бд и значения - данные атрибута

И далее циклом foreach пройтись по нашему массиву и при помощи метода create добавить в таблицу наши объекты

```

Post::create([ // указываем по какому атрибуту вставлять и модель будет понимать в какую колонку какое значение класть
    // можно написать ручками а можно вставить готовый массив
    // сначала нужно убрать защиту добавления в бд
    // protected $guarded = []; // разрешение внесения добавлений (в модель Post)
    // //разрешает полное редактирование всех колонок-атрибутов
    // protected $fillable = ['title']; // разрешает изменения только названия(того что укажем)
];
dd('created');

```

Можно писать атрибуты сразу в метод create (точно так же, как и в массив) и так тоже будет добавлять

(если еще раз попытаться создать тоже самое, то все создастся)

3.2.12 Методы обновления данных(update)

```
Route::get('/posts/update', 'PostsController@update');
```

Создадим роут для обновления

```
public function update(){
    $post = Post::find(2);

    $post->update([
        'title' => 'post 2',
    ]);
    dd($post); // dd - прерывает программу а dump продолжает
}
```

Далее пропишем метод класса, где переменная пост берет в себя все данные с таблицы, где айди = 2 и далее мы вызываем у переменной (уже объекта таблицы) пост метод update

```
#changes: array:2 [▼
    "title" => "post 2"
    "updated_at" => "2022-07-27 09:35:14"
]
```

Получаем наши изменения

(если мы еще раз попытаемся обновить, то обновление не встанет)

```
public function update(){
    $post = Post::find(1);
    dump($post);
    $post -> update([
        'title' => 'post 1',
    ]);
}
```

3.2.13 Метод удаления данных(delete) и soft delete

```
Route::get('/posts/delete', 'PostsController@delete');

public function delete(){
    $post = Post::find(1);
    dump($post);
    $post -> delete();
    dd('deleted');
}
```

Вот и все удаление

Ошибка при повторном удалении

Error

Call to a member function delete() on null

<http://127.0.0.1:8000/posts/delete>

!!! ТАК УДАЛЯТЬ НЕ КРУТО И НЕ НАДО !!!

Такое удаление считается неправильным и неграмотным

Удалять нужно С ВОЗМОЖНОСТЬЮ ВОССТАНОВЛЕНИЯ и информацией
кто что удалил

3.2.14 SOFT DELETE

Мягкое удаление - удаление произошло, но по факту запись осталась (но она
нигде и никогда не вылезет)

```
/*
public function up()
{
    Schema::create('pets', function (Blueprint $table) {
        $table->id();
        $table->string('name');
        $table->unsignedinteger('old');
        $table->text('content')->nullable();
        $table->boolean('status')->default(1)->nullable();
        $table->timestamps();

        $table->softDeletes();
    });
}
```

Для применения этого способа нужно в миграции нашей таблицы прописать
атрибут

```
class Pet extends Model
{
    use HasFactory;
    use SoftDeletes;
    protected $table = 'pets';
    protected $guarded = [];
}
```

Далее в привязанной модели прописываем ТРЕЙД (use -> trade)

Трейд - готовая реализация метода (готовый импортируемый код)

```
PS C:\OpenServer\domains\laravel\first-app> php artisan migrate:fresh
```

Далее обновляем миграцию (этим способом ларавел удалит все таблицы и заново их загрузит) (ПОТЕРЯЕМ ВСЕ ДАННЫЕ)

Параметры								
	id	name	old	content	status	created_at	updated_at	deleted_at
<input type="checkbox"/>	1	barsik	11	cat	0	2022-07-28 00:01:25	2022-07-28 00:01:25	NULL

После этого в нашей таблице появилась новая колонка deleted_at

Если в этой колонке есть какая-то data, то ларавел воспринимает ее как удаленная

ТЕПЕРЬ МОЖНО УДАЛЯТЬ ЗАПИСИ ОБЫЧНЫМ СПОСОБОМ

ВОТ ТАК УДАЛЯТЬ КРУТО И ТАК УДАЛЯТЬ НУЖНО ВСЕГДА

```
public function restore(){
    $pet = Pet::withTrashed()->find(2);
    dump($pet);
    $pet -> restore();
    dd('restored');
}
```

Метод для восстановления данных

3.2.15 Комбинированные методы создания и обновления данных

Это гарантия реализации на уровне CRUD

```
// firstOrCreate
```

Бывают ситуации, когда нужно взять что-то из базы и если такого элемента нет, то создать его (проверка на дубликаты) (добавляем в базу только уникальные элементы по определенным критериям)

```

public function firstOrCreate(){
    $pet = Pet::find(1);

    // $anotherPet = [
    //     'name' => 'rusya',
    //     'old' => 7,
    //     'status' => 0,
    //     'content' => 'dog'
    // ];

    $pet = Pet::firstOrCreate([
        'name' => 'rusya2'
    ], [
        'name' => 'rusya2',
        'old' => 7,
        'status' => 0,
        'content' => 'animal'
    ]);
    dump($pet -> content);
    dd('finished');
}

```

Метод firstOrCreate принимает два массива: первый указывает атрибут, который будет искаться в таблице, и если он его находит, то выполняется first, если не находит, то create (ЧТОБЫ ЭЛЕМЕНТ СОЗДАЛСЯ НЕ ДОЛЖНО СУЩЕСТВОВАТЬ ЗАДАННЫХ ЗНАЧЕНИЙ)

```
// updateOrCreate
```

Обновление записей с проверкой на дубликаты (проверка записей на дубликаты если данные изменены, но ОПРЕДЕЛЕННЫЕ атрибуты совпадают то его обязательно обновить) (одинаковые заголовки у постов)

```

public function updateOrCreate(){
    $pet = Pet::find(4);

    $pet = Pet::updateOrCreate([
        'name' => 'rusya2'
    ], [
        'name' => 'rusyaCREATE',
        'old' => 7,
        'status' => 0,
        'content' => 'not'
    ]);
}

```

Если он нашел объект с именем `rusya2`, то он его ОБНОВЛЯЕТ, а если не нашел, то СОЗДАЕТ

3.3 Миграции. Редактирование миграций

Миграции по сути - таблицы.

Создание миграции - создание таблицы и тд

Про нейминг и создание читать в блоке модели

```
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

class CreateLangsTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('langs', function (Blueprint $table) {
            $table->id();
            $table->string('title');
            $table->text('content')->nullable();
            $table->unsignedInteger('old')->nullable()->default(0);
            $table->unsignedInteger('status')->nullable();

            $table->timestamps();
            $table->softDeletes();
        });
    }
}
```

Метод `up` отвечает за внос изменений

```
public function down()
{
    Schema::dropIfExists('langs');
}
```

Down за бекап или откат

Существует 2 вида миграций:

Миграции создания

Миграции изменения (редактировать атрибут, добавить новый, изменить, удалить атрибут и дропнить таблицу (просто удалить через админку нельзя тк миграция существует, и она при деплое внесется))

3.3.1 Создание колонок

При создании миграции редактирования создавать при этом связанную модель не надо

Именовать такие таблицы принято add_column_..._to_posts_table
delte_column_... _to_friends_table update_column_...

```
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

class AddColumnsToPostsTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::table('posts', function (Blueprint $table) {
            //
        });
    }

    /**
     * Reverse the migrations.
     *
     * @return void
     */
    public function down()
    {
        Schema::table('posts', function (Blueprint $table) {
            //
        });
    }
}
```

При таком нейминге сразу laravel задает обращение к указанной таблице

НУЖНО ПРОПИСЫВАТЬ КАК НАКАТЫВАНИЕ, ТАК И ОТКАТЫВАНИЕ
ОБЯЗАТЕЛЬНО
ТАКЖЕ ДЕЛАТЬ ПРОВЕРКУ (НАКАТ-ОТКАТ-НАКАТ)

```
class AddColumnsToPostsTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::table('posts', function (Blueprint $table) {
            $table->text('content')->nullable(); // обязательно нулабл
        });
    }
}
```

Прописываем добавление случ образом С ОБЯЗАТЕЛЬНЫМ НУЛАБЛ
(потому что если таблица уже существует и в старых версиях нет этой
колонки, то будет жопа)

```

    }
    public function down()
    {
        Schema::table('posts', function (Blueprint $table) {
            $table->dropColumn(['content']);
        });
    }
}
```

Далее обязательно прописываем как мы будем откатывать

```
D:\Lessons\first_project>php artisan migrate      накатывание
D:\Lessons\first_project>php artisan migrate:rollback      откатывание
```

```
public function up()
{
    Schema::table('posts', function (Blueprint $table) {
        $table->text('content')->nullable()->after('title'); // обязательно нулабл
    });
}
```

Так можно накатить ‘после’

3.3.2 Удаление колонок

```
> php artisan make:migration delete_column_discription_to_posts_table
```

```
    */
    public function up()
    {
        Schema::table('posts', function (Blueprint $table) {
            $table->dropColumn('discription');
        });
    }
}
```

Удаление

```
public function down()
{
    Schema::table('posts', function (Blueprint $table) {
        $table->text('discription')->nullable()->after('content');
    });
}
```

Откат (было удаление стало создание (тоже с нулабл))

```
php artisan migrate  
discription_to_posts
```

3.3.3 Редактирование колонок

ПЕРЕИМЕНОВАНИЕ КОЛОНКИ

```
discription_to_posts_table (54.5ms)
php artisan make:migration edit_column_content_to_posts_table
```

```
    /*
    public function up()
    {
        Schema::table('posts', function (Blueprint $table) {
            $table->renameColumn('content', 'postscontent');
        });
    }
}
```

```
    */
    public function down()
    {
        Schema::table('posts', function (Blueprint $table) {
            $table->renameColumn('postscontent', 'content');
        });
    }
}
```

```
PS C:\OpenServer\domains\laravel\first-app> php artisan migrate
Migrating: 2022_07_29_122310_edit_column_content_to_posts_table

Error

Class 'Doctrine\DBAL\Driver\AbstractMySQLDriver' not found

at C:\OpenServer\domains\laravel\first-app\vendor\laravel\framework\src\Illuminate\Database\PDO\MySQLDriver.php:8
    4
    5     use Doctrine\DBAL\Driver\AbstractMySQLDriver;
    6     use Illuminate\Database\PDO\Concerns\ConnectsToDatabase;
    7
→  8 class MySQLDriver extends AbstractMySQLDriver
    9 {
    10     use ConnectsToDatabase;
    11 }
    12

1  C:\OpenServer\domains\laravel\first-app\vendor\composer\ClassLoader.php:571
   include()

2  C:\OpenServer\domains\laravel\first-app\vendor\composer\ClassLoader.php:428
   Composer\Autoload\includeFile()
PS C:\OpenServer\domains\laravel\first-app>
```

Активация Windows

Чтобы активировать Windows, перейдите

УВИДИМ ТАКУЮ ОШИБКУ

Для решения этой проблемы нужно установить расширение DBAL в композере

```
composer require doctrine/dbal
```

3.3.4 ИЗМЕНЕНИЕ ТИПА ДАННЫХ

```
php artisan make:migration change_column_postcontent_string_to_posts_table
```

```
public function up()
{
    Schema::table('posts', function (Blueprint $table) {
        $table->string('posts_content')->change();
    });
}
```

```
public function down()
{
    Schema::table('posts', function (Blueprint $table) {
        $table->text(['posts_content'])->change();
    });
}
```

3.3.5 УДАЛЕНИЕ ТАБЛИЦЫ

```
public function up()
{
    Schema::dropIfExists('posts');
}
```

```
public function down()
{
    Schema::create('posts', function (Blueprint $table) {
        $table->id();
        $table->string('title');
        $table->string('postscontent');
        $table->unsignedInteger('likes')->nullable();
        $table->boolean('status')->default(1);
        $table->timestamps();
    });
}
```

```
D:\lessons\first_project>php artisan migrate:Fresh
Dropped all tables successfully.
Migration table created successfully.
```

Перезагрузка всех таблиц

3.4 View

3.4.1 Знакомство

VIEW - шаблон куда мы заморачиваем данные для сайта



.blade.php - синтаксис blade под php

Blade упрощает написание php в html тк по сути это html теги с удобным синтаксисом php

```

@if (Route::has('login'))
    <div class="hidden fixed top-0 right-0 px-6 py-4 sm:flex sm:justify-end sm:items-center sm:gap-2" style="background-color: #fff; border-bottom: 1px solid #eaeaf2; backdrop-filter: blur(10px); z-index: 100;">
        @auth
            <a href="{{ url('/home') }}" class="text-sm text-gray-700 font-medium rounded-md py-1 px-3 shadow-sm transition duration-150 ease-in-out hover:bg-gray-100 focus:outline-none focus-visible:ring-2 focus-visible:ring-blue-500">Home
        @else
            <a href="{{ route('login') }}" class="text-sm text-gray-700 font-medium rounded-md py-1 px-3 shadow-sm transition duration-150 ease-in-out hover:bg-gray-100 focus:outline-none focus-visible:ring-2 focus-visible:ring-blue-500">Login
            @if (Route::has('register'))
                <a href="{{ route('register') }}" class="text-sm text-gray-700 font-medium rounded-md py-1 px-3 shadow-sm transition duration-150 ease-in-out hover:bg-gray-100 focus:outline-none focus-visible:ring-2 focus-visible:ring-blue-500">Register
            @endif
        @endauth
    </div>
@endif

```

Пример .blade.php

Наш роут обладает функцией callback (ответ на запрос)

```

public function index(){
    $posts = Post::all();
    return view('posts', compact('posts'));
}

```

Так мы возвращаем view (в аргументах только название файла без пути и без .blade.php)

```

<div>
    <?php foreach($posts as $post)>
</div>

```

Теперь вместо такого мы используем blade который все упрощает

```

@forelse($posts as $post)
    <div>{{ $post -> title }}</div>
@empty
    <div>No posts found</div>
@endforelse

```

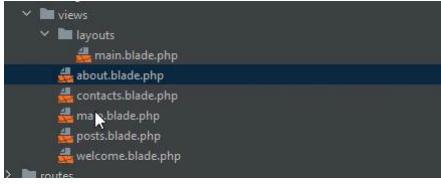
Теперь так

<!-- --> - комментарии

3.4.2 Шаблоны view

При создании страниц сайта всегда что-то будет повторяться(например, структура) и чтобы не писать ее каждый раз во view есть ШАБЛОНИЗАЦИЯ

Мы переносим повторяющийся текст в отдельный файл в папку view-layouts и после можем к ней обращаться



```
<html lang="en">
<head>
|   <meta charset="UTF-8">
|   <meta http-equiv="X-UA-Compatible" content="IE=edge">
|   <meta name="viewport" content="width=device-width, initial-scale=1.0">
|   <title>Doc</title>
</head>
<body>
|   @yield('doc')
</body>
</html>
```

В файл шаблона переносим повторяющийся код

```
@extends('layouts.main')
@section('doc')
|   @foreach($posts as $post)
|       <div>{{$post -> title}}</div>
|   @endforeach
@endsection()
```

В файл, где нужен этот код вставляем такую структуру

```
class PostsController extends Controller
{
    public function index(){
        $posts = Post::all();
        return view('posts', compact('posts'));
    }
}
```

Обязательно прописать для каждой страницы свой контроллер

```
Route::get('/posts/main', 'PostsController@main')->name('post.main');
Route::get('/posts/server1', 'PostsController@server1')->name('post.server1');
Route::get('/posts/server2', 'PostsController@server2')->name('post.server2');
Route::get('/posts/create', 'PostsController@create')->name('post.create');
```

Роутеру можно задать имя

```
@extends('layouts.main')
@section('doc')
    @foreach($server as $item)
        <h1>{{$item->title}}</h1>
        <p>{{$item->content}}</p>
    @endforeach
@endsection()
```

```
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <div>
        <nav>
            <ul>
                <li><a href="{{route('post.main')}}">Main</a></li>
                <li><a href="{{route('post.server1')}}">Server 1</a></li>
                <li><a href="{{route('post.server2')}}">Server 2</a></li>
                <li><a href="{{route('post.create')}}">Create</a></li>
                <li><a href=""></a>Donate</li>
            </ul>
        </nav>
    </div>
    @yield('doc')
</body>
</html>
```

```

class PostsController extends Controller
{
    public function main(){
        return view('main');
    }

    public function create(){
        $postsArr = [
            'title' => 'magic',
            'content' => 'magic content',
            'likes' => 100,
            'status' => 1,
        ], [
            'title' => 'tech',
            'content' => 'tech content',
            'likes' => 10,
            'status' => 0,
        ];
    }

    foreach($postsArr as $post){
        // dump($post);
        $mypost[] = Post::create($post);
    }

    return view('postsCreate', compact('mypost'));
}

```

```

public function server1(){
    $server = Post::where('title','=','magic')->get();
    // dump($server);
    return view('server1', compact('server'));
}

public function server2(){
    $server = Post::where('title','=','tech')->get();
    return view('server2', compact('server'));
}

```

3.5 Bootstrap в laravel

```
composer require laravel/ui
```

Для начала установим ui

```
ui                                     Scaffold basic login and registration views and routes
ui:auth                                Scaffold the authentication controllers
```

```
PS C:\OpenServer\domains\laravel\two\two> php artisan ui bootstrap
Bootstrap scaffolding installed successfully.
```

```
Please run "npm install && npm run dev" to compile your fresh scaffolding.
```

Далее устанавливаем bootstrap

```
npm install && npm run dev
```

Прописываем команду, которая подключит дополнительные css стили
установит bootstrap
(структура laravel css не всегда .css может быть и .scss)

Суть npm run dev - компиляция из формата scss и др. в css и собирает в папку public

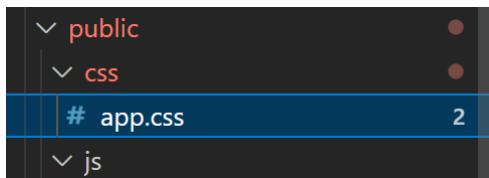
```
composer require laravel/ui
php artisan ui bootstrap
npm install
npm run dev
npx mix
```

ВЫПОЛНЯЕМ ВОТ ЭТИ КОМАНДЫ И ПОЛУЧАЕМ УСПЕХ

```
✓ Compiled Successfully in 7840ms
File      Size
/js/app.js 2.22 MiB
css/app.css 232 KiB
webpack compiled successfully
```

```
two > JS webpack.mix.js > ...
1 const mix = require('laravel-mix');
2
3 /*
4 | -----
5 | Mix Asset Management
6 | -----
7 |
8 | Mix provides a clean, fluent API for defining some Webpack build steps
9 | for your Laravel application. By default, we are compiling the Sass
10| file for the application as well as bundling up all the JS files.
11|
12 */
13
14 mix.js('resources/js/app.js', 'public/js')
15     .sass('resources/sass/app.scss', 'public/css')
16     .sourceMaps();
17
```

Тут можно все переделать



Все стили из bootstrap лежат в этой директории

Папка public нужна для обращения к ресурсам через публичный доступ(вопрос безопасности)

<link rel="stylesheet" href="{{ asset('css/app.css') }}> (грамотная сборка ссылки) ({{ PUBLIC_PATH }}) (asset - все доп. файлы проекта этот хелпер всегда попадает в нашу папку)
Подключение bootstrap к файлу

3.6 CRUD через интерфейс

Реализовывать будет связку frontend и backend

Всего для реализации crud в ларавел есть 7 шаблонов(actions)(рестфул контроллеры)

Actions Handled By Resource Controller

Verb	URI	Action	Route Name
GET	/photos	index	photos.index
GET	/photos/create	create	photos.create
POST	/photos	store	photos.store
GET	/photos/{photo}	show	photos.show
GET	/photos/{photo}/edit	edit	photos.edit
PUT/PATCH	/photos/{photo}	update	photos.update
DELETE	/photos/{photo}	destroy	photos.destroy

Хендлеры для crud через интерфейс(конвенция о названии роутеров и их запросов)

3.6.1 INDEX

Перечень всех элементов (посты фотки и т.д. и т.п.) (главная страница)

```
Route::get('/servers', 'ServerController@index')->name('server.index'); //страница всех серверов
```

Роут для индекса

```
public function index(){
    $servers = Server::all();
    return view('server.index', compact('servers')); //вывод всех серверов
}
```

АКШОН для индекса

```
@extends('layouts.main')
@section('content')
<div>
    @foreach($servers as $server)
        <div>{{$server->id}}. {{$server->title}}</div>
    @endforeach
</div>
@endsection
```

Вью для индекса

3.6.2 CREATE

отвечает за создание объектов бд

```
Route::get('/servers/create', 'ServerController@create')->name('server.create'); //страница создания сервера
```

```
public function create(){
    return view('server.create');
```

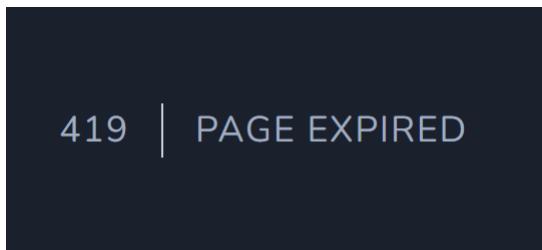
```
@extends('layouts.main')
@section('content')


<form action="{{route('server.store')}}" method="POST">
        @csrf
        <!-- СТАВИТ ЗАЩИТУ -->
        <div class="mb-3">
            <label for="title" class="form-label">Title</label>
            <input type="text" name="title" class="form-control" id="title" placeholder="Enter title">
        </div>
        <div class="mb-3">
            <label for="content" class="form-label">Content</label>
            <!-- <input type="password" class="form-control" id="exampleInputPassword1"> -->
            <textarea class="form-control" name="content" id="content" cols="5" rows="5" placeholder="Enter content..."></textarea>
        </div>
        <div class="mb-3">
            <label for="image" class="form-label">Image</label>
            <input type="text" name="image" class="form-control" id="image" placeholder="Image..."></input>
        </div>
        <button type="submit" class="btn btn-primary">Create</button>
    </form>


@endsection()
```

3.6.3 STORE

обработчик форм методом post (для этого нужно добавить защиту @csrf)



Ошибка сайта без добавления защиты

Межсайтовая подделка запроса (Cross-site request forgery)



CSRF — вид атак на посетителей веб-сайтов, использующий недостатки протокола HTTP. Если жертва заходит на сайт, созданный злоумышленником, от её лица тайно отправляется запрос на другой сервер, осуществляющий некую вредоносную операцию. Википедия

Отзыв

НУЖНО УЖЕ УМЕТЬ СОЗДАВАТЬ И ОБРАБАТЫВАТЬ ФОРМЫ(нейминг элементов такой же как название атрибутов в таблице бд)

```
Route::post('/servers', 'ServerController@store')->name('server.store'); //обработчик формы
```

Роут обработчик

```
public function store(){
    $data = request()->validate([
        'title' => 'string',
        'content' => 'string',
        'image' => 'string'
    ]);
    // dump($data);
    Server::create($data);
    return redirect()->route('server.index');
}
```

Файл обработчик формы (ключи в массиве должны совпадать с именем бд в validate мы задаем ЗАЩИТУ говоря, что мы ожидаем и какой тип и только это) (если не указать ожидаемый тип может быть ошибка пустого поля)

```

<div>
    <form action="{{route('server.store')}}" method="POST">
        @csrf
        <!-- СТАВИТ ЗАЩИТУ -->
        <div class="mb-3">
            <label for="title" class="form-label">Title</label>
            <input type="text" name="title" class="form-control" id="title" placeholder="Enter title">
        </div>
        <div class="mb-3">
            <label for="content" class="form-label">Content</label>
            <!-- <input type="password" class="form-control" id="exampleInputPassword1"> -->
            <textarea class="form-control" name="content" id="content" cols="5" rows="5" placeholder="Enter content"></textarea>
        </div>
        <div class="mb-3">
            <label for="image" class="form-label">Image</label>
            <input type="text" name="image" class="form-control" id="image" placeholder="Image URL">
        </div>
        <button type="submit" class="btn btn-primary">Create</button>
    </form>
</div>
@endsection()

```

Форма для обработки(ОБЯЗАТЕЛЬНО С ЗАЩИТОЙ @csrf)(НЕЙМИНГ КАК В БД)

3.6.4 SHOW

Выщепить что-то конкретное (конкретный пост или конкретный сервер по айди или названию) (по методу гет)

`Route::post('/servers/{server}', 'ServerController@show')->name(['server.show']); //обработчик формы`
Роут show (вместо {server} может быть все что угодно это просто условное обозначение)

```

public function show($id){
    $id = Server::findOrFail($id);
    dd($id->title);
}

```

(метод 1)

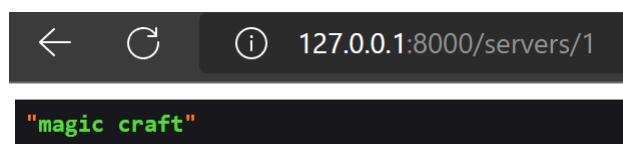
Экшон show FindOrFail - не выдает ошибку при ненаходе а перенаправляет на страницу 404

```

public function show(Server $id){
    dd($id->title);
}

```

(метод 2)



Получаем в ответ

```
crud > resources > views > server > show.blade.php > div > div > h1
1 @extends('layouts.main')
2 @section('content')
3 <div>
4     <div><h1>{{$id->title}}</h1></div>
5     <div>{{$id->content}}</div>
6 </div>
7 @endsection()
```

```
@extends('layouts.main')
@section('content')
<div>
    @foreach($servers as $server)
        <div><a href="{{route('server.show', [$server->id])}}>{{$server->id}}. {{$server->title}}</a></div>
    @endforeach
</div>
@endsection
```

Так мы осуществляем переход по ссылке с определенному серверу

3.6.5 EDIT / UPDATE

обработчик формы для обновления конкретного поста
Update - patch обновление поста

```
Route::get('/servers/{id}/edit', 'ServerController@edit')->name('server.edit'); //обрабтчик формы
Route::patch('/servers/{id}', 'ServerController@update')->name(['server.update']); //обрабтчик формы
```

```
public function edit(Server $id){
    // dd($id->title);
    return view('server.edit', compact('id'));
}

public function update(Server $id){
    $data = request()->validate([
        'title' => 'string',
        'content' => 'string',
        'image' => 'string'
    ]);
    $id -> update($data);
    return redirect()->route(['server.show', $id->id]);
}
```

```

@extends('layouts.main')
@section('content')


<form action="{{route('server.update', $id)}}" method="POST">
        @csrf
        @method('patch')
        <!-- СТАВИТ ЗАЩИТУ -->
        <div class="mb-3">
            <label for="title" class="form-label">Title</label>
            <input type="text" name="title" class="form-control" id="title" placeholder="Enter title" value=".
        </div>
        <div class="mb-3">
            <label for="content" class="form-label">Content</label>
            <!-- <input type="password" class="form-control" id="exampleInputPassword1"> -->
            <textarea class="form-control" name="content" id="content" cols="5" rows="5" placeholder="Enter co:
        </div>
        <div class="mb-3">
            <label for="image" class="form-label">Image</label>
            <input type="text" name="image" class="form-control" id="title" placeholder="Image...">
        </div>
        <button type="submit" class="btn btn-primary">Create</button>
    </form>


@endsection()

```

3.6.6 DELETE

удаление поста

```

public function destroy(Post $post){
    $post->delete($post);
    return redirect()->route('post.index');
}

```

```

<form action="{{route('post.destroy', $post)}}" method="POST">
    @csrf
    @method('delete')
    <button type="button" class="btn btn-primary" href="{{route('post.destroy', $post->id)}}>Удалить</bu
</form>
<a href="{{route('post.index')}}">Назад</a>

```

3.7 Отношения один ко многим

Выражают отношения моделей(таблиц) в базе

УЖЕ НУЖНО ЗНАТЬ ПРО ОТНОШЕНИЯ

Один к одному применяется ооооооченб редко

Один ко многим - самое частое потом многое ко многому

—

Возьмем отношения ПОСТЫ - КАТЕГОРИИ (категория котики - много постов)

```
public function up()
{
    Schema::create('categories', function (Blueprint $table) {
        $table->id();
        $table->string('title');
        $table->timestamps();
    });
}
```

Создадим таблицу с категориями

```
Schema::create('posts', function (Blueprint $table) {
    $table->id();
    $table->string('title');
    $table->string('content');
    $table->unsignedInteger('likes')->nullable();
    $table->string('image')->nullable();
    $table->boolean('status')->default(1);
    $table->softDeletes();
    $table->timestamps();

    $table->unsignedBigInteger('category_id')->nullable(); // добавляем колонку категории

    $table->index('category_id', 'post_category_idx'); // добавляем колонку как индекс

    $table->foreign('category_id')->references('id')->on('categories'); // связываем таблицы
});
```

Привяжем таблицу с категориями к основной таблице(последние 3 строки)

The screenshot shows a database configuration interface. A table row for the 'posts' table is being edited. The 'category_id' column is selected, and its properties are displayed in a modal dialog. The 'Type' dropdown is set to 'unsigned BigInteger'. The 'Function' dropdown is set to 'foreign'. The 'Value' dropdown is set to 'category_id' and has a tooltip 'on("categories")'. The 'Null' dropdown is set to 'nullable'. A checkbox 'Игнорировать' (Ignore) is checked. A tooltip for the 'Value' dropdown lists two entries: 'cats - 1' and 'dogs - 2'. Below the table, a message says 'Получаем связь атрибута таблицы posts к id таблице categories' (We get the connection of attribute posts to id of table categories).

Получаем связь атрибута таблицы posts к id таблице categories

```

 Illuminate\Database\Eloquent\Collection {#1026 ▶
 #items: array:2 [▼
   0 => App\Models\Category {#1028 ▶
     #connection: "mysql"
     #table: "categories"
     #primaryKey: "id"
     #keyType: "int"
     #incrementing: true
     #with: []
     #withCount: []
     #perPage: 15
     #exists: true
     #wasRecentlyCreated: false
     #attributes: array:4 [▼
       "id" => 1
       "title" => "cats"
       "created_at" => null
       "updated_at" => null
     ]
     #original: array:4 [▶]
     #changes: []
     #casts: []
     #classCastCache: []
     #dates: []
     #dateFormat: null
     #appends: []
     #dispatchesEvents: []
     #observables: []
     #relations: []
     #touches: []
     #timestamps: true
     #hidden: []
     #visible: []
     #fillable: []
     #guarded: array:1 [▶]
   ]
   1 => App\Models\Category {#1027 ▶}
 ]
}

public function index()
{
    $category = Category::find(1);

    $posts = Post::where('category_id', $category->id)->get();
    dd($posts);

    // return view('post.index', compact('posts'));
}

```

ЭТО МОДЕЛЬ ОКАЗЫВАЕТСЯ

Выцепить все посты по категории

-- СПОСОБ 1

```

class Category extends Model
{
    use HasFactory;
    protected $guarded =[];
    protected $table = 'categories';

    public function posts(){
        return $this -> hasMany(Post::class, 'category_id', 'id');
        //this - обращение к созданному на основе классу объекта
    }
}

```

Отношения

```

class Post extends Model
{
    use HasFactory;
    use SoftDeletes;

    protected $table = 'posts';
    protected $guarded = false;

    public function category()
    {
        return $this->belongsTo(Category::class, 'category_id', 'id');
    }
}

```

Обратное отношение

```

class PostController extends Controller
{
    public function index(){
        // $posts = Post::all();
        // return view('post.index', compact('posts'));

        $categ = Category::find(1);
        // // dump($categ->title);
        // $posts = Post::where('category_id', $categ->id)->get();
        // foreach($posts as $post){
        //     dd($post->title);
        // }

        dd([$categ->posts]);
    }
}

```

Проверка

3.7.1 Модификация crud - категории(один ко многим)

3.7.2 Создание:

```

public function create(){
    $category = Category::all();
    return view('post.create', compact('category'));
}

public function store(){
    $data = request()->validate([
        'title' => 'string',
        'content' => 'string',
        'image' => 'string',
        'category_id' => ''
    ]);
    Post::create($data);
    return redirect()->route('post.index');
}

public function show(Post $post){
    return view('post.show', compact('post'));
}

```

Меняем create store

```

<div class="input-group mb-3">
    <label class="input-group-text" for="category">Категория</label>
    <select class="form-select" id="category" name="category_id">
        @foreach($category as $id)
            <option value="{{ $id->id }}>{{ $id->title }}</option>
        @endforeach
    </select>
</div>

```

Добавляем селект

Результат

```
<div><p><b>Категория:</b>
{{$post->category_id}}</p></div>
```

Добавляем в show

Результат

3.7.3 Редактирование:

```
public function edit(Post $post){
    $category = Category::all();
    return view('post.edit', compact(['post', 'category']));
}
```

Меняем edit

```

public function update(Post $post){
    $data = request() -> validate([
        'title' => 'string',
        'content' => 'string',
        'image' => 'string',
        'status' => 'boolean',
        'category_id' => 'integer'
    ]);

    $post->update($data);
    return redirect() -> route('post.index', $post->id);
}

```

Меняем update

```

<select class="form-select" id="category" required name="category_id">
    @foreach($category as $id)
        <option
            {{ $id->id === $post->category_id ? 'selected' : ''}}
            value='{{ $id->id }}'>{{ $id->title }}</option>
    @endforeach
</select>

```

Добавляем селект с проверкой (update)



Результат

3.8 Отношения многие ко многим

Реализуем как теги - посты

РАБОТАЕТ ЭТО НЕ КАК ОДИН КО МНОГИМ А ЧЕРЕЗ ТАБЛИЦУ ЗЕРКАЛО

```

php artisan make:model Tag -m
Model created successfully.
Created Migration: 2022_08_05_073110_create_tags_table

```

Таблица тегов

```

PS C:\OpenServer\domains\laravel\crudapi> php artisan make:model PostTag -m
Model created successfully.
Created Migration: 2022_08_05_073250_create_post_tags_table

```

Промежуточная таблица (имя из двух связь таблиц по алфавиту)
Можно не создавать модель, но лучше создать

```
2022_08_05_024131_create_posts_table.php  
2022_08_05_073110_create_tags_table.php  
2022_08_05_073250_create_post_tags_table.php
```

Таблицы

```
class CreateTagsTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('tags', function (Blueprint $table) {
            $table->id();
            $table->string('title');
            $table->timestamps();
        });
    }
}
```

Содержание таблицы тег можно удалить, а можно оставить(лучше оставить)

```
public function up()
{
    Schema::create('post_tags', function (Blueprint $table) {
        $table->id();

        $table->unsignedBigInteger('post_id');
        $table->unsignedBigInteger('tag_id');

        $table->index('post_id', 'post_tag_post_idx');
        $table->index('tag_id', 'post_tag_tag_idx');

        $table->foreign('post_id','post_tag_post_fk')->on('posts')->references('id');
        $table->foreign('tag_id','post_tag_tag_fk')->on('tags')->references('id');

        $table->timestamps();
    });
}
```

Содержание таблицы ПостТег

```
> app > models > <-- Post.php > Post > ↓ tags  
<?php  
  
namespace App\Models;  
  
use Illuminate\Database\Eloquent\Factories\HasFactory;  
use Illuminate\Database\Eloquent\Model;  
use Illuminate\Database\Eloquent\SoftDeletes;  
  
class Post extends Model  
{  
    use HasFactory;  
    use SoftDeletes;  
    protected $table = 'posts';  
    protected $guarded = [];  
  
    public function tags(){  
        return $this->belongsToMany(Tag::class, 'post_tags', 'post_id', 'tag_id');  
    }  
}
```

Добавляем связь для Post

```
class Tag extends Model  
{  
    use HasFactory;  
  
    public function posts(){  
        return $this->belongsToMany(Post::class, 'post_tags', 'tag_id', 'post_id');  
    }  
}
```

Добавляем связь для Tag

```
class PostController extends Controller  
{  
    public function index(){  
        // $posts = Post::all();  
        // return view('post.index', compact('posts'));  
  
        $post = Post::find(1);  
        dd($post->tags);  
    }  
}
```

Проверка

По аналогии делается обратное

3.8.1 Модификация crud - Теги(многие ко многим)

СПОСОБ 1 (ПЛОХОЙ) - НАРУШЕНИЕ СОХРАННОСТИ БАЗЫ (НЕТ ТРАНЗАКЦИЙ)

```
class PostTag extends Model
{
    use HasFactory;
    protected $guarded = false;
}
```

Снимаем защиту с зеркала

```
<div class="input-group mb-3">
<select class="form-select" multiple aria-label="multiple select example" name="tags[]>
    <option selected>Выберите теги(ctrl)</option>
    @foreach($tags as $tag)
        <option value="{{ $tag->id}}>{{ $tag->title}}</option>
    @endforeach
</select>
```

Добавляем опцию выбора мультиселект

```
public function store()
{
    $data = request()->validate([
        'title' => 'string',
        'content' => 'string',
        'image' => 'string',
        'category_id' => 'integer',
        'tags' => 'array'
    ]);
    $tags = $data['tags'];
    unset($data['tags']);
    // dd($tags,$data);
    $post = Post::create($data);
    foreach ($tags as $tag) {
        PostTag::firstOrCreate([ // исключаем дублирование
            'tag_id' => $tag,
            'post_id' => $post->id
        ]);
    }
    return redirect()->route('post.index');
}
```

Обрабатываем

СПОСОБ 2 - БОЛЕЕ ПРОФЕССИОНАЛЬНЫЙ, НО ВСЕ РАВНО БЕЗ ТРАНЗАКЦИЙ

3.8.2 Создание

```

<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\SoftDeletes;

class Post extends Model
{
    use HasFactory;
    use SoftDeletes;
    protected $table = 'posts';
    protected $guarded = [];

    public function tags(){
        return $this->belongsToMany(Tag::class, 'post_tags', 'post_id', 'tag_id');
    }
}

```

Связь для постов

```

class Tag extends Model
{
    use HasFactory;

    public function posts(){
        return $this->belongsToMany(Post::class, 'post_tags', 'tag_id', 'post_id');
    }
}

```

Связь для тегов

```

public function create(){
    $category = Category::all();
    $tags = Tag::all();
    return view('post.create', compact('category', 'tags'));
}

```

Добавляем теги

```

<div class="input-group mb-3">
    <label class="input-group-text" for="tags">Теги (ctrl+лкм)</label>
    <select name="tags[]" class="form-select" id="tags" multiple aria-label="multiple select">
        @foreach($tags as $tag)
            <option value="{{ $tag->id }}">{{ $tag->title }}</option>
        @endforeach
    </select>
</div>

```

Показываем окно тегов



```

public function store(){
    $data = request() -> validate([
        'title' => 'string',
        'content' => 'string',
        'image' => 'string',
        'category_id' => 'integer',
        'tags' => ''
    ]);

    $tag = $data['tags'];
    unset($data['tags']);

    $post = Post::create($data);

    $post->tags()->attach($tag);

    return redirect() -> route('post.index');
}

```

Редактируем store

Отображения пока не сделал

3.8.3 Редактирование

```

public function edit(Post $post){
    $category = Category::all();
    $tags = Tag::all();
    return view('post.edit', compact(['post', 'category', 'tags']));
}

```

Добавим в edit

```

public function update(Post $post)
{
    $data = request()->validate([
        'title' => 'string',
        'content' => 'string',
        'image' => 'string',
        'category_id' => 'integer',
        'tags' => ''
    ]);
    $tags = $data['tags'];
    unset($data['tags']);
    // dd($tags, $data);
    $post -> update($data);

    $post->tags()->sync($tags); // tags() - запрос в базу
    return redirect()->route('post.show', $post->id);
}

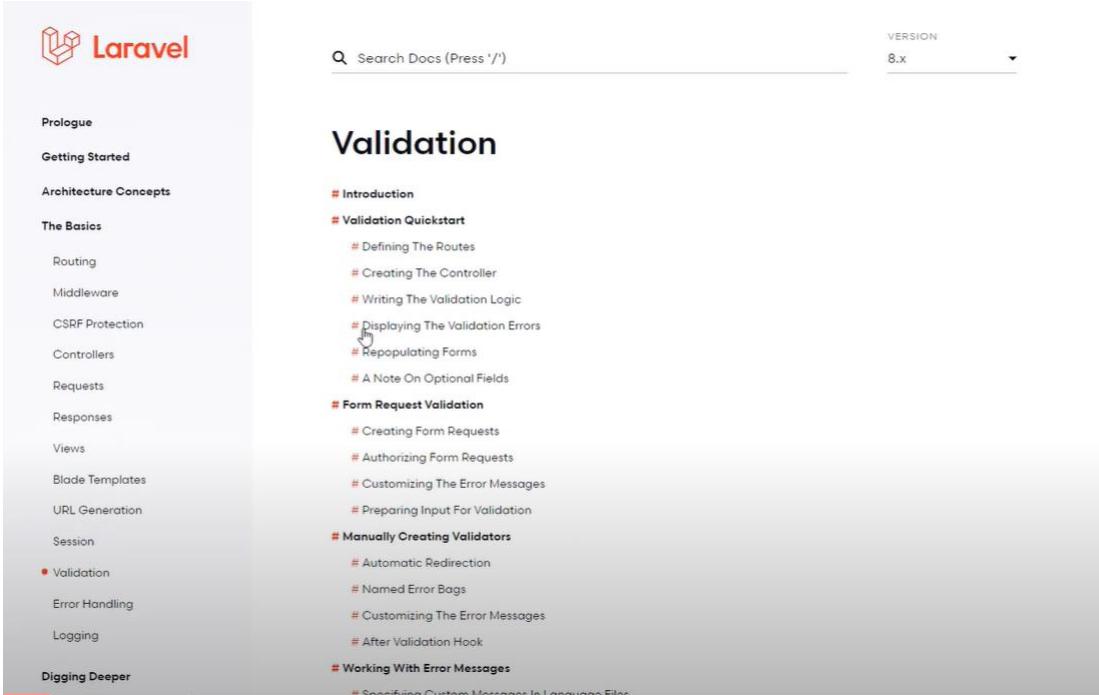
```

Меняем атак на синк чтобы не только добавляли новое, но и убирали старое(удаляет все до момента привязки и добавляет новое)

```
</div>
<div class="input-group mb-3">
    <select class="form-select" multiple aria-label="multiple select example" name="tags[]>
        <option disabled>Выберите теги(ctrl)</option>
        @foreach($tags as $tag)
            <option @foreach($post->tags as $postTag)
                {$tag->id === $postTag->id ? 'selected' : ''}>
                value="{{$tag->id}}>{{$tag->title}}
            </option>
        @endforeach
    </select>
</div>
<button type="submit" class="btn btn-primary">Изменить</button>
```

Добавляем селект

3.9 Модификация crud - обработчик ошибок



The screenshot shows the Laravel documentation website. The top navigation bar includes the Laravel logo, a search bar labeled "Search Docs (Press '/')", and a "VERSION" dropdown set to "8.x". The left sidebar contains a navigation tree with categories like Prologue, Getting Started, Architecture Concepts, The Basics, Routing, Middleware, CSRF Protection, Controllers, Requests, Responses, Views, Blade Templates, URL Generation, Session, Validation (which is currently selected), Error Handling, Logging, and Digging Deeper. The main content area is titled "Validation" and lists several sections: Introduction, Validation Quickstart (with sub-sections like Defining The Routes, Creating The Controller, Writing The Validation Logic, Displaying The Validation Errors, Repopulating Forms, and A Note On Optional Fields), Form Request Validation (with sub-sections like Creating Form Requests, Authorizing Form Requests, Customizing The Error Messages, and Preparing Input For Validation), Manually Creating Validators (with sub-sections like Automatic Redirection, Named Error Bags, Customizing The Error Messages, and After Validation Hook), and Working With Error Messages (with sub-sections like Specifying Custom Messages In Language Files). A cursor is hovering over the "Displaying The Validation Errors" link under the Validation Quickstart section.

ПОЛЬЗУЕМСЯ ДОКУМЕНТАЦИЕЙ

Способ 1 - просто добавить атрибут required в html (не оч.)

Способ 2 - обработчик ошибок(круто если не стирать введенный тест при ошибке)

```
<div class="mb-3">
    <label for="title" class="form-label">Title</label>
    <input name="title" type="text" class="form-control" id="title" placeholder="Название"
    @error('title')
        <p class="text-danger">Ошибка ввода</p>
    @enderror
</div>
```

Добавляем error

```
@error('content')
    <p class="text-danger">{{$message}}</p>
@enderror
```

Зарезервированная переменная выводит ошибку

```
$data = request()->validate([
    'title' => 'string',
    'content' => 'required|string',
    'image' => 'string',
    'category_id' => 'integer',
    'tags' => 'array'
]);
```

Так можно добавить обязательность ввода

The content field is required.

Результат

```
<input value="{{old('title')}}" name="title" type="text" class="form-control" id="title">
```

Чтобы введённый без ошибок текст остался

A screenshot of a web form. The 'Title' field contains 'dsfg'. The 'Content' field has the placeholder 'Содержание:' and a red error message 'The content field is required.' below it.

```
<select class="form-control" id="category" name="category_id">
    @foreach($categories as $category)
        <option
            {{ old('category_id') == $category->id ? ' selected' : '' }}>
            {{ $category->title }}</option>
    @endforeach
</select>
```

Чтобы категория осталась

3.10 Отношения один ко многим и многие ко многим через конвенцию Laravel

Можно кодить так как было описано выше, но это не ЧЕРЕЗ КОНВЕНЦИЮ

Через конвенцию можно сделать проще и круче

```
public function up()
{
    Schema::create('posts', function (Blueprint $table) {
        $table->id();
        $table->string('title');
        $table->string('content');
        $table->string('image')->nullable();
        $table->boolean('status')->default(1);
        $table->unsignedBigInteger('category_id')->nullable();
        $table->timestamps();
        $table->softDeletes();

        $table->unsignedBigInteger('category_id');
    });
}
```

Создаем обычный атрибут, который будем связывать (имя должно совпадать с именем модели) (category - Category) (ед. числ)

```
_posts_table
> php artisan make:model Category АКТИВ
```

Создадим модель с привязкой (один ко многим)

```
Category.php
Post.php
User.php
> Providers
> bootstrap
> config
< database
> factories
< migrations
> 2014_10_12_000000_create_users_table.php
> 2014_10_12_100000_create_password_rese...
> 2019_08_19_000000_create_failed_jobs_tab...
> 2019_12_14_000001_create_personal_acces...
> 2022_08_08_093452_create_posts_table.php
> 2022_08_08_093908_create_categories_table...
```

Получаем

```
class CreateCategoriesTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('categories', function (Blueprint $table) [
            $table->id();
            $table->string('title');
            $table->timestamps();
        ]);
    }
}
```

Содержание категории

```
PS C:\OpenServer\domains\laravel\relations2> php artisan make:model Tag -m
Создаем теги (многие ко многим)
```

```
    /**
     * Run the migrations.
     */
    public function up()
    {
        Schema::create('tags', function (Blueprint $table) [
            $table->id();
            $table->string('title');
            $table->timestamps();
        ]);
    }
}
```

Содержание тегов

```
PS C:\OpenServer\domains\laravel\relations2> php artisan make:migration create_post_tag_table --create
Created Migration: 2022_08_08_094437_create_post_tag_table
```

Для связи многие ко многим нужно создать отдельную МИГРАЦИЮ БЕЗ МОДЕЛИ(имя в ед. числе!!)

```

class CreatePostTagTable extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('post_tag', function (Blueprint $table) {
            $table->id();
            $table->timestamps();
        });
    }
}

```

Получаем

```

public function up()
{
    Schema::create('post_tag', function (Blueprint $table) {
        $table->id();

        $table->unsignedBigInteger('post_id');
        $table->unsignedBigInteger('tag_id');

        $table->timestamps();
    });
}

```

Содержание таблицы post_tag

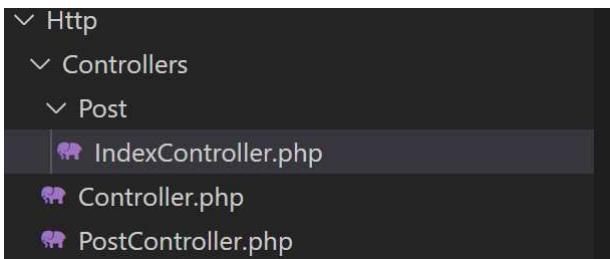
НА ЭТОМ ВСЕ ларавел уже все сам понимает(если правильно задать имена)

**ЕСЛИ ВЫЛАЗИЕТ ОШИБКА, ТО НУЖНО УДАЛИТЬ ТСБЛИЦУ
МИГРАЦИИ В БД И УДАЛИТЬ ТАБЛИЦУ**

НЕ РАБОТАЕТ

3.11 Однометодные контроллеры

Однометодные и многометодные контроллеры одинаково хорошо
Просто считается, что однометодные контроллеры считаются более
грамотными



Создаем папку, где будут наши однометодные контроллеры и именуем контроллеры по названию функции

```
<?php

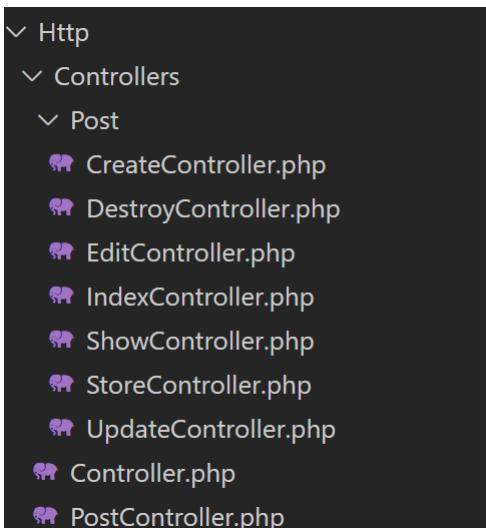
namespace App\Http\Controllers\Post;
use App\Http\Controllers\Controller;

use App\Models\Post;

class IndexController extends Controller
{
    public function __invoke(){
        $posts = Post::all();
        return view('post.index', compact('posts'));
    }
}
```

Очищаем контроллер и добавляем `__invoke` который в концепции ООП РНР выполняется первый и функцию можно никак не называть т. к. она - название файла

Также обязательно меняем `namespace` и `use`



Переносим все функции одного контроллера по разным контроллерам

```

<?php

use Illuminate\Support\Facades\Route;
use App\Http\Controllers\Post;

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

Route::group(['namespace' => 'Post'], function(){
    Route::get('/posts', 'IndexController')->name('post.index');
    Route::get('/posts/create', 'CreateController')->name('post.create');
    Route::post('/posts', 'StoreController')->name('post.store');
    Route::get('/posts/{post}', 'ShowController')->name('post.show');
    Route::get('/posts/{post}/edit', 'EditController')->name('post.edit');
    Route::patch('/posts/{post}', 'UpdateController')->name('post.update');
    Route::delete('/posts/{post}', 'DestroyController')->name('post.destroy');
});|

```

Содержание роутеров

3.12 Класс Request

Основная задача этого класса проверка полученных данных на корректность и их передача дальше
Используется для оптимизации

```

class StoreController extends Controller
{
    public function __invoke()
    {
        $data = request()->validate([
            'title' => 'required|string',
            'content' => 'string',
            'image' => 'string',
            'category_id' => '',
            'tags' => '',
        ]);
        $tags = $data['tags'];
        unset($data['tags']);

        $post = Post::create($data);
    }
}

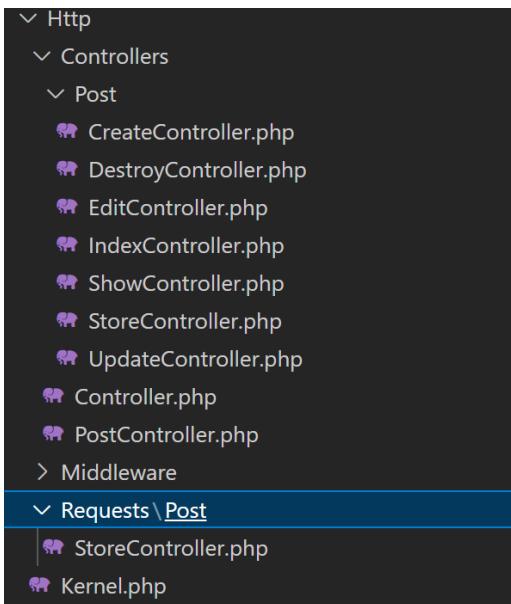
```

Такой код на профессиональной арене считается ПЛОХИМ И НЕПРОФЕССИОНАЛЬНЫМ

Тут мы используем метод request() но так делать тоже не оч. круто

```
php artisan make:request Post/StoreRequest
```

Создаем реквест вот так (имя реквеста - имя контроллера, где будет реквест)



```
public function authorize()
{
    return true;
}
```

В реквесте меняем FALSE на TRUE тем самым включая его

```
public function rules()
{
    return [
        'title' => 'string',
        'content' => 'string',
        'image' => 'string',
        'category_id' => 'integer',
        'tags' => ''
    ];
}
```

Добавляем правила

```
<?php

namespace App\Http\Controllers\Post;
use App\Http\Controllers\Controller;
use App\Http\Requests\Post\StoreRequest;
use App\Models\Post;

class StoreController extends Controller
{
    public function __invoke(StoreRequest $request){
        $data = $request -> validated();

        $tag = $data['tags'];
        unset($data['tags']);

        $post = Post::create($data);

        $post->tags()->attach($tag);

        return redirect() -> route('post.index');
    }
}
```

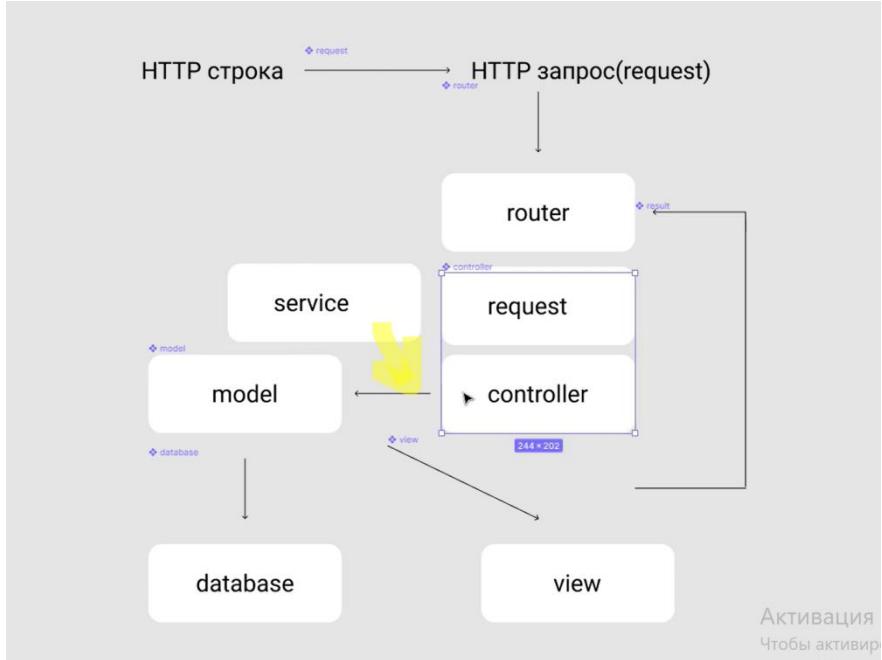
Меняем контроллер

ТАК МЫ КРУТО ОПТИМИЗИРОВАЛИ КОД

3.13 Класс Service

Берет на себя обязанности по CRUD

Логика - берем алгоритм работы с бд который связан с отдачей результата и бд



SERVICE - ПРОСЛОЙКА МЕЖДУ КОНТРОЛЛЕРОМ И МОДЕЛЬЮ

Прямая обязанность - грамотное взаимодействие с бд

Сервис помогает контроллеру взаимодействовать с моделью

```
class StoreController extends Controller
{
    public function __invoke(StoreRequest $request)
    {
        $data = $request->validated();
        $tags = $data['tags'];
        unset($data['tags']);

        $post = Post::create($data);

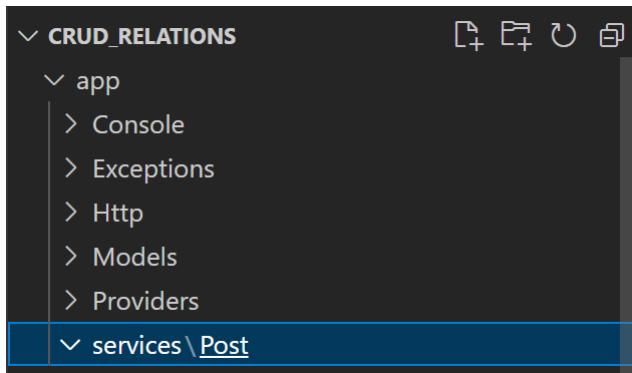
        $post->tags()->attach($tags);

        return redirect()->route(route: 'post.index');
    }
}
```

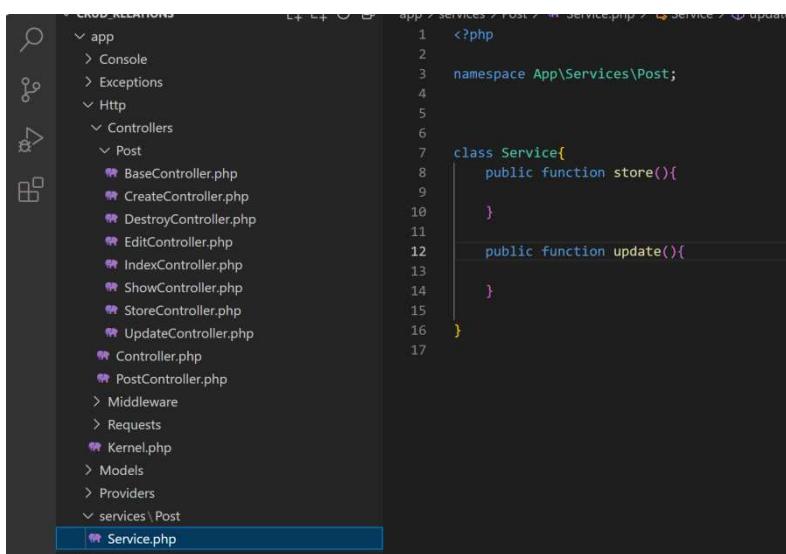
На самом деле вот такой код - нарушение норм (в будущем будет мусор и много путаницы)

По-хорошему нужно вынести работу с моделью в отдельное место (мы уже вынесли валидацию)

Сервис - концепт не ларавел а общепринятый концепт разработки на PHP



Создаем папку на территории app



Создаем в папке класс

```
namespace App\Http\Controllers\Post;

use App\Services\Post\Service;

class BaseController
{
    public $service;

    public function __construct(Service $service){
        $this->service = $service;
    }
}
```

Создаем базовый контроллер для подключения сервиса к контроллерам СО СВОЙСТВОМ И КОНСТРУКТОРОМ

1 присвоили свойству базового контроллера экземпляр на основании класса Service(стал объектом на основании класса)

2 создаем объект от класса сервис и присваиваем его в переменную

```
class BaseController extends Controller
{
    public $service;

    public function __construct(Service $service){
        $this->service = $service;
    }
}
```

Прописываем базовому контроллеру наследование от класса контроллера

```
class CreateController extends BaseController
{
    public function __invoke()
    {
        $category = Category::all();
        $tags = Tag::all();
        return view('post.create', compact('category', 'tags'));
    }
}
```

Меняем наследование в контроллеров

```
<?php

namespace App\Services\Post;
use App\Models\Post;

class Service{
    public function store($data){

        $tag = $data['tags'];
        unset($data['tags']);

        $post = Post::create($data);

        $post->tags()->attach($tag);
    }

    public function update(){

    }
}
```

Содержание класса сервис

```

class StoreController extends BaseController
{
    public function __invoke(StoreRequest $request){
        $data = $request -> validated();

        $tag = $data['tags'];
        unset($data['tags']);

        $this->service->store();

        $post = Post::create($data);

        $post->tags()->attach($tag);

        return redirect() -> route('post.index');
    }
}

```

Теперь можно вызвать класс сервис и у него метод стор

```

class StoreController extends BaseController
{
    public function __invoke(StoreRequest $request){
        $data = $request -> validated();
        $this->service->store($data);
        return redirect() -> route('post.index');
    }
}

```

Меняем наш контроллер

КОНТРОЛЛЕР - МЕСТО ПЕРЕСЕЧЕНИЯ МНОГИХ ЛОГИК ПОЭТОМУ
ЕГО НЕ НУЖНО НАГРУЖАТЬ ЕГО НУЖНО РАСПРЕДЕЛЯТЬ ПО
РЕКВЕСТАМ СЕРВИСАМ И ТД

3.14 Классы Factory и Seed

Деплой - Развёртывание программного обеспечения — это все действия, которые делают программную систему готовой к использованию. Данный процесс является частью жизненного цикла программного обеспечения.

Деплой — процесс «разворачивания» веб-сервиса, например, сайта, в рабочем окружении. Рабочее окружение — место, где сайт запускается и доступен для запросов. Это может быть как готовый хостинг, так и своя собственная серверная инфраструктура. Деплоятся не только веб-сервисы, но любые сервисы, доступные по сети.

3.14.1 Класс seed

Нужен для развертывания каких-либо данных при деплое на сервер

```
ПРОВОДНИК ... DatabaseSeeder.php ×
CRUD_RELATIONS ...
Controller.php
PostController.php
Middleware ...
Requests\Post
StoreRequest.php
UpdateRequest.php
Kernel.php
Models ...
Providers ...
services\Post
Service.php
bootstrap ...
config ...
database ...
factories ...
migrations ...
seeders ...
DatabaseSeeder.php
```

```
1 <?php
2
3 namespace Database\Seeders;
4
5 use Illuminate\Database\Seeder;
6
7 class DatabaseSeeder extends Seeder
8 {
9     /**
10      * Seed the application's database.
11      *
12      * @return void
13      */
14     public function run()
15     {
16         // \App\Models\User::factory(10)->create();
17     }
18 }
```

Расположение и содержание

```
PS C:\OpenServer\domains\laravel\crud_relations> php artisan migrate --help
Description:
Run the database migrations

Usage:
migrate [options]

Options:
--database[=DATABASE]           The database connection to use
--force                           Force the operation to run when in production
--path[=PATH]                     The path(s) to the migrations files to be executed (multiple values allowed)
)
--realpath                        Indicate any provided migration file paths are pre-resolved absolute paths
--schema-path[=SCHEMA-PATH]       The path to a schema dump file
--pretend                          Dump the SQL queries that would be run
--seed                            Indicates if the seed task should be run
--seeder[=SEEDER]                 The class name of the root seeder
```

Как найти команду (позволяет запустить все что находится в методе)

```
...
*/
public function run()
{
    echo 'aaaaaaaaaaaaaaaaaaaa';
    // \App\Models\User::factory(10)->create();
}
```

```
--ansi                         Force ANSI output
--no-ansi                        Disable ANSI output
--n, --no-interaction             Do not ask any interactive question
--env[=ENV]                       The environment the command should run under
--vvv|vv|v, --verbose              Increase the verbosity of messages: 1 for normal output, 2 for more verbose output and 3 for debug

D:\lessons\first_project>php artisan migrate --seed
Nothing to migrate.
aaaaaaaaaaaaaaaaaaaaaaaDatabase seeding completed successfully.
```

Результат

```
public function run()
{
    // \App\Models\User::factory(10)->create();
}
```

Причем в методе идет обращение к классу factory
ПОГОВОРИМ ДАЛЬШЕ ПОТОМ

3.14.2 Factory (фабрика)

Это изначальная ФАБРИКА для пользователей, которую предоставляет ларавел

Для постов категорий и тегов нужно создать СВОИ ФАБРИКИ

Основное назначение фабрики- дать готовые объекты на выходе

Объекты могут сразу отправиться в базу или просто стать массивом (и т.д.)

В дальнейшем мы сможем работать с этими объектами



Расположение

3.14.3 Совместная работа

```
public function run()
{
    $category = [];
    $tag = [];
    Category::create();
    // \App\Models\User::factory(10)->create();
```

Можно все прописать в сиде НО НЕ НУЖНО ТК У НАС ЕСТЬ ФАБРИКА
ФАБРИКА сделает все автоматически

```
php artisan make:factory PostFactory --model=Post
```

Создадим фабрику для постов с привязкой к модели (можно добавить равно)(не работает)

```
use App\Models\Post;
use Illuminate\Database\Eloquent\Factories\Factory;

class PostFactory extends Factory
{
    protected $model = Post::class;
```

Добавляем модель вручную

```
namespace Database\Factories;

use App\Models\Post;
use Illuminate\Database\Eloquent\Factories\Factory;

class PostFactory extends Factory
{
    protected $model = Post::class;
    /**
     * Define the model's default state.
     *
     * @return array
     */
    public function definition()
    {
        return [
            //
        ];
    }
}
```

Содержание класса фактори

```
public function definition()
{
    return [
        "name" => "blabla",
        "old" => '19'
    ];
}
```

Этот метод при обращении к фабрики выдает массив

```
/*
public function run()
{
    \App\Models\User::factory(10)->create();
}
```

При вызове фабрики в сидре ОБРАЩЕНИЕ ИДЕТ К САМОЕ МОДЕЛИ

```
class Post extends Model
{
    use HasFactory;
```

Использование фабрики

```

namespace Database\Seeders;

use App\Models\Post;
use Illuminate\Database\Seeder;

class DatabaseSeeder extends Seeder
{
    /**
     * Seed the application's database.
     *
     * @return void
     */
    public function run()
    {
        $posts = Post::factory(10)->make();
        dd($posts);
        // \App\Models\User::factory(10)->create();
    }
}

```

Вызываем сид с фабрикой (не create а make чтобы ничего не создалось в бд а только вывелоось на экран)

```

PS C:\OpenServer\domains\laravel\crud_relations> php artisan migrate --seed
Nothing to migrate.
Illuminate\Database\Eloquent\Collection^ {#961
#items: array:10 [
  0 => App\Models\Post^ {#964
    #table: "posts"
    #guarded: []
    #connection: null
    #primaryKey: "id"
    #keyType: "int"
    +incrementing: true
    #with: []
    #withCount: []
    +preventsLazyLoading: false
    #perPage: 15
    +exists: false
    +wasRecentlyCreated: false
    #escapeWhenCastingToString: false
    #attributes: array:2 [
      "name" => "blabla"
      "old" => "19"
    ]
    #original: []
  }
]

```

Вывод на экран и вызов

Вернулись МОДЕЛИ с зарезервированными методами (В БАЗУ НЕ ПОШЛО ЭТО ПРОСТО МАССИВ С КЛАССОМ ПОСТ(поэтому есть зарезервированными методы))

```
public function definition()
{
    return [
        'title' => 'blabla',
        'content' => 'blablablabla',
        'image' => 'blablablabla',
        'likes' => random_int(1, 2000),
        'is_published' => 1,
        'category_id' => 1
    ];
}
```

Для того чтобы отправить созданный объект в бд у него должны быть те же атрибуты что и у таблицы (значения можно задавать рандомные!!)

```
public function run()
{
    $posts = Post::factory(10)->make();
    dd($posts);
    // \App\Models\User::factory(10)->create();
}
```

```
$posts = Post::factory(10)->create();
```

Создаем 10 объектов в бд

```
'category_id' => Category::get()->random()->id
```

Берем рандомную категорию

```
public function definition()
{
    return [
        'title' => $this->faker->title(gender: 20),
```

Рандомный заголовок!!!

```
D:\lessons\first_project>php artisan make:factory CategoryFactory
Factory created successfully.
```

```
D:\lessons\first_project>php artisan make:factory TagFactory
Factory created successfully.
```

Создадим фабрику для категорий и тегов

```
protected $model = Tag::class;

/**
 * Define the model's default state.
 *
 * @return array
 */
public function definition()
{
    return [
        'title' => $this->faker->word
    ];
}
```

Заполняем

```
public function run()
{
    Category::factory( ...parameters: 20)->create();
    Post::factory( ...parameters: 200)->create();
    Tag::factory( ...parameters: 50)->create();
```

Создаем

```
public function run()
{
    Category::factory( ...parameters: 20)->create();
    $tags = Tag::factory( ...parameters: 50)->create();
    $posts = Post::factory( ...parameters: 200)->create();

    foreach ($posts as $post) {
        $tagsIds = $tags->random( number: 5)->pluck( value: 'id');
        $post->tags()->attach($tagsIds);
    }
}
```

Привязываем

Pluck выцепит только айди и вернет массив айдишников

3.15 Пагинация в Laravel

Пагинация (Pagination) – это порядковая нумерация страниц, которая в основном размещается вверху либо внизу страниц сайта.

Пагинация ОЧЕНЬ сильно снижает нагрузку на сайт т. к. мы загружаем не сразу все 10000 постов на сайт, а по 10-20 на страницу

```
class IndexController extends BaseController
{
    public function __invoke()
    {
        $posts = Post::paginate(10);
        return view('post.index', compact('posts'));
    }
}
```

Меняем Post::all на вот это



Получаем

```
Illuminate\Database\Eloquent\Collection {#1424 ▾
  #items: array:200 [▶]
}
```

ЭТО МЫ ПОЛУЧАЕМ ПРИ DD ВСЕХ ПОСТОВ

```
Illuminate\Pagination\LengthAwarePaginator {#1021 ▾
  #total: 200
  #lastPage: 20
  #items: Illuminate\Database\Eloquent\Collection {#1235 ▶}
  #perPage: 10
  #currentPage: 1
  #path: "http://127.0.0.1:8000/posts"
  #query: []
  #fragment: null
  #pageName: "page"
  #onEachSide: 3
  #options: array:2 [▶]
}
```

А ПРИ ПАГИНАЦИИ ВОТ ЭТО(через этот метод можно отобразить ссылки)

```
IndexController.php index.blade.php X
resources > views > post > index.blade.php > ...
1 @extends('layouts.post')
2 @section('content')
3     @foreach($posts as $post)
4         <a href="{{route('post.show', $post->id)}}>{{$post->id}}. {{$post->title}}</a>
5     @endforeach
6     <div>
7         {{$posts->links()}}
8     </div>
9     @endsection()
```

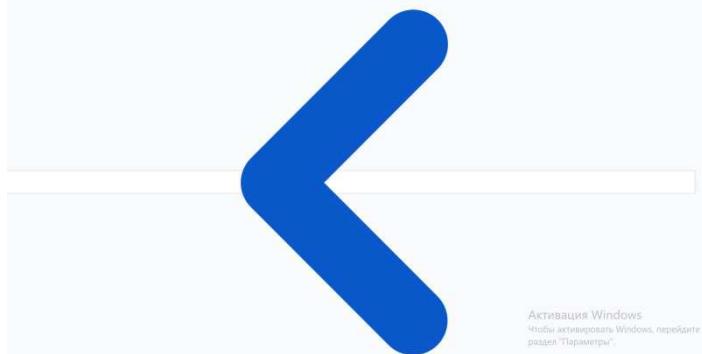
Добавляем ссылки

SaintAdmin Создать пост

[11. iure iure quia](#)
[12. error magnam vero](#)
[13. repellat voluptatum et](#)
[14. vel vitae vitae](#)
[15. eveniet quam qui](#)
[16. maiores totam nesciunt](#)
[17. consequuntur doloribus qui](#)
[18. accusantium omnis dicta](#)
[19. optio tempora odit](#)
[20. alias soluta accusamus](#)

[« Previous](#) [Next »](#)

Showing 11 to 20 of 200 results



Получаем вот что (выглядит все так некрасиво, потому что по умолчанию идет шаблон тейлвинд а не бутстррап)

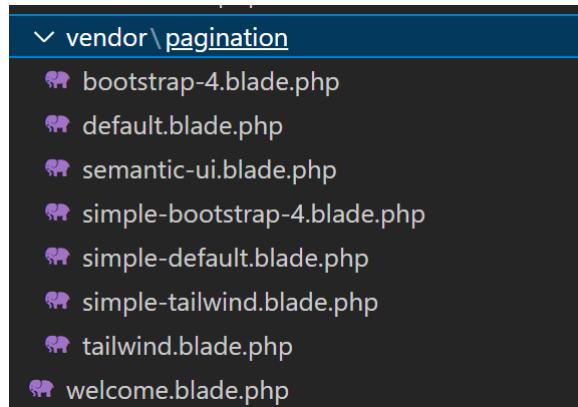
Database: Pagination

Документация

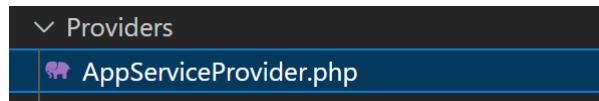
```
PS C:\OpenServer\domains\laravel\crudapi> php artisan vendor:publish --tag=laravel-pagination
Copied Directory [\vendor\laravel\framework\src\Illuminate\Pagination\resources\views] To [\resources\views\ve
ndor\pagination]
Publishing complete.
```

Выполняем эту команду

Происходит линковка (копирование из одной папки в другую для удобства чтения)



Получаем шаблоны пагинации



Заходим в этот файл и меняем там шаблон

ВСЕ ЧИТАТЬ В ДОКУМЕНТАЦИИ

```
use Illuminate\Pagination\Paginator;
use Illuminate\Support\ServiceProvider;

class AppServiceProvider extends ServiceProvider
{
    /**
     * Register any application services.
     *
     * @return void
     */
    public function register()
    {
        //
    }

    /**
     * Bootstrap any application services.
     *
     * @return void
     */
    public function boot()
    {
        Paginator::defaultView('vendor.pagination.bootstrap-4');
    }
}
```

Дефолтная пагинация

- [1. impedit id nihil](#)
- [2. harum natus ducimus](#)
- [3. dolor qui voluptas](#)
- [4. debitis distinctio nobis](#)
- [5. repellendus ut at](#)
- [6. natus repellat rem](#)
- [7. rem ipsa ut](#)
- [8. sequi in doloremque](#)
- [9. sed nobis culpa](#)
- [10. architecto reiciendis aut](#)

[«](#) [1](#) [2](#) [3](#) [4](#) [5](#) [6](#) [7](#) [8](#) [9](#) [10](#) [...](#) [19](#) [20](#) [»](#)

Получаем

ВСЕ ШАБЛОНЫ МОЖНО МЕНЯТЬ И ДОБАВЛЯТЬ НОВЫЕ

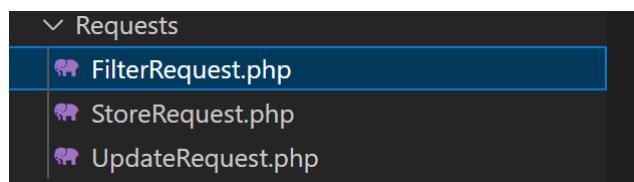
3.16 Шаблон Filter, фильтрация данных в Laravel

3.16.1 НЕИСПОЛЬЗУЕМЫЙ ШАБЛОН

```
class IndexController extends BaseController
{
    public function __invoke()
    {
        $posts = Post::where('is_published', 1)
            ->where('category_id', 1)
            ->get();
    }
}
```

Обычная фильтрация данных

Так делать не круто и это идиотизм т. к. нельзя ничего передать



Правильная фильтрация делается через реквесты т. к. они проверяют данные

127.0.0.1:8000/posts?category_id=5

Это называется query string или строка запроса (гет запрос после ?)

```
class IndexController extends BaseController
{
    public function __invoke(FilterRequest $request){
        $data = $request->validated();

        $query = Post::query();
        dd($query);

        if(isset($data['category_id'])){
            $query->where('category_id', $data['category_id']);
        }

        $post = $query->get();

        dd($post);
    }
}
```

Так по получаем наши данные из реквеста и валидируем их

Далее конструируем query динамичный запрос

Проверяем условия и строим запрос

Выводим результат

```
12     * @return bool
13     */
14    public function authorize()
15    {
16        return true;
17    }
18
19    /**
20     * Get the validation rules that apply to the request
21     *
22     * @return array
23     */
24    public function rules()
25    {
26        return [
27            'title' => 'string',
28            'content' => 'string',
29            'image' => 'string',
30            'category_id' => 'integer',
31            'tags' => 'array'
32        ];
33    }
34}
```

Фильтр

```

 Illuminate\Database\Eloquent\Collection {#351 ▶
   #items: array:27 [▼
     0 => App\Mode...\Post {#352 ▶}
     1 => App\Mode...\Post {#353 ▶}
     2 => App\Mode...\Post {#354 ▶}
     3 => App\Mode...\Post {#355 ▶}
     4 => App\Mode...\Post {#356 ▶}
     5 => App\Mode...\Post {#357 ▶}
     6 => App\Mode...\Post {#358 ▶}
     7 => App\Mode...\Post {#359 ▶}
     8 => App\Mode...\Post {#360 ▶}
     9 => App\Mode...\Post {#361 ▶}
    10 => App\Mode...\Post {#362 ▶}
    11 => App\Mode...\Post {#363 ▶}
    12 => App\Mode...\Post {#364 ▶}
    13 => App\Mode...\Post {#365 ▶}
    14 => App\Mode...\Post {#366 ▶}
    15 => App\Mode...\Post {#367 ▶}
    16 => App\Mode...\Post {#368 ▶}
    17 => App\Mode...\Post {#369 ▶}
    18 => App\Mode...\Post {#370 ▶}
    19 => App\Mode...\Post {#371 ▶}
    20 => App\Mode...\Post {#372 ▶}
    21 => App\Mode...\Post {#373 ▶}
    22 => App\Mode...\Post {#374 ▶}
    23 => App\Mode...\Post {#375 ▶}
    24 => App\Mode...\Post {#376 ▶}
    25 => App\Mode...\Post {#377 ▶}
    26 => App\Mode...\Post {#378 ▶}
  ]
  #escapeWhenCastingToString: false
}

```

Получаем

```

if(isset($data['title'])){
    $query->where(['title', 'like', "%{$data['title']}%"]);
}

```

Такая конструкция будет искать не только по полному слову но и при любом сочетании букв введенных букв в любом месте предложения

```

class IndexController extends BaseController
{
    public function __invoke(FilterRequest $request){
        $data = $request->validated();

        $query = Post::query();
        // dd($query);

        if(isset($data['category_id'])){
            $query->where('category_id', $data['category_id']);
        }

        if(isset($data['title'])){
            $query->where('title', 'like', "%{$data['title']}%");
        }

        $post = $query->get('title');

        dd($post);
        $posts = Post::paginate(10);
        return view('post.index', compact('posts'));
    }
}

```

Если по ключу запросы не приходят, то условия просто игнорируются

127.0.0.1:8000/posts?title=aut&category_id=12

Можно делать вот так

3.16.2 Используемый шаблон

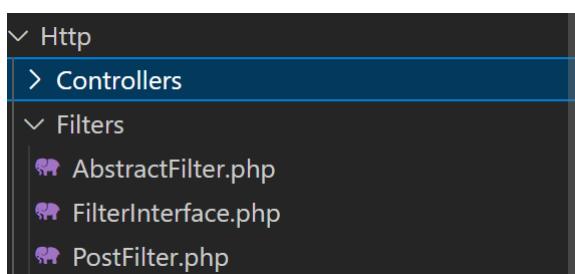
Шаблон выше не эффективен т. к. уже давно есть заранее придуманный шаблон

<https://www.youtube.com/watch?v=cL1eXKsnRJI&t=163s> ВИДЕО О ФИЛЬТРАХ

ЭТОТ ШАБЛОН СЛОЖНЫЙ
УЧИТЬ ЕГО НЕ НАДО

НУЖНО ПРОСТО ЗАПОМНИТЬ И ПОНЯТЬ, ЧТО КАК СТОИТ
ИЛИ ТУПО КОПИРОВАТЬ

Данный шаблон лишь один из реализаций фильтров

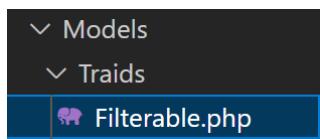


Создаем вот такой шаблон (2 класса 1 интерфейс) (два основных и один дополнительный)

Далее мы заполняем два обязательных файла

ФАЙЛЫ С КОММЕНТРИЕМИ В КОНЦЕ ДОКУМЕНТАЦИИ
(ФИЛЬТРЫ)

Далее в моделях мы должны создать трейды(контракты)



Название по конвенции (able - для всех трейдов)

Контракты в Laravel — это набор интерфейсов, которые описывают основной функционал, предоставляемый фреймворком. Например, контракт `Illuminate\Contracts\Queue\Queue` определяет методы, необходимые для организации

очередей, в то время как контракт `Illuminate\Contracts\Mail\Mailer` определяет методы, необходимые для отправки электронной почты.

Трейды подключаются к моделям или класса для дополнения их функционала

```
class Post extends Model
{
    use HasFactory;
    use Filterable;
    use SoftDeletes;
```

Далее нужно создать фильтр на основе класса в контроллере

```
class IndexController extends BaseController
{
    public function __invoke(FilterRequest $request){
        $data = $request->validated();

        $filter = app()->make(PostFilter::class, ['queryParams'=>array_filter($data)]);

        $post = Post::filter($filter);
        dd($post);

        $posts = Post::paginate(10);
        return view('post.index', compact('posts'));
    }
}

$post = Post::filter($filter)->paginate(10);
```

ТАК ДЕЛАЕТСЯ ПАГИНАЦИЯ

The screenshot shows a Laravel application's admin panel. At the top, there is a header with the text "SaintAdmin" and "Создать пост". Below the header, a list of post titles is displayed, each with a blue underline. The titles are: "1. voluptatum architecto autem", "5. et molestiae autem", "13. fuga aut aut", "19. aut quod velit", "64. natus aut et", "82. nobis aut beatae", "93. ullam et autem", "95. sed aut id", "101. aut ut voluptatem", and "114. aut rem autem". Below the list is a navigation bar with three buttons: a left arrow, a central button containing the number "1" (which is highlighted in blue), and a right arrow. The background of the page is white.

Результат

```
@extends('layouts.main')
@section('content')
    <div>
        <div>
            <a href="{{ route('post.create') }}" class="btn btn-primary mb-3">Add one</a>
        </div>

        @foreach($posts as $post)
            <div><a href="{{ route('post.show', $post->id) }}>{{ $post->id }}. {{ $post->title }}</a>
        @endforeach

        <div class="mt-3">
            {{ $posts->withQueryString()->links() }}
        </div>
    </div>
@endsection
```

Чтобы при переходе по страницам запросы не пропадали



127.0.0.1:8000/posts?title=ut&page=2

НА ЭТОМ ВСЕ

УЧИТЬ ЭТО НЕ НАДО

НУЖНО ПРОСТО ПОНЯТЬ, ЧТО КАК РАБОТАЕТ И
ОРИЕНТИРОВАТЬСЯ

3.17 Admin LTE в Laravel, устанавливаем админку

Админки устанавливаются +- одинаково

Мы возьмём админ лте тк она простая и популярная (можно использовать в проекте)

Скачиваем adminLte и распаковываем в папку

https://www.youtube.com/watch?v=w9kUQZDAWas&list=PLd2_Os8Cj3t8pnG4ubQemoqnTwf0VFEtU&index=32

Смотреть видос или прошлые проекты

Спасибо за урок, чтобы не менять в шаблоне все пути, просто в head можно прописать, например под <title></title> вот такой код: <base href="{{ asset('') }}>

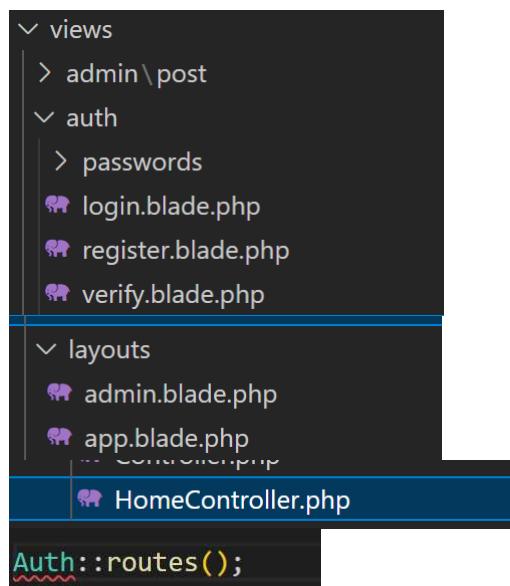
Я установил AdminLTE через композер. Команда для установки на их сайте в разделе Docs / Installation: после установки не нужно проставлять нигде {{ assets() }}, все прекрасно работает)

3.18 Авторизация Laravel

В ларавел есть нативные средства авторизации

```
PS C:\OpenServer\domains\laravel\laravel> php artisan ui:auth  
Authentication scaffolding generated successfully.
```

Подключение



Подключаемые файлы

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

Добавляем на главную страницу homecontroller

Получаем

MySQL вернула пустой результат (т.е. ноль строк). (Запрос занял 0,0004 сек.)

`SELECT * FROM `users``

Автоматом уже заранее есть база данных с пользователями

ВСЕ ЭТО ЛЕГКО РЕДАКТИРУЕТСЯ В ИСХОДНЫХ ФАЙЛЫХ

```

protected function validator(array $data)
{
    return Validator::make($data, [
        'name' => ['required', 'string', 'max:255'],
        'email' => ['required', 'string', 'email', 'max:255', 'unique:users'],
        'password' => ['required', 'string', 'min:8', 'confirmed'],
    ]);
}

```

НАЗВАНИЯ ДОЛЖНЫ БЫТЬ ПРАВИЛЬНЫМИ И ПОСМОТРЕТЬ ИХ
МОЖНО ТУТ

`{ {auth()->user()->name} }`

Поменять <https://stackoverflow.com/questions/73180945/jetstream-css-and-js-not-working-and-showing-viteresources-css-app-css-re>

3.19 Класс Middleware в Laravel (Роли авторизации)

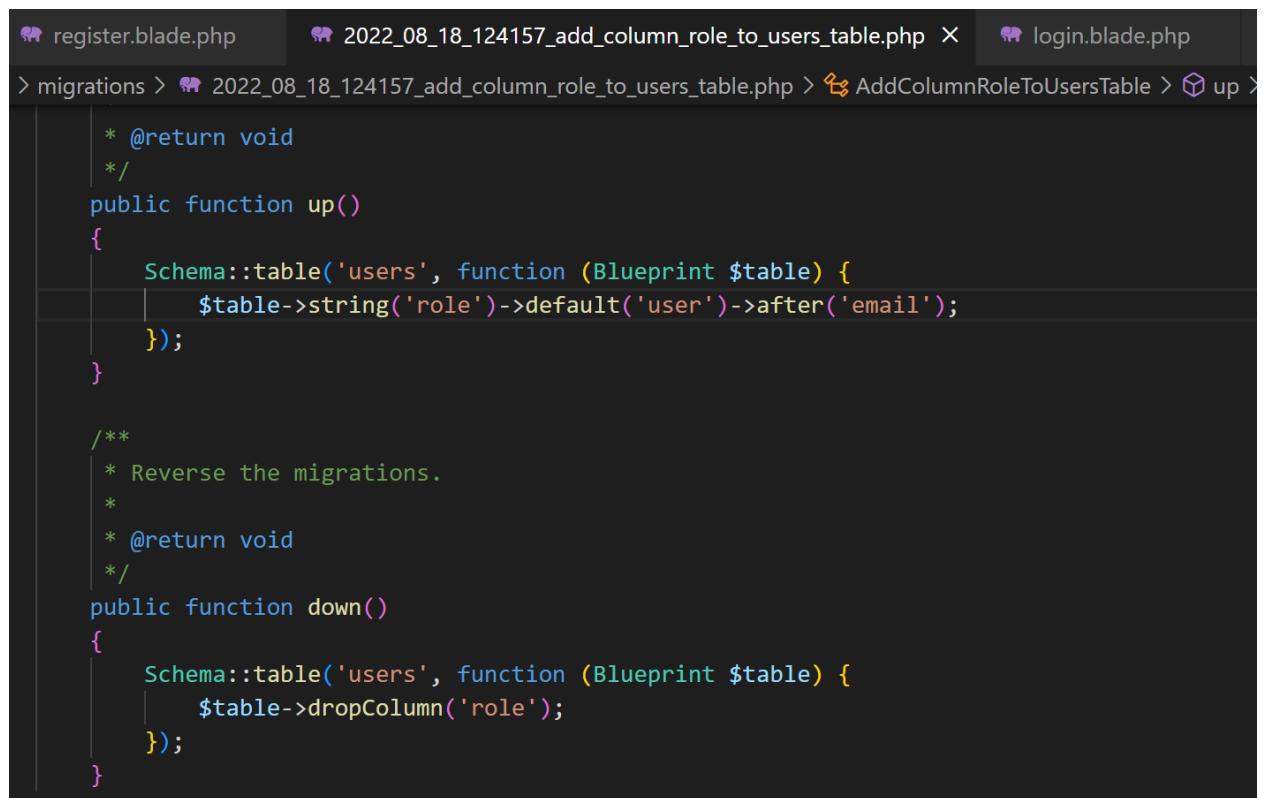
Пока что наша регистрация просто ни о чем

Правильная регистрация осуществляется с распределением ролей чтобы простой юзер не попал в админку

Middleware - в переводе - ПОСРЕДНИК(что-то между) МЕЖДУ ЗАПРОСОМ И КОНТРОЛЛЕРОМ

Его прямая обязанность допускать работу в контроллере или не допускать (ПРОВЕРЯТЬ РОЛЬ)

1 Меняем миграцию users



The screenshot shows a code editor with three tabs at the top: 'register.blade.php', '2022_08_18_124157_add_column_role_to_users_table.php', and 'login.blade.php'. The middle tab is active and displays a PHP migration file. The code defines two database schema alterations: an 'up' method that adds a 'role' column to the 'users' table with a default value of 'user' and an 'email' constraint, and a 'down' method that drops the 'role' column. The code uses the Schema::table() and Blueprint \$table objects.

```
* @return void
*/
public function up()
{
    Schema::table('users', function (Blueprint $table) {
        $table->string('role')->default('user')->after('email');
    });
}

/**
 * Reverse the migrations.
 *
 * @return void
 */
public function down()
{
    Schema::table('users', function (Blueprint $table) {
        $table->dropColumn('role');
    });
}
```

2 Создаем посредника

```
> php artisan make:middleware AdminPanelMiddleware
```

The screenshot shows a file browser window with a tree view. Under the 'Middleware' folder, there are several files: AdminPanelMiddleware.php, Authenticate.php, EncryptCookies.php, PreventRequestsDuringMaintenance.php, RedirectIfAuthenticated.php, TrimStrings.php, TrustHosts.php, TrustProxies.php, and VerifyCsrfToken.php. Below this, the content of AdminPanelMiddleware.php is displayed in a code editor:

```
<?php

namespace App\Http\Middleware;

use Closure;
use Illuminate\Http\Request;

class AdminPanelMiddleware
{
    /**
     * Handle an incoming request.
     *
     * @param \Illuminate\Http\Request $request
     * @param Closure(\Illuminate\Http\Request): (\Illuminate\Http\Response|\Illuminate\Http\RedirectResponse) $next
     */
    public function handle(Request $request, Closure $next)
    {
        return $next($request);
    }
}
```

Содержимое посредника

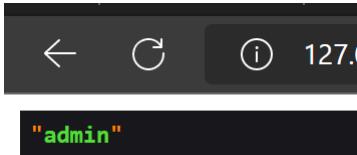
В нем у нас происходит некая проверка и по ее завершению мы что-то выполняем или пересылаем при запрете

```
$user = User::where('is_auth', 1)  
auth()->
```

Данный хелпер вытягивает пользователя, который заходит на сайт ИМЕННО НАШ НАШЕМ КОМПЕ

```
public function handle(Request $request, Closure $next)  
{  
    dd(auth()->user()->role);  
    return $next($request);  
}
```

Так мы можем вытянуть роль



```
public function handle(Request $request, Closure $next)
{
    if(auth()->user()->role !== 'admin'){
        return redirect('/admin');
    }
    return $next($request);
}
```

Делаем проверку

3 Регистрируем посредника



Сюда

```
protected $routeMiddleware = [
    'auth' => \App\Http\Middleware\Authenticate::class,
    'auth.basic' => \Illuminate\Auth\Middleware\AuthenticateWithBasicAuth::class,
    'cache.headers' => \Illuminate\Http\Middleware\SetCacheHeaders::class,
    'can' => \Illuminate\Auth\Middleware\Authorize::class,
    'guest' => \App\Http\Middleware\RedirectIfAuthenticated::class,
    'password.confirm' => \Illuminate\Auth\Middleware\RequirePassword::class,
    'signed' => \Illuminate\Routing\Middleware\ValidateSignature::class,
    'throttle' => \Illuminate\Routing\Middleware\ThrottleRequests::class,
    'verified' => \Illuminate\Auth\Middleware\EnsureEmailIsVerified::class,
    'admin' => AdminPanelMiddleware::class
];
```

Тут

3 Прописываем атрибуты в роутах

```
Route::group(['namespace' => 'Admin', 'prefix' => 'admin', 'middleware' => 'admin'], function(){
    Route::group(['namespace' => 'Post'], function(){
        Route::get('/post', 'IndexController')->name('admin.post.index');
    });
});
```

Теперь всегда при переходе в админку будет проходить проверка на админа

3.20 Класс Policy

Middleware - разрешение и запрет доступа

Policy - демонстрация и скрытие доступа (ссылки, контент, функции и т.д) - СКРЫТЬ ЭЛЕМЕНТЫ ИСХОДЯ ИЗ РОЛИ ПОЛЬЗОВАТЕЛЯ

Например, чтобы постоянно не набирать в строке запроса админ, а чтобы у админа уже была кнопка перехода

```
<nav>
    <li class="nav-item">
        <a href="{{route('admin.post.index')}}">АдминПанель</a>
    </li>
</nav>
```

Создаем какой-нибудь элемент, который должен видеть только админ

```
php artisan make:policy AdminPolicy -m User
```

Создаем СЛЕДОВАТЕЛЯ с привязкой к модели пользователь

```
/*
public function view(User $user, User $model)
{
    //
}
```

Все функции следователя относятся к CRUD

```
public function forceDelete(User $user, User $model)
{
    //
}
```

Удаляет пост с учетом софт делит (полностью удалит)

```
public function view(User $user, User $model)
{
    if($model->role !== 'admin'){
        return false;
    }
}
```

Отображение и скрытие элементов происходит через метод view

```
public function view(User $user, User $model)
{
    return $model->role === 'admin';
}
```

Можно так т.к. метод уже делает проверку

СЛЕДОВАТЕЛЬ МОЖЕТ ИСПОЛЬЗОВАТЬСЯ В КОНТРОЛЛЕРАХ И ВЬЮВАХ

```

ПРОВОДНИК ... home.blade.php AdminPolicy.php AuthServiceProvider.php X
LARAVEL app > Providers > AuthServiceProvider.php > AuthServiceProvider > $policies
    Controllers ...
        Middleware ...
            AdminPanelMiddleware.php
            Authenticate.php
            EncryptCookies.php
            PreventRequestsDuringMaint...
            RedirectIfAuthenticated.php
            TrimStrings.php
            TrustHosts.php
            TrustProxies.php
            VerifyCsrfToken.php
        Requests ...
        Service ...
        Kernel.php
    Models ...
    Policies ...
        AdminPolicy.php
    Providers ...
        AppServiceProvider.php
        AuthServiceProvider.php

```

```

3     namespace App\Providers;
4
5     use App\Models\User;
6     use App\Policies\AdminPolicy;
7     use Illuminate\Foundation\Support\Providers\AuthServiceProvider as ServiceProvider;
8     use Illuminate\Support\Facades\Gate;
9
10    class AuthServiceProvider extends ServiceProvider
11    {
12        /**
13         * The policy mappings for the application.
14         *
15         * @var array<class-string, class-string>
16         */
17        protected $policies = [
18            // 'App\Models\Model' => 'App\Policies\ModelPolicy',
19            User::class => AdminPolicy::class
20        ];
21
22        /**
23         * Register any authentication / authorization services.

```

Выходные данные КОНСОЛЬ ОТЛАДКИ ТЕРМИНАЛ

Регистрируем следователя

```

class IndexController extends BaseController
{
    public function __invoke(){
        $this->authorize('view', auth()->user());
        $posts = Post::paginate(10);
        $count = 1;
        return view('post.index', compact('posts', 'count'));
    }
}

```

Делаем в контроллере проверку на роль



Ошибка если роль не совпадает с админом

```

@can('view', auth()->user())
<nav>
    <li class="nav-item">
        <a href="{{route('admin.post.index')}}">АдминПанель</a>
    </li>
</nav>
@endcan

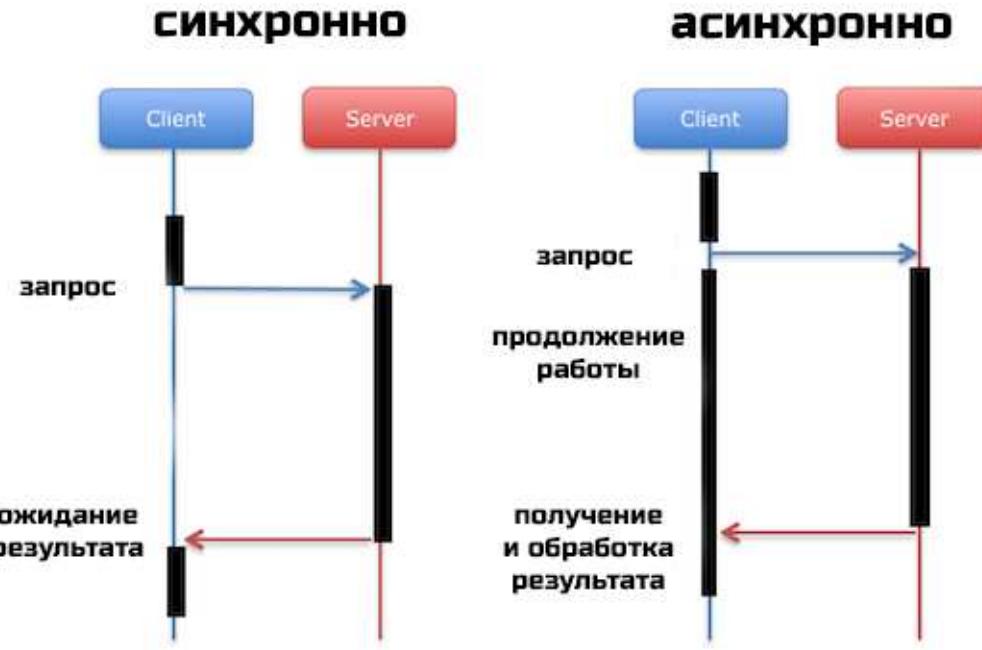
```

ТАК МОЖНО СКРЫТЬ ЭЛЕМЕНТЫ

3.21 Асинхронный CRUD в Laravel. Приложение Postman.

Postman – приложение, имитирующее запросы с фронтенда

ajax – технология работы загрузчика и обновление информации на сайте без перезагрузки сайта) работает это благодаря АСИНХРОННЫМ ЗАПРОСАМ



<https://russianblogs.com/article/7728163812/>

<https://www.youtube.com/watch?v=3D2kYmEa8rk>

Постман используется для теста запросов перед продакшином (реакция ответа на асинхронный запрос с сайта) (имитация запросов с фронта)

The screenshot shows the Postman application interface. On the left, there's a sidebar with various sections like Collections, APIs, Environments, Mock Servers, Monitors, Flows, and History. A prominent 'Create a collection for your requests' section is displayed, along with a 'Create collection' button. The main workspace shows a collection named 'SAINT' containing a folder 'My first collection' with two sub-folders: 'First folder inside collection' and 'Second folder inside collection'. Each folder contains several requests. To the right, an 'Untitled Request' is being edited with a 'GET' method and an 'Enter request URL' field. Below it, tabs for Params, Authorization, Headers (6), Body, Pre-request Script, Tests, and Settings are visible. A table for Query Params is shown with one row: 'Key' (Value) and 'Value' (Value). A 'Send' button is at the top right. The bottom right features a cartoon character holding a rocket, and the text 'Enter the URL and click Send to get a response'.

Интерфейс

The screenshot shows the Postman request editor. At the top, a 'GET' method and the URL 'http://127.0.0.1:8000/posts' are specified. A 'Send' button is on the right. Below the URL, tabs for Body, Cookies (2), Headers (9), and Test Results are visible, with 'Body' being the active tab. The Body content is displayed in a code editor with line numbers, showing the HTML structure of the response. The response body is as follows:

```

1 <html lang="en">
2
3   <head>
4     <meta charset="UTF-8">
5     <meta http-equiv="X-UA-Compatible" content="IE=edge">
6     <meta name="viewport" content="width=device-width, initial-scale=1.0">
7     <link rel="stylesheet" href="http://127.0.0.1:8000/css/app.css">
8     <title>SaintPosts</title>
9   </head>
10
11  <body>
12    <div class="container">
13      <header class="row">
14        <nav class="navbar navbar-expand-lg navbar-light bg-light">
15          <div class="container-fluid">
16            <a class="navbar-brand" href="http://127.0.0.1:8000/posts">Saint-Posts</a>
17            <button class="navbar-toggler" type="button" data-bs-toggle="collapse"
               data-bs-target="#navbarNav" aria-controls="navbarNav" aria-expanded="false"
               aria-label="Toggle navigation">
              <span class="navbar-toggler-icon"></span>
            </button>
            <div class="collapse navbar-collapse" id="navbarNav">

```

Ниже получим ответ (кнопки выше меняют вид)



Так мы можем создавать что-то в базе для тестов (body-row-json)

Json тут используется как ассоциативный массив

```
1 {  
2   ... "name": "boris",  
3   ... "age": 20  
4 }
```

Синтаксис

```
return [  
    'title' => 'string',  
    'content' => 'string',  
    'image' => 'string',  
    'category_id' => '',  
    'tags' => '',  
];
```

Наше поле ожидает вот такие ключи

НО РАБОАТЬ НИЧЕГО НЕ БУДЕТ ТК НУЖЕН csrf токен
ДЛЯ ЭТОГО МЫ ДОБАВИМ НУЖНЫЙ РОУТ В ИСКЛЮЧЕНИЯ

```
VerifyCsrfToken.php x StoreRequest.php web.php  
laravel > laravel > app > Http > Middleware > VerifyCsrfToken.php  
6  
7 class VerifyCsrfToken extends Middleware  
8 {  
9     /**  
10      * The URIs that should be excluded from CSRF verification.  
11      *  
12      * @var array<int, string>  
13      */  
14     protected $except = []  
15     '/posts/store'  
16     [];  
17 }
```

СЮДА (в except)

```

1 {
2   "title": "boris",
3   "content": "20",
4   "images": "2",
5   "status": 0,
6   "category_id": 6
7 }

```

```

1 {
2   "title": "artem",
3   "content": "123",
4   "images": "123",
5   "status": 1,
6   "category_id": 5
7 }

```

Запрос в постман

Постман помогает нам отслеживать ошибки, тестировать запросы и смотреть программную часть сайта

3.22 Restful API. Класс Resource в Laravel, асинхронный ответ с бека.

ВСЕ ЧТО БУДЕТ НИЖЕ ЭТО restful api

<https://aws.amazon.com/ru/what-is/restful-api/>

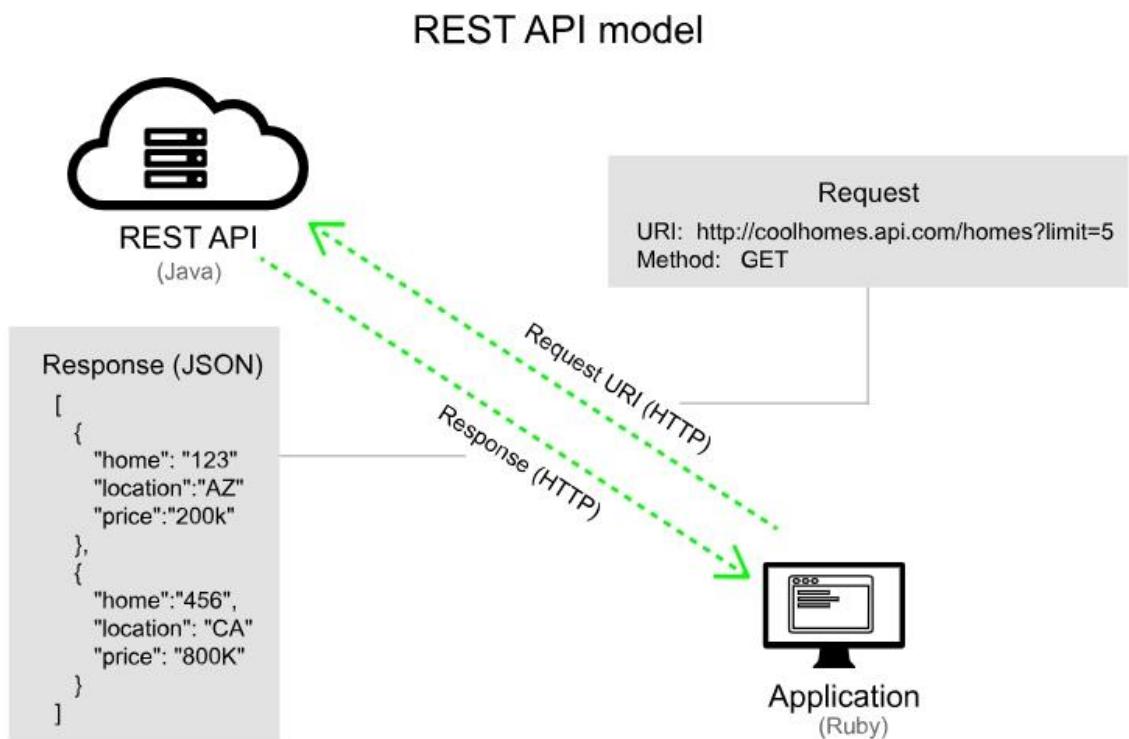
3.22.1 RESTful API

это интерфейс, используемый двумя компьютерными системами для безопасного обмена информацией через Интернет.

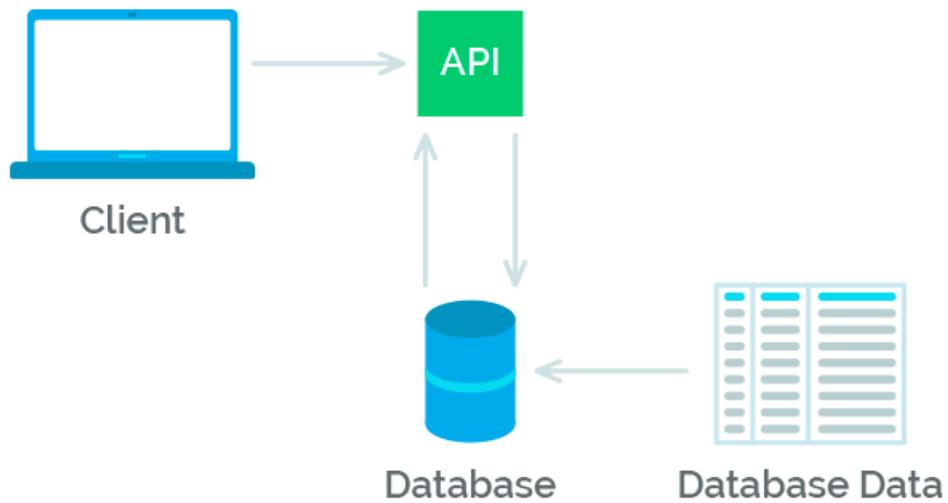
Representational State Transfer (REST) — это программная архитектура, которая определяет условия работы API. Первоначально REST создавалась как руководство для управления взаимодействиями в сложной сети, такой как Интернет.

Большинство бизнес-приложений должны взаимодействовать с другими внутренними и сторонними приложениями для выполнения различных задач. Например, чтобы генерировать ежемесячные платежные ведомости, ваша внутренняя бухгалтерская система должна обмениваться данными с банковской системой вашего клиента, чтобы автоматизировать выставление счетов и взаимодействовать с внутренним приложением по учету рабочего времени. RESTful API поддерживают такой обмен информацией, поскольку они следуют безопасным, надежным и эффективным стандартам программного взаимодействия.

Веб-службы, реализующие архитектуру REST, называются веб-службами RESTful. Как правило, термин RESTful API относится к сетевым RESTful API. Однако REST API и RESTful API являются взаимозаменяемыми терминами.



REST API Design



3.22.2 Принципы архитектурного стиля REST:

Единый интерфейс
отсутствие сохранения состояния
многоуровневой системной
кэширование
Код по запросу

REST API не зависит от используемой технологии. Вы можете создавать как клиентские, так и серверные приложения на разных языках программирования, не затрагивая структуру API. Также можно изменить базовую технологию на любой стороне, не влияя на обмен данными.

3.22.2 Как работает RESTful API?

Базовый принцип работы RESTful API совпадает с принципом работы в Интернете. Клиент связывается с сервером с помощью API, когда ему требуется какой-либо ресурс. Разработчики описывают принцип использования REST API клиентом в документации на API серверного приложения. Ниже представлены основные этапы запроса REST API:

1. Клиент отправляет запрос на сервер. Руководствуясь документацией API, клиент форматирует запрос таким образом, чтобы его понимал сервер.

2. Сервер аутентифицирует клиента и подтверждает, что клиент имеет право сделать этот запрос.
3. Сервер получает запрос и внутренне обрабатывает его.
4. Сервер возвращает ответ клиенту. Ответ содержит информацию, которая сообщает клиенту, был ли запрос успешным. Также запрос включает сведения, запрошенные клиентом.

3.22.3 HTTP-аутентификация

HTTP определяет некоторые схемы аутентификации, которые можно использовать при реализации REST API. Ниже представлены две такие схемы:

Базовая аутентификация

При базовой аутентификации клиент отправляет имя пользователя и пароль в заголовке запроса. Он кодирует их с помощью метода кодирования base64, который преобразует пару имя пользователя–пароль в набор из 64 символов для безопасной передачи.

Аутентификация носителя

Аутентификация носителя — это процесс предоставления управления доступом носителю токена. Как правило, токен носителя представляет собой зашифрованную строку символов, которую сервер генерирует в ответ на запрос входа в систему. Клиент отправляет токен в заголовках запроса для доступа к ресурсам.

3.22.4 OAuth

OAuth сочетает в себе пароли и токены для безопасного входа в любую систему. Сначала сервер запрашивает пароль, а затем дополнительный токен для завершения процесса авторизации. Он может проверять токен в любое время, а также через определенный период времени в соответствии с областью и сроком действия.

3.22.5 Асинхронный ответ в бека

Асинхронный ответ в бека на асинхронный запрос должен быть json формата

Асинхронный ответ с бэка должен быть в виде массивы json формата для удобной работы

Вот как это сделать:

Для этого в ларавел есть класс Resource

The screenshot shows a code editor with two files open:

App\Services\Post.php

```
namespace App\Services\Post;

use App\Models\Post;

class Service
{
    public function store($data)
    {
        $tags = $data['tags'];
        unset($data['tags']);

        $post = Post::create($data);

        $post->tags()->attach($tags);

        return $post;
    }
}
```

App\Http\Controllers\StoreController.php

```
class StoreController extends BaseController
{
    public function __invoke(StoreRequest $request)
    {
        $data = $request->validated();

        $post = $this->service->store($data);

        $arr = [
            'title' => $post->title,
            'content' => $post->content,
            'image' => $post->image,
        ];
        return $arr;
        return redirect()->route('post.index');
    }
}
```

A tooltip "СМ на ретурн" is visible near the bottom right of the editor.

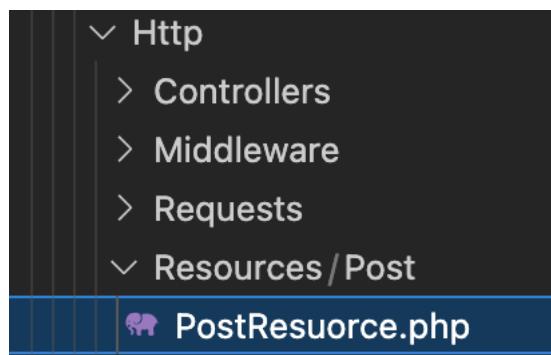
```
1 {  
2     "title": "Boris",  
3     "content": "Some content",  
4     "image": "blabla.jpeg"  
5 }
```

Вообще можно сделать все вот таким способом чтобы получить json ответ от бека, но в ларавел есть свой класс чтобы не писать все вот это

Resource – по сути просто сформированный массив данных которые потом преобраз в json контент

```
php artisan make:resource Post/PostResource
```

Создаем ресурс



Расположение

```
<?php

namespace App\Http\Resources\Post;

use Illuminate\Http\Resources\Json\JsonResource;

class PostResource extends JsonResource
{
    /**
     * Transform the resource into an array.
     *
     * @param \Illuminate\Http\Request $request
     * @return array|\Illuminate\Contracts\Support\Arrayable\JsonSerializable
     */
    public function toArray($request)
    {
        return parent::toArray($request);
    }
}
```

Содержание

Теперь создадим json ответ на асинхронный запрос с фронта при помощи ресурса(класса, который форматирует все json массив)

```
public function toArray($request)
{
    return [
        "title" => $this->title,
        "content" => $this->content,
    ];
}
```

Меняем в ресурсе (пост попадает в виде контекста (this = пост))

```
public function __invoke(StoreRequest $request){
    $data = $request -> validated();
    $post = $this->service->store($data);

    return new PostResource($post);
    // return redirect() -> route('post.index');
}
```

Меняем в сторконтроллере

По сути, мы просто сформировали ответ через ресурс

The screenshot shows the Postman interface. At the top, the URL is set to `http://127.0.0.1:8000/posts/store`. The method is selected as `POST`. The `Body` tab is active, showing the following JSON payload:

```

1 {
2   "title": "boris",
3   "content": "20",
4   "images": "2",
5   "status": 0,
6   "category_id": 6
7 }

```

Below the body, the response section shows the status `201 Created` with a response time of `74 ms` and a size of `1.14 KB`. The response body is displayed in `Pretty` format:

```

1 {
2   "data": {
3     "title": "boris",
4     "content": "20"
5   }
6 }

```

Отправляем асинхронный запрос и получаем ответ в виде json

Data – общепринятый стандарт при возвращении ресурса (ресурс возвращается с ключом data)

Это для 1 поста. А если их 10?

```
return PostResource::collection($posts);
```

В indexcontroller теперь не создаем новый класс ресурса, а обращаемся к его статистическому методу (если мы знаем, что ответ будет не один пост) (без new тк будет не один пост, а массив постов)

PostResource прежде всего просто php класс

http://127.0.0.1:8000/posts

```

1 ...
2 ... "title": "boris",
3 ... "content": "20",
4 ... "images": "?"

```

Body Cookies (2) Headers (9) Test Results

Pretty Raw Preview Visualize **JSON**

```

1 {
2   "data": [
3     {
4       "title": "delectus incidunt animi",
5       "content": "Qui quas repellendus maiores est. Tenetur nam sint error eligendi. Labore
6         recusandae placeat est."
7     },
8     {
9       "title": "ut consequatur officia",
10      "content": "Nobis id sint fuga ea. Commodo dolor perferendis non qui. Vero itaque aliquid non.
11        "
12     },
13     {
14       "title": "eveniet neque",
15       "content": "Reprehenderit facilis similique rerum. Repudiandae vitae perferendis porro."
16     }
17   ]
18 }

```

Обращаемся через гет и видим резу

3.22.6 ФИЛЬТРАЦИЯ:

```

1 ...
2 ... "title": "ut"
3 ...

```

Body Cookies (2) Headers (9) Test Results

Pretty Raw Preview Visualize **JSON**

```

1 {
2   "data": [
3     {
4       "title": "ut"
5     }
6   ]
7 }

```

В БОДИ -РОУ ФОРМИРУЕМ ФИЛЬТР

При подключении фильтра в этом поле можно фильтровать запросы

3.22.7 ПЕРЕЛИСТЫВАНИЕ СТРАНИЦ

The screenshot shows a code editor with a sidebar containing a file tree. The tree includes files like DestroyController.php, EditController.php, IndexController.php, ShowController.php, StoreController.php, UpdateController.php, AboutController.php, ContactController.php, Controller.php, HomeController.php, MainController.php, MyPlaceController.php, and PostController.php. Under the Requests folder, there is a Post folder which contains FilterRequest.php, StoreRequest.php, and UpdateRequest.php. The main editor area displays a PHP class with a rules() method:

```
23     */
24     public function rules()
25     {
26         return [
27             'title' => '',
28             'content' => '',
29             'category_id' => '',
30             'page' => '',
31             'per_page' => |
32         ];
33     }
34 }
35 }
36 }
```

Для реализации перелистывания фильтра на странице постмана нужно добавить в фильтр две св-ва(последние два)

The screenshot shows a code editor with a file named FilterRequest.php. The _invoke() method contains the following code:

```
public function _invoke(FilterRequest $request)
{
    $data = $request->validated();

    $page = $data['page'] ?? 1;
    $perPage = $data['per_page'] ?? 10;

    $filter = app()->make(abstractPostFilter::class, ['queryParams' => array_filter($data)]);
    $posts = Post::filter($filter)->paginate($perPage, ['*'], 'page', $page);

    return PostResource::collection($posts);
    //return view('post.index', compact('posts'));
}
```

Меняем индекс контроллер

GET http://127.0.0.1:8000/posts

Params Authorization Headers (9) **Body** Pre-request Script Test:

none form-data x-www-form-urlencoded raw binary G

```
1 {
2     "title": "aut",
3     "page": 2
4 }
```

Body Cookies (2) Headers (9) Test Results

Pretty Raw Preview Visualize JSON

```
1 {
2     "data": [
3         {
4             "id": 85,
5             "title": "Ad laudantium aut iusto alias unde.",
6             "content": "Tempore ullam eveniet enim aperiam exercitatio",
7             "image": "https://via.placeholder.com/640x480.png/0088aa?1"
8         },
9     ]
}
```

Теперь при запросе мы можем конкретно указать какую стр отобразить
Пока не надо см видео 36

Апдейт работает так же

```
return $post;
}

public function update($post, $data)
{
    $tags = $data['tags'];
    unset($data['tags']);

    $post->update($data);
    $post->tags()->sync($tags);
    $post = $post->fresh();
}
```

Меняем сервис (последняя строка) (принудительное обновление базы)

```
class UpdateController extends BaseController
{
    public function __invoke(UpdateRequest $request, Post $post)
    {
        $data = $request->validated();

        $post = $this->service->update($post, $data);

        return new PostResource($post);
    }
}
```

Контроллер

```
use Illuminate\Foundation\Http\Middleware\VerifyCsrfToken as Middleware;

class VerifyCsrfToken extends Middleware
{
    /**
     * The URIs that should be excluded from CSRF verification.
     *
     * @var array
     */
    protected $except = [
        '/posts',
        '/posts/*',
    ];
}
```

Убрали защиту

The screenshot shows the Postman interface. At the top, there are tabs for Home, Workspaces, API Network, and Explore. A search bar says 'Search Postman' and an 'Upgrade' button is visible. Below the header, a list of requests is shown, with the first one being a PATCH request to 'http://127.0.0.1:8000/posts/7'. The 'Body' tab is selected, showing a JSON payload with a single key-value pair: 'title': '112'. The response section shows a status of 'Status: 200 OK' with a time of '70 ms' and a size of '1.23 KB'. The response body is displayed in JSON format:

```

1
2   "data": {
3     "id": 7,
4     "title": "112",
5     "content": "Velit et voluptatem rem omnis. Sed ut sapiente omnis maiores. Et dolorem sit et dolor dolor."
6   }
7

```

3.23 JWT Token. Асинхронные роуты

composer require php-open-source-saver/jwt-auth

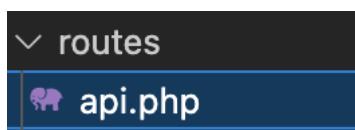
**ОБЯЗАТЕЛЬНО ПОЧИТАТЬ И ПОСОМТРЕТЬ ПРО jwt ТЕХНОЛОГИЯ
КРУТАЯ И ВАЖНАЯ**

На самом деле роуты в web.php они не асинхронные. Нам нужно выключать защиту csrf что создает серьёзные уязвимости в сайте

Web.php предназначен для интерфейса с веба (не асинхронный интерфейс)

На самом деле роуты нужно прописывать в api.php и действовать нужно через него

И не нужно выключать csrf

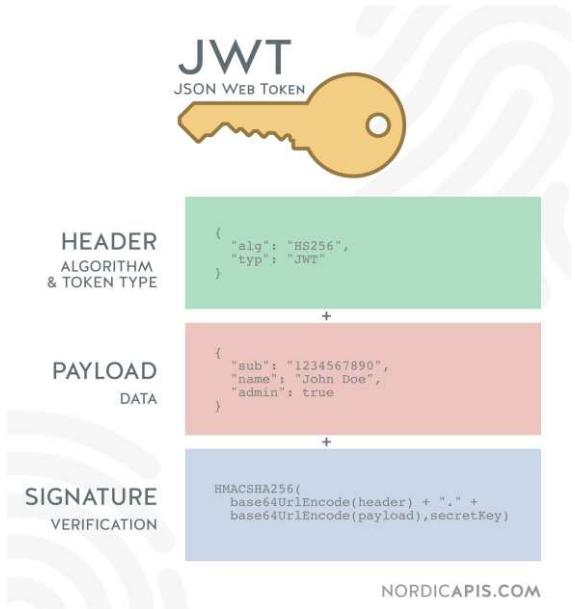


Тут

НО, ПРЕЖДЕ ЧЕМ ИСПОЛЬЗОВАТЬ ЭТИ РОУТЫ НУЖНО
ПОДКЛЮЧИТЬ jwt token (джот токен)

Юзер запрашивает токен посредством отправки на роут логина логин и пароль – в ответ получает jwt токен – вставляет – можно использовать роуты api.php

JSON Web Token — это открытый стандарт для создания токенов доступа, основанный на формате JSON. Как правило, используется для передачи данных для аутентификации в клиент-серверных приложениях.



Подключим токен

Нужно открыть папку vendor/tymon/Tymon\JWTAuth\Contracts\JWTSubject;

<https://jwt-auth.readthedocs.io/en/develop/laravel-installation/> - сайт

Тупо следуем указаниям которые там есть и все

composer require tymon/jwt-auth --ignore-platform-reqs (самая первая команда)

далее переходим сюда и делаем тут по списку

<https://jwt-auth.readthedocs.io/en/develop/quick-start/>

Тут мы первым делом подключаем интерфейсы, которые обязательны к применению

Для получения токена ПОД АККАУНТ НУЖНО АВТОРИЗИРОВАТЬСЯ

РЕГА = 1 АК = 1 ТОК

http://127.0.0.1:8000/api/auth/login?email=m@mail.ru&password=2021990qwe

Save |

POST http://127.0.0.1:8000/api/auth/login?email=m@mail.ru&password=2021990qwe Send

Params ● Authorization Headers (8) Body Pre-request Script Tests Settings Cookies

Query Params

KEY	VALUE	DESCRIPTION	...	Bulk Edit
<input checked="" type="checkbox"/> email	m@mail.ru			
<input checked="" type="checkbox"/> password	2021990qwe			
Key	Value	Description		

Наш запрос (внимание на .../api/auth/...)

Pretty Raw Preview Visualize JSON

```

1  {
2      "access_token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.
3          eyJpc3MiOiJodHRwOlwvXC8xMjcuMC4wLjE6ODAwMFwvYXBpXC9hdXRoXC9sb2dpbiIsIm1hdCI6MTYxODQ5NzI
4          g3MmRiN2E1OTc2ZjcfQ.NqofLdwDcatD4pQgh8cAtaU0feQ7TL_k6Ru8lDNtxkQ",
5      "token_type": "bearer",
6      "expires_in": 3600
7  }

```

Получаем токен

```
{
    "access_token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJodHRwOj8vMTI3LjAuMC4xOjgwMDAvYXBpL2F1dGgvbG9naW4iLCJpYXQiOjE2NjQxMjQ3MjEsImV4cCI6MTY2NDEyODMyMSwibmJmIjoxNjY0MTI0NzIxLCJqdGkiOiIwc3Fza1RINGRsT1hQWHNsIiwic3ViIjoiMiIsInBydiI6IjIzYmQ1Yzg5NDImNjAwYWRiMzllNzAxYzQwMDg3MmRiN2E1OTc2ZjcfQ.xhwYROj2ZASm1Je5jX5opmDoAKjHb4li5fSF19zDe4o",
    "token_type": "bearer",
    "expires_in": 3600
}
```

ЧТО ДЕЛАТЬ С ТОКЕНОМ?

```

'middleware' => 'api',
'prefix' => 'auth'

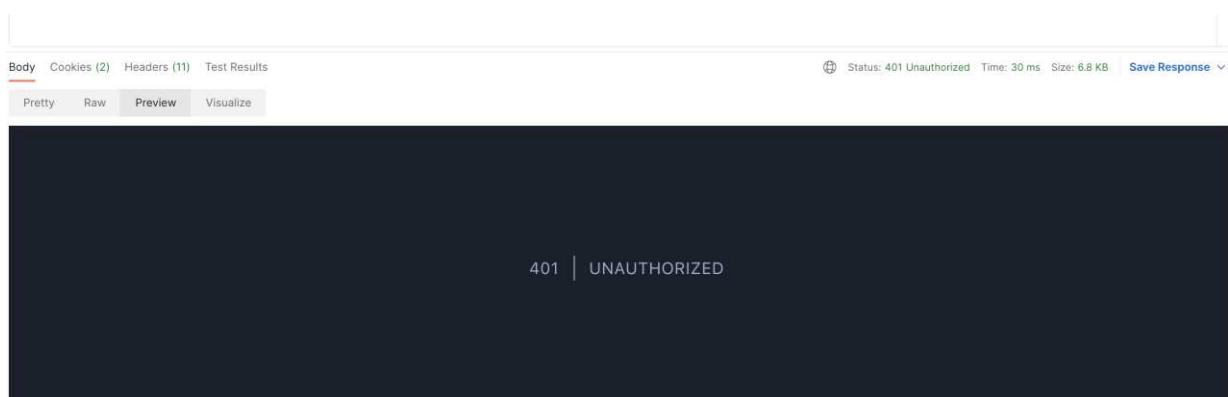
], function ($router) {
    Route::post('login', action:'AuthController@login');
    Route::post('logout', action:'AuthController@logout');
    Route::post('refresh', action:'AuthController@refresh');
    Route::post('me', action:'AuthController@me');

});

Route::group(['namespace' => 'Post', 'middleware' => 'jwt.auth'], function () {
    Route::get('posts', action:'IndexController');
});

```

Прописываем роут теперь не в веб а в апи след образом



(если не подключить посредника jwt выдаст ошибку авторизации)

Authorization tab selected: Bearer Token

Далее делаем вот такой запрос и Авторизуемся ПОЛУЧЕННЫМ ТОКЕНОМ

И никакой csrf токен отключать не надо, все работает как надо

См видео 38

3.24 CRUD с транзакцией

Транзакция (англ. transaction) — группа последовательных операций с базой данных, которая представляет собой логическую единицу работы с данными.

Транзакции

Транзакция – набор операций (изменений), который должен быть выполнен полностью или не выполнен совсем (единий логический блок).

Транзакции параллельны, если их выполнение пересекается во времени.

Механизм транзакций – основа обеспечения целостности БД.

Свойства транзакции (**ACID**):

- Atomicity – атомарность,
- Consistency – согласованность,
- Isolation – изолированность,
- Durability – устойчивость.

Понятие транзакции

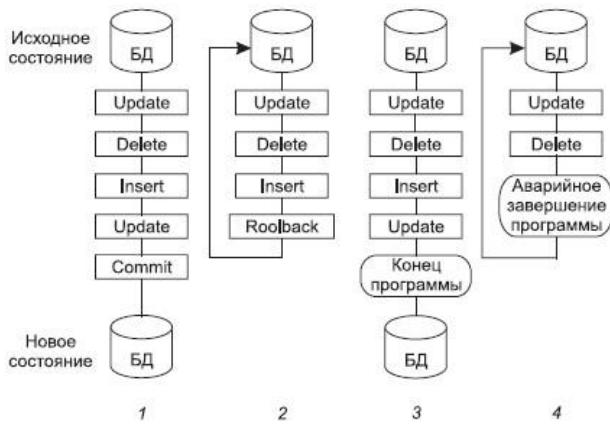
Транзакция – это последовательность операций, проводимых над БД, выполняемых как единое целое и переводящих БД из одного непротиворечивого состояния в другое непротиворечивое состояние

Количество операций, входящих в транзакцию, может быть любым от одной до сотен, тысяч

Разработчик решает, какие команды должны выполняться как одна транзакция, а какие могут быть разбиты на несколько последовательно выполняемых транзакций.

При выполнении транзакции СУБД должна обеспечить обработку набора команд, входящих в транзакцию, так, чтобы гарантировать правильность и надежность работы системы.

Транзакция должна удовлетворять ACID – требованиям



CRUD с транзакцией – создание тегов, постов, классов и т.д. в ОДНОМ ПОТОКЕ, это обеспечит защиту от ошибок и целостность БД. Избавление от разорванной связи

API.php – это роуты только для взаимодействия с вебом. Пользователи там не бывают

Часть 4. Laravel (продвинутый)

4.1 Новый проект – сайт для блогеров Скачиваем шаблон и создаем проект

В данной документации мы будем создавать Сайт-Блог с админпанелью и регистрацией пользователей с созданием ленты и подписок (мини соцсеть)

Шаблон скачиваем с сайта bootstrapdash

<https://www.bootstrapdash.com/product/edica>

Создаем проект и пушим в гит

composer create-project laravel/laravel LaravelBlog 9.1

4.2 Подключаем bootstrap, auth и базу данных

БЕЗ sqlite

На laravel 9 перед npm run dev установить mix - выполнить npm install laravel-mix --save-dev иначе не компилирует файлы, если используете mysql и не проходит миграции, проверьте и установите кодировку как в базе - config/database.php

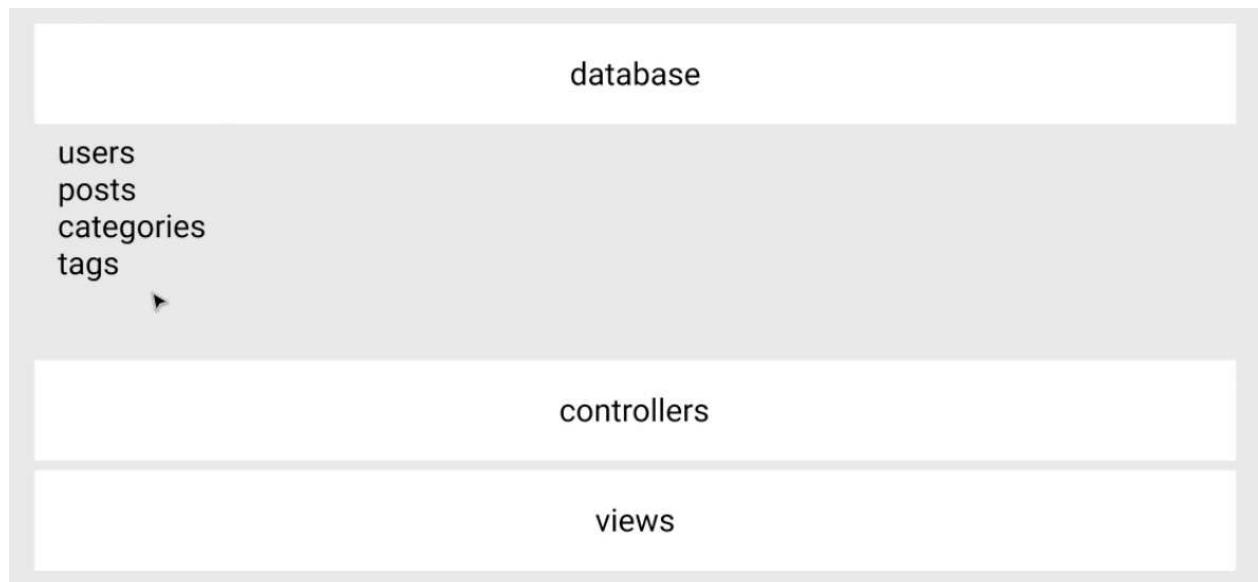
```
composer require laravel/ui  
php artisan ui bootstrap  
php artisan ui:auth  
npm install && npm run dev
```

В ЛАРАВЕЛ 9.19 появился vite вместо mix. НУЖНО УЧИТЬСЯ

4.3 Первоначальный план и создаем миграции

Для того чтобы грамотно составить программный продукт, нужно его правильно спроектировать.

Разобьём нашу задачу создать блог на подзадачи



Есть много вариаций, но обычно сначала разраб архитектура бд и все остальное натягивается на архитектуру

Сначала нужно создать первичную версию (рисунок -шаблон), который мы будем далее разрабатывать после оформления базиса

ПОСЛЕ МИГРАЦИИ ТАБЛИЦЫ ВСЕ ИЗМЕНЕНИЯ ДЕЛАТЬ ЧЕРЕЗ ДРУГИЕ ТАБЛИЦЫ (МИГРАЦИИ НА РЕДАКТИРОВАНИЕ)

1. Создаем модели КАТЕГОРИИ, ТЕГИ, ПОСТЫ, ПОСТЫ_ТЕГИ (с миграциями и таким же порядком)
2. Создаем отношения

```
public function up()
{
    Schema::create('categories', function (Blueprint $table) {
        $table->id();
        $table->timestamps();
        $table->string('title');
    });
}
```

```
public function up()
{
    Schema::create('tags', function (Blueprint $table) {
        $table->id();
        $table->timestamps();
        $table->string('title');
    });
}
```

```
Schema::create('posts', function (Blueprint $table) {
    $table->id();
    $table->string('title');
    $table->text('content');

    $table->unsignedBigInteger("category_id")->nullable();

    $table->timestamps();

    $table->index("category_id", "post_category_idx");
    $table->foreign("category_id", "post_category_fk")->on('categories')->references('i
});
```

```
public function up()
{
    Schema::create('post_tags', function (Blueprint $table) {
        $table->id();
        $table->unsignedBigInteger("post_id");
        $table->unsignedBigInteger("tag_id");
        $table->timestamps();

        $table->index("post_id", "post_tag_post_idx");
        $table->index("tag_id", "post_tag_tag_idx");

        $table->foreign("post_id", "post_tag_post_fk")->on('posts')->references('id');
        $table->foreign("tag_id", "post_tag_tag_fk")->on('tags')->references('id');
    });
}
```

3. Мигрируем

4.4 Подгружаем фронт страницы блога. Список постов

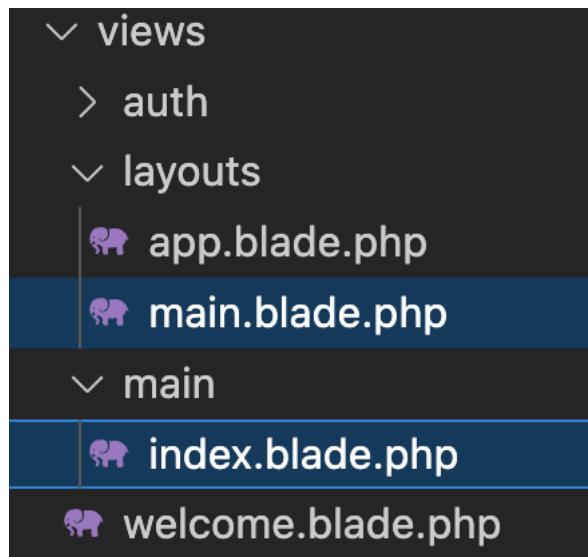
```
Route::group(['namespace' => 'App\Http\Controllers>Main'], function () {
    Route::get('/', 'IndexController')->name('index');
});

Auth::routes();
```

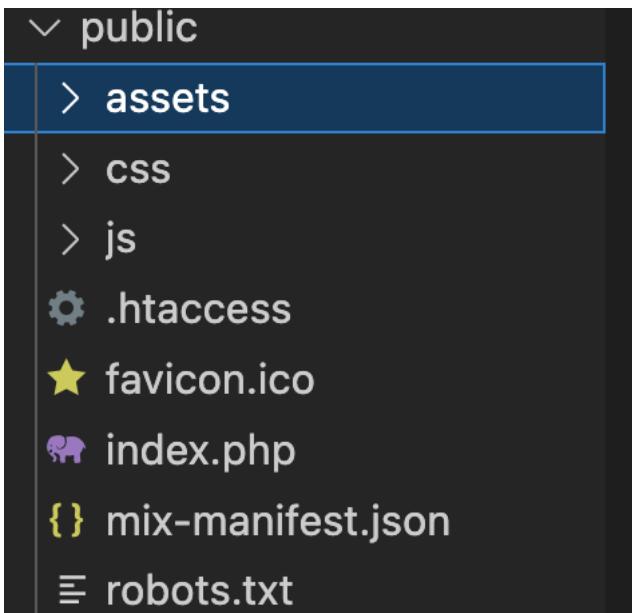
Строим роут главной страницы

```
class IndexController extends Controller
{
    public function __invoke()
    {
        return view('main.index');
    }
}
```

Контроллер главной страницы



Вью главной страницы с лэйаут



Добавляем в паблик ассет для сайта-блога

```
<head>
<!-- <base href="{{ asset('') }}" -->
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Edica :: Home</title>
    <link rel="stylesheet" href="{{asset('assets/vendors/flag-icon-css/css/flags.min.css')}}">
    <link rel="stylesheet" href="{{asset('assets/vendors/font-awesome/css/all.min.css')}}">
    <link rel="stylesheet" href="{{asset('assets/vendors/aos-aos.css')}}">
    <link rel="stylesheet" href="{{asset('assets/css/style.css')}}">
    <script src="{{asset('assets/vendors/jquery/jquery.min.js')}}">
```

Меняю ссылки ЛИБО РУКАМИ ЛИБО 1 СТР

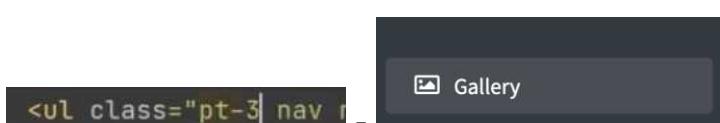
4.5 Ставим фронт админки, установка Admin Lte

Просто ставим админку

См гитхаб

4.6 Определяемся со стратегией и создаем задел для CRUD категорий

Задача: реализовать КРУД для постов, тегов и категорий в базе

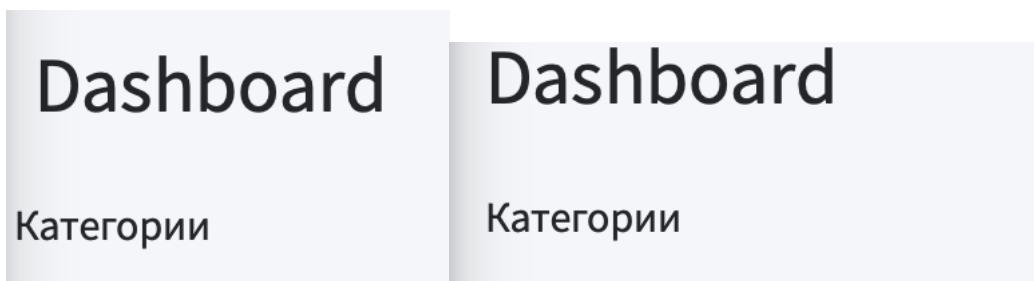


Стратегии разработки:

Делать фронт-бэк
Делать бэк-фронт
Делать блок фронт – блок бэк (пошагово)

```
<div class="row">
    <div class="col-12">
        <p>Категории</p>
    </div>
</div>
```

Col-12 выравнивает



		Категории	Теги	Посты
Index	READ			
Create	CREATE			
Store	CREATE			
Show	READ			
Edit	UPDATE			
Update	UPDATE			
Delete	DELETE			

Создадим вот такую штуку где будем помечать готовность

4.7 Страница создания категории(форма создания)

Роут-контроллер-вью
Роут-контроллер-реквест(валидатор)

```

class StoreController extends Controller
{
    public function __invoke(StoreRequest $request)
    {
        $data = $request->validated();
        dd($data);
    }
}

<form action="{{route('admin.category.store')}}" class="w-25" method="POST">
    @csrf
    <input name="title" type="text" class="form-control" placeholder="Название">
    <button type="submit" class="mt-3 btn btn-block btn-outline-success">Создать</button>
    <button type="reset" class="btn btn-block btn-outline-secondary">Сбросить</button>
</form>

```

Name такой же как имя в реквесте и в базе

```

array:1 [▼ // app/Http/Controllers/Admin/Category/StoreController.php:14
    "title" => "1"
]

```

```

class StoreController extends BaseController
{
    public function __invoke(StoreRequest $request)
    {
        $data = $request->validated();
        $this->service->store($data);
        return view('admin.categories.create');
    }
}

```

```

class Service{
    public function store($data){
        Category::create($data);
    }
}

```

ТАК ДЕЛАТЬ НЕЛЬЗЯ тк нарушается целостность БД, мы сможем создать много категорий с одним названием, а так делать НЕ КРУТО

```
class Service{
    public function store($data){
        Category::FirstOrCreate($data);
    }
}
```

НАДО ДЕЛАТЬ ТАК

```
public function store(StoreRequest $request)
{
    $data = $request->validated();
    Category::firstOrCreate(['title' => $data['title']], [
        'title' => $data['title']
    ]);
}
```

Или так

```
Route::get('/show/{category}', 'ShowController')->name('admin.category.show')
```

```
public function __invoke(Category $category)
```

```
<div class="col-sm-6 d-flex align-items-center">
    <h1 class="m-0 mr-2">$category->title</h1> <a href="{{route('category.show', $category)}}>
</div><!-- /.col -->
<div class="col-sm-6">
```

4.8 Добавляем атрибут content для поста, визуальный редактор Summernote

Добавляем папку summernote из админки в наши плагины

Подключаем соединение к плагину (для этого смотрим в файл админлтепагес-вормс-едитор)

```
<!-- summernote -->
<link rel="stylesheet" href="../../plugins/summernote/summernote-bs4.min.css">
```

```
plugins/summernote/summernote-bs4.min.css
```

```
<!-- Bootstrap 4 -->
<script src="../../plugins/bootstrap/js/bootstrap.bundle.min.js"></script>
```

<https://summernote.org/deep-dive/>

Тут можно посмотреть конфиги

```
array:2 [▼ // app/Http/Controllers/Admin/Post/StoreController.php:13
    "title" => "1"
    "content" => "<p>pdf</p>"
]
```

Что попадает

```
<div class="form-group w-25">
    <input id="title" name="title" type="text" class="form-control" placeholder="Название">
    @error('title')
        <p class="text-danger">Поле необходимо для заполнения</p>
    @enderror
</div>
<div class="form-group">
    <textarea id="summernote" name="content">{{old('content')}}</textarea>
    @error('content')
        <p class="text-danger">Поле необходимо для заполнения</p>
    @enderror
</div>
```

См ХЕЛПЕР олд

4.9 Добавляем атрибуты для загрузки изображений

Опять обращаемся к adminlte и берем оттуда форму загрузки изображения

Просто берем форму с файлами

Но далее нужно это все подключить (с помощью скрипта в файле админлте-пагес-вормс-генерал)

Также подключить плагин bs_custom_file

```
<!-- bs-custom-file-input -->
<script src="plugins/bs-custom-file-input/bs-custom-file-input.min.js"></script>
```

```
<script>
    $(function() {
        bsCustomFileInput.init();
    });
</script>
```

Далее делаем так чтобы форма могла подцепить файл

```
<form action="{{route('admin.post.store')}}" method="POST" enctype="multipart/form-data">
```

Чтобы картинки приходили как файл

```
public function rules() // правила
{
    return [
        "title" => "required|string",
        "content" => "required|string",
        "preview_image" => "required|file",
        "main_image" => "required|file"
    ];
}
```

Поменяли реквест

```
array:4 [▼ // app/Http/Controllers/Admin/Post/StoreController.php:13
  "title" => "123"
  "content" => "<p>123</p>"
  "preview_image" => Illuminate\UploadedImage {#344 ▼
    -test: false
    -originalName: "icons8-imac-50.png"
    -mimeType: "image/png"
    -error: 0
    #hashName: null
    path: "/private/var/folders/3k/slqjg8ln4w5gzg5q9cfqwn8r0000gn/T"
    filename: "phpRhsZUD"
    basename: "phpRhsZUD"
    pathname: "/private/var/folders/3k/slqjg8ln4w5gzg5q9cfqwn8r0000gn/T/phpRhsZUD"
    extension: ""
    realpath: "/privat.../T/phpRhsZUD"
    aTime: 2022-10-17 10:02:19
    mTime: 2022-10-17 10:02:19
    cTime: 2022-10-17 10:02:19
    inode: 3889073
    size: 410
    perms: 0100600
    owner: 501
    group: 20
    type: "file"
    writable: true
    readable: true
    executable: false
    file: true
    dir: false
    link: false
  }
  "main_image" => Illuminate\UploadedImage {#345 ▶}
]
```

Вот что приходит

В данном случае файл идет в локальное хранилище в папку storage, а в базе путь к этому файлу

```
AWS_ACCESS_KEY_ID=
AWS_SECRET_ACCESS_KEY=
AWS_DEFAULT_REGION=us-east-1
AWS_BUCKET=
AWS_USE_PATH_STYLE_ENDPOINT=false
```

Но также можно подключить и стороннее хранилище, а в базе будем хранить путь к файлу

Алгоритм действий:

- 1) Берем поступившие файлы
- 2) Помещаем в storage
- 3) Отдаем путь к файлам

```
"images/6r5Aj7073e1MMg1p8tItem9U8quF3hrfRc006dH22.jpg"
```

Путь

```
public function store($data){  
    $previewImage = $data['preview_image'];  
    $mainImage = $data['main_image'];  
    $previewImagePath = Storage::put('/images', $previewImage);  
    $mainImagePath = Storage::put('/images', $mainImage);  
    $post = Post::FirstOrCreate($data);  
    return $post;  
}
```

Как все работает (ТАК ДЕЛАТЬ НЕ НАДО)

```
class Service{  
    public function store($data){  
        $data['preview_image'] = Storage::put('/images', $data['preview_image']);  
        $data['main_image'] = Storage::put('/images', $data['main_image']);  
        $post = Post::FirstOrCreate($data);  
        return $post;  
    }  
}
```

А вот так надо (тут в дату поступает сразу путь) (переопределяем ключ)

Но будет ошибка тк нужно же и колонки добавить

```

public function up()
{
    Schema::table('posts', function (Blueprint $table) {
        $table->string('preview_image')->nullable();
        $table->string(['main_image'])->nullable();
    });
}

```

Добавляем миграцию (обязательно нуллабл для уже сущ таблиц)

```

public function down()
{
    Schema::table('posts', function (Blueprint $table) {
        $table->dropColumn('preview_image');
        $table->dropColumn(['main_image']);
    });
}

```

И обязательно дропнуть

```

public function down()
{
    Schema::table('posts', function (Blueprint $table) {
        $table->dropColumn('preview_image');
    });
    Schema::table('posts', function (Blueprint $table) {
        $table->dropColumn('main_image');
    });
}

```

Для sqlite

preview_image	main_image
NULL	NULL
images/PmjvqjIFySALOjK9P6pXusxp4Elu2ON9RjLR3MNC.jp...	images/gsK1SQvX00Jf9yG4iBViUHSn61APqsHZmslEnaV3.jp...

4.10 Добавляем возможность привязывать пост к категории

```
"category_id" => "required|exists:categories, id"
```

Как обычно создаем форму с категориями и в реквест это (проверяет наличие такого ключа в таблице)

Реквест менять не надо

```
@foreach($categories as $category)
<option value="{{$category->id}}"
    {{$category->id == old('category_id') ? 'selected' : ''}}>{{$category->title}}</option>
@endforeach
```

Валидация

4.11 Добавляем возможность привязывать тэги

Добавим плагин select2

```
<div class="form-group">
    <label>Добавить теги</label>
    <select name="tag_ids[]" class="select2" multiple="multiple" data-placeholder=">
        @foreach($tags as $tag)
            <option value="{{$tag->id}}>{{$tag->title}}</option>
        @endforeach
    </select>
</div>
```

```
return [
    "title" => "required|string",
    "content" => "required|string",
    "preview_image" => "required|file",
    "main_image" => "required|file",
    "category_id" => "required|integer|exists:categories,id",
    "tag_ids" => "nullable|array",
    "tag_ids.*" => "nullable|integer|exists:tags,id"
];
```

```

class Service{
    public function store($data){
        $tagIds = $data["tag_ids"];
        unset($data["tag_ids"]);

        $data['preview_image'] = Storage::put('/images', $data['preview_image']);
        $data['main_image'] = Storage::put('/images', $data['main_image']);
        $post = Post::FirstOrCreate($data);

        $post->tags()->attach($tagIds);
        return $post;
    }
}

```

```

class Post extends Model
{
    use HasFactory;
    use SoftDeletes;
    protected $table = "posts";
    protected $guarded = false;

    public function tags(){
        return $this->belongsToMany(Tag::class, 'post_tags', 'post_id', 'tag_id');
    }
}

```

```

class StoreController extends BaseController
{
    public function __invoke(StoreRequest $request)
    {
        try {
            $data = $request->validated();
            // dd($data);
            $this->service->store($data);
        } catch (\Exception $exception) {
            abort(404);
        }
        return redirect()->route('admin.post.index');
    }
}

```

Завернем в транзакцию

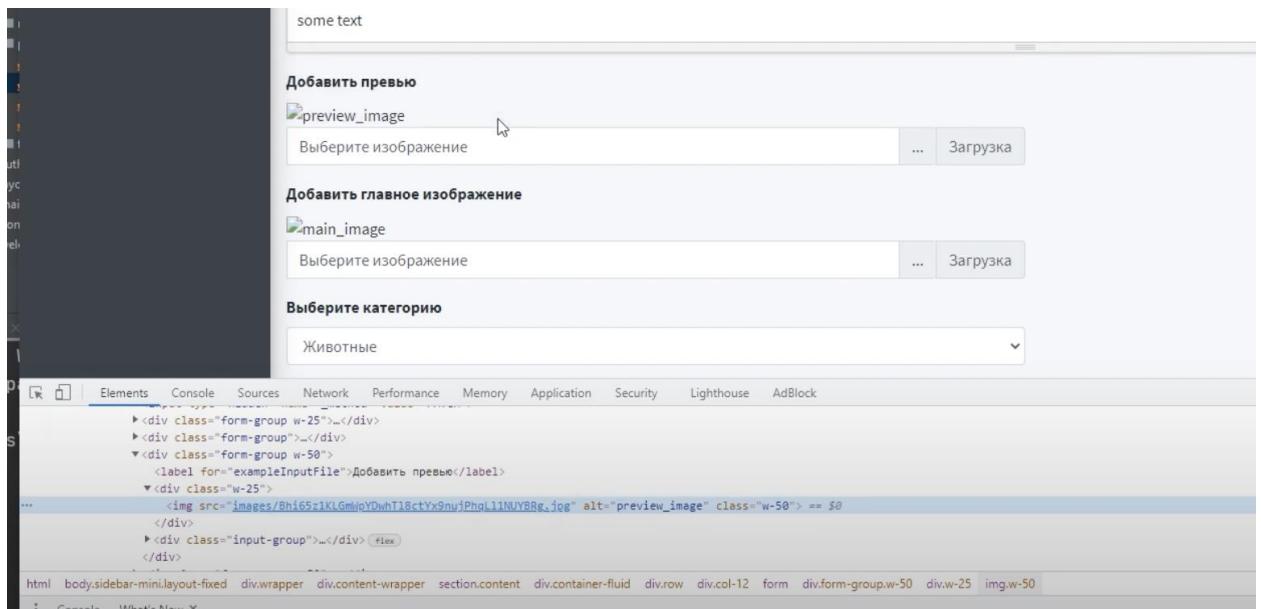
```

foreach($tags as $tag)
    <option {{ is_array( old('tag_ids') ) && in_array($tag->id, old('tag_ids')) ? 
    | 'selected' : '' }} value="{{ $tag->id}}">{{ $tag->title}}</option>

```

Валидация

4.12 Реализуем апдейт у постов, форма, отображение изображений



У картинки должен быть абсолютный путь(тут не такой) чтобы вне зависимости от того где сайт все отображалось нормально

Просто добавляем хелпер public_path(..)

```
<div class="w-50">
|   
</div>
```



4.12.1 Как отобразить картинку

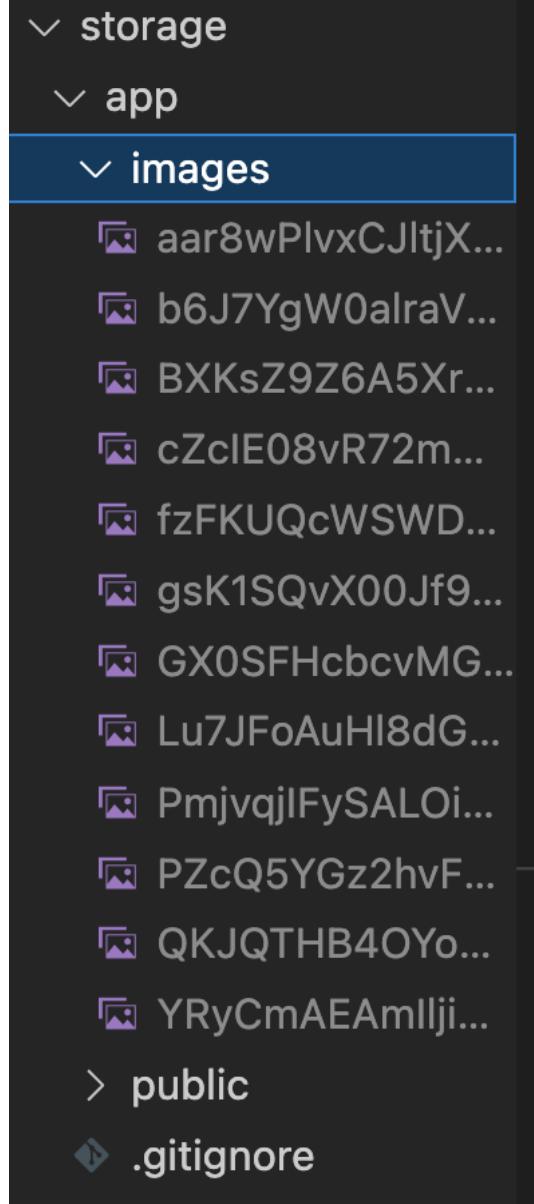
Далее чтобы картинки отображались нужно прописать php artisan storage:link

```
D:\lessons\blog>php artisan storage:link
The [D:\lessons\blog\public\storage] link has been connected to [D:\lessons\blog\storage\app\public].
The links have been created.
```

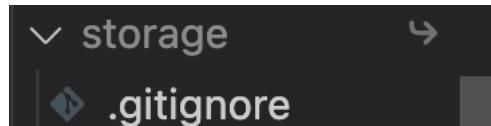
Так мы создали СИМВОЛИЧНУЮ ССЫЛКУ, это означает что все картинки сидят в стораге/апп/имг, НО у нас есть папка паблик

Когда мы прописываем эту команду у нас получ символичная ссылка в public

Символичная ссылка по сути – ярлык на рабочем столе



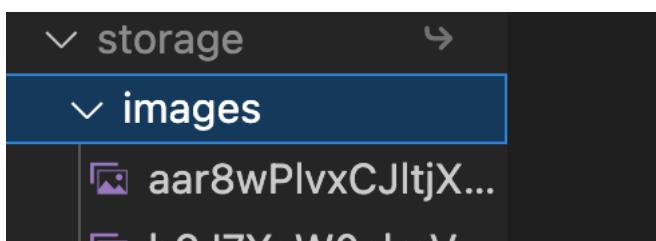
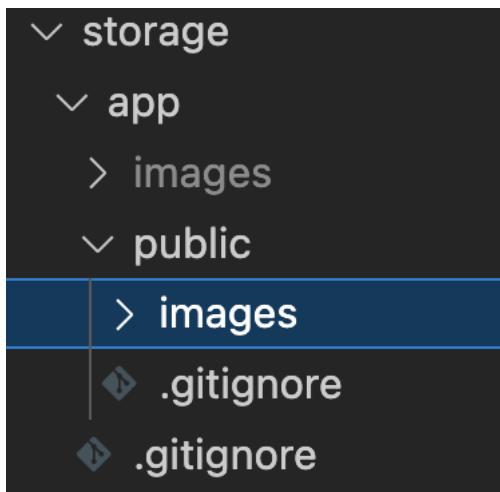
Папка



Символическая ссылка на эту папку в папке паблик

Таким образом мы открыли доступ для получения картинок

Далее переносим папку images внутрь папки паблик



Теперь папка появилась и в ссылке

Это не копия, а точка входа

```
<div class="w-50">
    
</div>
```

Далее прописываем точку входа с помощью хелпера asset (либо url(...))

Хелпер asset – это путь к папке паблик, сама папка в пути отображаться не будет

```
$data = $request->validated();
$tagIds = $data['tag_ids'];
unset($data['tag_ids']);

$data['preview_image'] = Storage::disk('public')->put('path: /images', $data['preview_image']);
$data['main_image'] = Storage::put('path: /images', $data['main_image']);
$post = Post::firstOrCreate($data);
$post->tags()->attach($tagIds);
} catch (Exception $exception) {
    abort(404);
}

return redirect('/admin/post/' . $post->id);
```

Теперь указываем куда будет попадать картинка при загрузке, чтобы фотка сразу попадала в паблик

```

<div class="form-group">
    <label>Добавить теги</label>
    <select name="tag_ids[]" class="select2" multiple="multiple" data-placeholder="Выберите теги..." style="width: 100%;"
        @foreach($tags as $tag)
            <option {{ is_array( array_column([], 'id') ) $post->tags && in_array($tag->id, $post->tags) ? 'selected' : '' }} value="{{ $tag->id}}">{{ $tag->title}}</option>
        @endforeach
    </select>
</div>

```

Так можно выцепить теги, НО У НАС ПРИХОДИТ КОЛЛЕКЦИЯ(аналог в ларавел который по сути массив, но нам для работы нужен обычный массив), а нужен нам массив

Такой способ преобразования называется ЧЕЙНИТЬ

Осталось все это запаковать и отправить на обновление

```

public function update($data, $post){
    $tagIds = $data["tag_ids"];
    unset($data["tag_ids"]);

    $data['preview_image'] = Storage::disk('public') -> put('/images', $data['preview_image']);
    $data['main_image'] = Storage::disk('public') -> put('/images', $data['main_image']);

    $post->update($data);
    $post->tags()->sync($tagIds); // удаляет все привязки с тегами
    return $post;
}

```

Так выглядит сервис обновления

Автору спасибо! В edit формах поменяли old('title') на \$post->title, но здесь мы теряем функционал old. В мануале прочитал, можно сохранить функционал old и данные вывести с модели вначале, записав их вторым параметром old('title', \$post->title)

```

public function update($data, $post){
    $tagIds = $data["tag_ids"];
    unset($data["tag_ids"]);

    // dd($data);
    if( array_key_exists('preview_image',$data)){
        $data['preview_image'] = Storage::disk('public')->put('/images', $data['preview_image']);
    }
    if( array_key_exists('main_image',$data)){
        $data['main_image'] = Storage::disk('public')->put('/images', $data['main_image']);
    }

    $post->update($data);
    $post->tags()->sync($tagIds); // удаляет все привязки с тегами
    return $post;
}

```

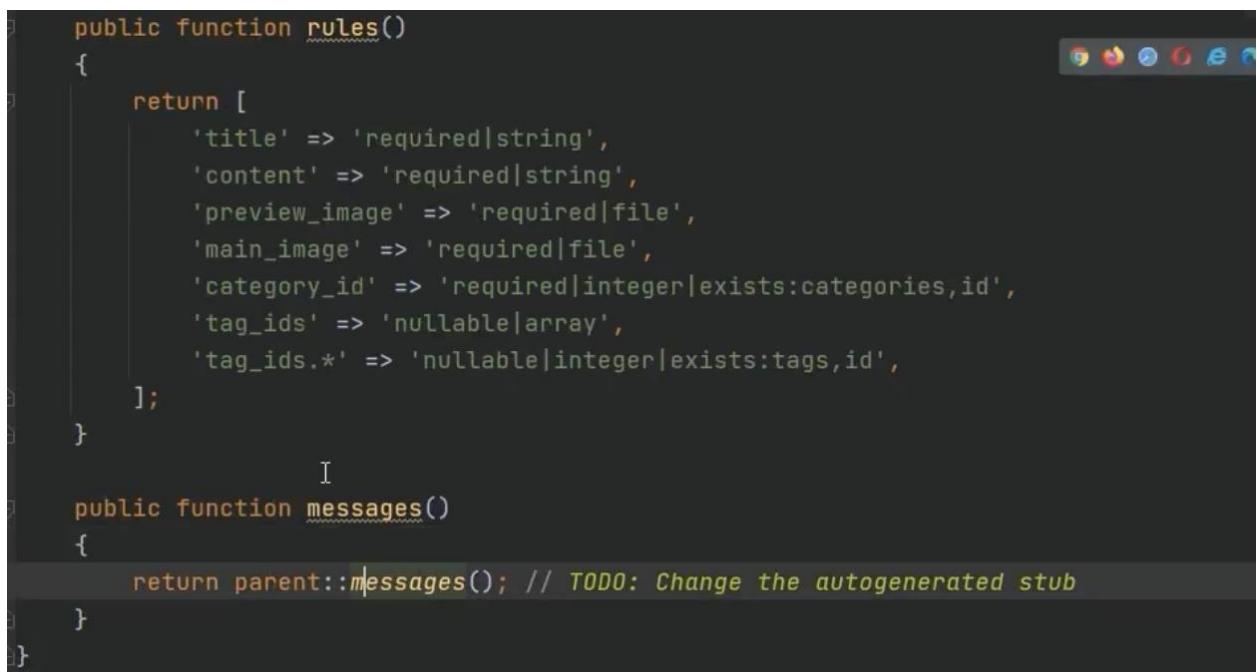
Тут добавили if тк вылезет ошибка

```
class="col-12">
<form action="{{route('admin.post.update', $post->id)}}" method="POST" enctype="multipart/form-data">
@csrf
@method('patch')
```

НЕ ЗАБЫВАТЬ ДОБАВЛЯТЬ enctype

4.13 Изменяем сообщения об ошибке в классе Request

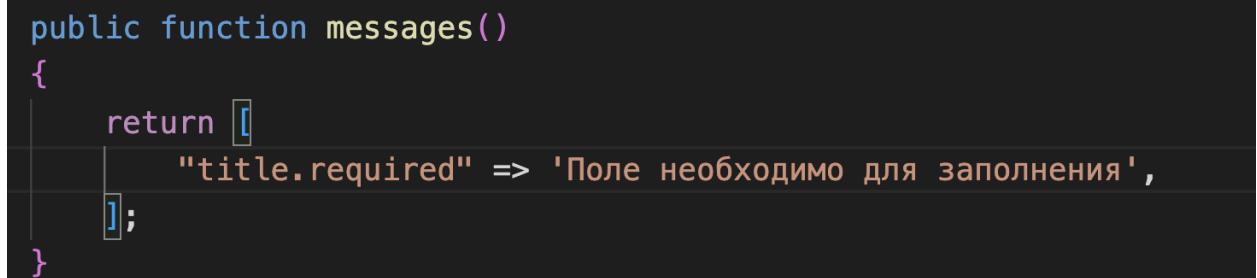
Реквест это зарезервированный класс у которого есть зарезервированные методы



```
public function rules()
{
    return [
        'title' => 'required|string',
        'content' => 'required|string',
        'preview_image' => 'required|file',
        'main_image' => 'required|file',
        'category_id' => 'required|integer|exists:categories,id',
        'tag_ids' => 'nullable|array',
        'tag_ids.*' => 'nullable|integer|exists:tags,id'
    ];
}

public function messages()
{
    return parent::messages(); // TODO: Change the autogenerated stub
}
```

Например messages()



```
public function messages()
{
    return [
        "title.required" => 'Поле необходимо для заполнения',
    ];
}
```

Так мы можем прописать сообщения об ошибках .required означает что если будет ошибка заполнения

```
public function messages()
{
    return [
        "title.required" => 'Поле необходимо для заполнения',
        "content.required" => 'Поле необходимо для заполнения',
        "preview_image.required" => 'Поле необходимо для заполнения',
        "preview_image.file" => 'Необходимо выбрать файл',
        "main_image.required" => 'Поле необходимо для заполнения',
        "main_image.file" => 'Необходимо выбрать файл',
        "category_id.required" => 'Поле необходимо для заполнения',
        "category_id.integer" => 'Неправильный формат категории',
        "category_id.exists" => 'Категории нет в базе',
        "tag_ids.required" => 'Поле необходимо для заполнения',
        "tag_ids.array" => 'Должен быть хотябы 1 тег',
    ];
}
```

```
@error('tag_ids')
<p class="text-danger">{{$message}}</p>
enderror
```

Так выводим сообщения

НО ДАЛЕЕ У НАС БУДЕТ ОШИБКА ЕСЛИ ОСТАВИТЬ ТЕГИ ПУСТЫМИ

```
public function store($data)
{
    try {
        Db::beginTransaction();
        if (isset($data['tag_ids'])) {
            $tagIds = $data['tag_ids'];
            unset($data['tag_ids']);
        }
    }
```

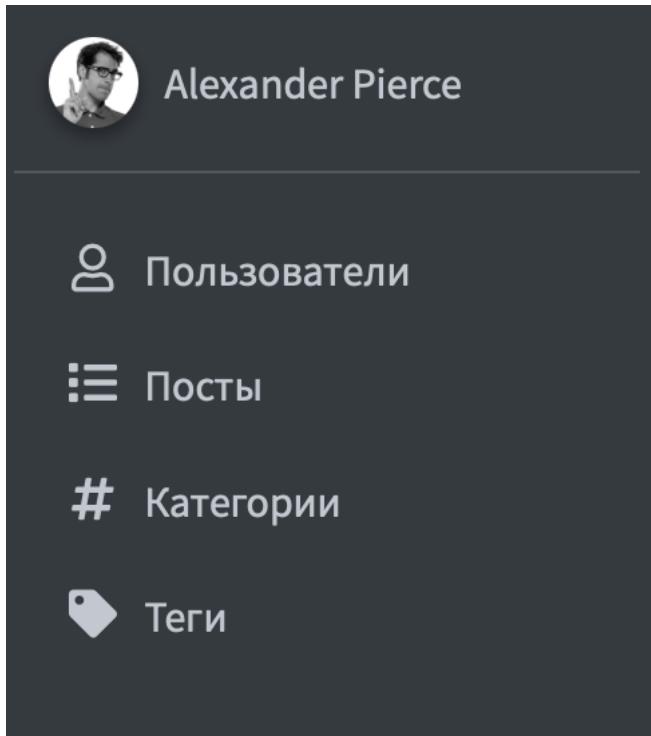
Нужно сделать проверку а есть ли теги вообще

```
if (isset($tagIds)) {
    $post->tags()->attach($tagIds);
}
DB::commit();
```

И тут тоже

4.14 Полностью добавляем основу CRUDa для пользователей

Будем добавлять возможность управления пользователями, удаление, добавление, бан, расширение групп полномочий и тд и тп



```
<li class="nav-item">
    <a href="{{route('admin.post.index')}}" class="nav-link">
        <i class="nav-icon far fa-thin fa-user"></i>
        <p>
            Пользователи
        </p>
    </a>
</li>
```

Создадим колонку пользователи

Далее полностью готовим шаблон crud для users такой же как categories

ШАГ 2

Тк у нас уже есть таблица пользователей от ларавел, создавать нам ничего не надо

users	
id	integer (auto increment)
name	varchar
email	varchar
email_verified_at	datetime
password	varchar
remember_token	varchar
created_at	datetime
updated_at	datetime
deleted_at	datetime
role	integer = 1
key #1 (id)	
users_email_unique (email)	UNIQUE

Создадим соответствующие элементы на странице при создании

Блэйд – реквесты

```
return [
    "name" => "required|string",
    "email" => "required|string|email",
    "password" => "required|string",
];
```

Store

```
{
    return [
        "name" => "required|string",
        "email" => "required|string|email",
    ];
}
```

Update

```
array:3 [▼ // app/Http/Controllers
    "name" => "231"
    "email" => "1@mail.ru"
    "password" => "1212"
]
```

При дд

!Важно учитывать что при регистрации и вводе пароля пароль обрабатывается специальным классом **hash**

```
class StoreController extends BaseController
{
    public function __invoke(StoreRequest $request)
    {
        $data = $request->validated();
        $data['password'] = Hash::make($data['password']);
        $this->service->store($data);
        return redirect()->route('admin.user.index');
    }
}
```

Тут мы с помощью метода make ЗАСЕКРЕТИЛИ ПАРОЛЬ чтобы потом можно было войти (мы переназначили дату)

```
class Service
{
    public function store($data)
    {
        $user = User::FirstOrCreate(["email" => $data['email']], $data);
        return $user;
    }
}
```

Далее в сервисе мы должны указать по какому признаку мы делаем создание (по уникальному ключу)

Теперь пользователи будут добавляться

Пользователи		
Создать		
ID	Имя	Действия
1	231	 

Редактирование

231

1@mail.ru

Изменить

Далее реализуем редактирование

Удаление уже реализовано, но без софтаделита, подключим его

```
• saint@MacBook-Pro-Artem LaravelBlog % php artisan make:migration add_column_softDeletes_to_users_table
```

```
INFO  Created migration [2022_10_19_164022_add_column_softDeletes_to_users_table].
```

```
public function up()
{
    Schema::table('users', function (Blueprint $table) {
        $table->softDeletes();
    });
}

/**
 * Reverse the migrations.
 *
 * @return void
 */
public function down()
{
    Schema::table('users', function (Blueprint $table) {
        $table->dropSoftDeletes();
    });
}
```

```
class User extends Authenticatable
{
    use HasApiTokens, HasFactory, Notifiable, SoftDeletes;
```

```
protected $fillable = [
    'name',
    'email',
    'password',
];
```

```
▶ saint@MacBook-Pro-Artem LaravelBlog % php artisan migrate
  INFO  Running migrations.
    2022_10_19_164022_add_column_softdeletes_to_users_table ..... 59ms DONE
```

Не забыть про ролбэк

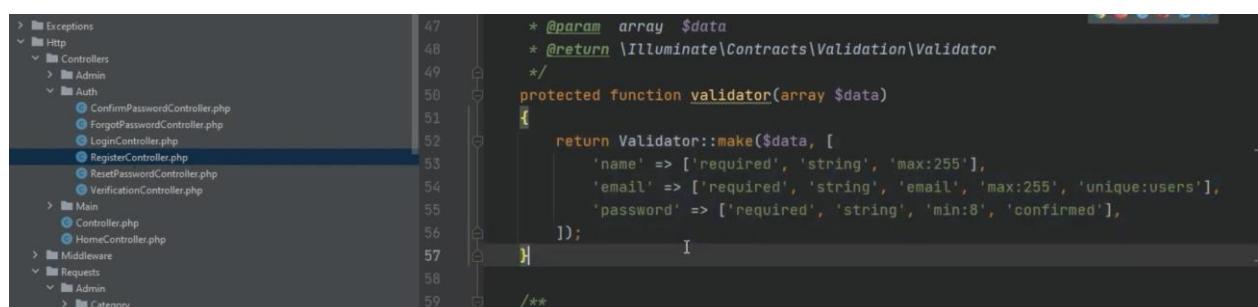
На данный момент crud готов, Но если у нас в базе уже будет существовать почта, и мы захотим зарегать ее еще раз, то будет ошибка

```
Illuminate\Database\QueryException
SQLSTATE[23000]: Integrity constraint violation: 19 UNIQUE constraint failed: users.email (SQL: update "users" set "email" = john@mail.ru, "updated_at" = 2021-08-25 18:56:32 where "id" = 3)
```

<http://127.0.0.1:8000/admin/users/3>

А ошибки нам не надо показывать обычному пользователю

Эту ошибку мы должны отвалидировать



```
    * @param array $data
    * @return \Illuminate\Contracts\Validation\Validator
    */
protected function validator(array $data)
{
    return Validator::make($data, [
        'name' => ['required', 'string', 'max:255'],
        'email' => ['required', 'string', 'email', 'max:255', 'unique:users'],
        'password' => ['required', 'string', 'min:8', 'confirmed'],
    ]);
}
```

Вот такие правила могут быть

```
public function rules() // правила
{
    return [
        "name" => "required|string",
        "email" => "required|string|email|unique:users",
        "password" => "required|string",
    ];
}
```

Валидируем на стор и апдейт

```
public function messages()
{
    return [
        "email.unique" => "Данная почта уже существует"
    ];
}
```

```
<input id="email" name="email" type="mail" class="form-control mb-2" placeholder="Email" />
@error('email')
<p class="text-danger">{{$message}}</p>
@enderror
```

Добавление пользователя

The screenshot shows a user registration form. The first input field contains the number '12'. The second input field contains the email '1@mail.ru'. Below the inputs, a red error message 'Данная почта уже существует' (This email already exists) is displayed. A third input field is labeled 'Пароль' (Password).

```
return [
    'name.required' => 'Это поле необходимо для заполнения',
    'name.string' => 'Имя должно быть строкой',
    'email.required' => 'Это поле необходимо для заполнения',
    'email.string' => 'Почта должна быть строкой',
    'email.email' => 'Ваша почта должна соответствовать формату [mail@some.domain]',
    'email.unique' => 'Пользователь с таким email уже существует',
];
```

4.15 Внедряем роли для пользователей и улучшаем валидацию для апдейта

```
:\lessons\blog>php artisan make:migration add_column_role_to_users_table
```

Создадим колонку для ролей

```

public function up()
{
    Schema::table('users', function (Blueprint $table) {
        $table->unsignedSmallInteger('role')->nullable();
    });
}

```

Добавили колонку ИНДЕКС и нуллабл (нуллабл обязательно тк это добавление)

Такой тип данных мы делаем для имитации атрибута forinkey, если бы у нас была отдельная таблица с ролями. Тк у нас имитация – то это УСЛОВНЫЕ АЙДИ. Сделали мы тк потому что требования позволяют, у нас не 10000 ролей

Не забываем дропнуть

```

use ...;

class User extends Authenticatable
{
    use HasFactory, Notifiable, SoftDeletes;

    const ROLE_ADMIN = 0;
    const ROLE_READER = 1;
}

```

Создаем условные константы, и теперь когда мы привяжем роль к пользователю в таблице у нас вместо слов будут цифры

	remember_token	created_at	updated_at	deleted_at	role
1	xygW0xl2N.7BvI...	<null>	2021-08-25 18:52:50	2021-08-25 18:52:53	2021-08-25 18:52:53
2	DufW.0zQi1l.5i...	<null>	2021-08-25 18:55:52	2021-08-25 18:55:52	<null>
3	lBPa.bU1Yjd/TC...	<null>	2021-08-25 18:56:09	2021-08-25 19:01:55	<null>

И если мы захотим расширить функционал и создать внешнюю таблицу то будем ссылаться на таблицу users используя айди и forinkey, то цифры ролей останутся и все пользователи которые были останутся со своими ролями, нам не нужно будет ничего переписывать , убирать админ и ридер (текстовые слова) наши роли автоматом ПЕРЕПРИВЯЖУТСЯ

ТАК мы не создавали новую таблицу, но сделали так что можно теперь дополнить в случае чего

```
public static function getRoles()
{
    return [
        ];
}
```

Далее создаем статический метод который возвращает массив, в котором будет расшифровка наших 0 1

ЭТОТ ПРИМЕР НАЗЫВАЕТСЯ **МЕПИНГ**(соответствие)

```
public static function getRoles()
{
    return [
        self::ROLE_ADMIN => 'Админ',
        self::ROLE[ADMIN] => 'Админ',
    ];
}
```

Так на фронт нам придут не 0 1 а админ и читатель

```
<?php

namespace App\Http\Controllers\Admin\User;

use App\Http\Controllers\Controller;
use App\Models\User;

class CreateController extends Controller
{
    public function __invoke()
    {
        $roles = User::getRoles();
        return view('admin.user.create', compact('var_name: 'roles'));
    }
}
```

Реализовываем взятии роли в контроллере и передачу в блэйд

```

self::ROLE_ADMIN => 'Админ',
self::ROLE_READER => 'Читатель',
0 => 'Админ',
1=> 'Читатель',

```

Придет вот такой массив

```

<div class="form-group w-50">
    <label>Выберите роль</label>
    <select name="role" class="form-control">
        @foreach($roles as $id => $role)
            <option value="{{ $id }}"
                   {{ $id == old('role_id') ? ' selected' : '' }}>{{ $role }}</option>
        @endforeach
    </select>

    @error('role')
        <div class="text-danger">{{ $message }}</div>
    @enderror
</div>

```

Реализовываем селект с выбором ролей для админки

Тут мы в форич пробегаем по пришедшему массиву как ключ(0 или 1) – значение(админ читатель)

Далее переносим все это дело в едит

```

<div class="form-group w-50">
    <label>Выберите роль</label>
    <select name="role" class="form-control">
        @foreach($roles as $id => $role)
            <option value="{{ $id }}"
                   {{ $id == $user->role ? ' selected' : '' }}>{{ $role }}</option>
        @endforeach
    </select>

    @error('role')
        <div class="text-danger">{{ $message }}</div>
    @enderror
</div>

```

```
public function rules()
{
    return [
        'name' => 'required|string',
        'email' => 'required|string|email|unique:users',
        'password' => 'required|string',
        'role' => 'required|integer'
    ];
}
```

Добавляем в реквесты роль

```
protected $fillable = [
    'name',
    'email',
    'password',
    'role',
];
```

Разрешаем редактирование роли

!!!!!!!!!!!!!!

Редактирование пользователя

The screenshot shows a user edit form with the following fields and errors:

- Name: guest (empty)
- Email: guest@mail.ru (exists)
- Role: Читатель (selected)

A red error message "Пользователь с таким email уже существует" (User with such email already exists) is displayed below the email field. A blue "Обновить" (Update) button is at the bottom.

Пофиксить уникальность майла при редактировании

Для этого нужно в едит реализовать игнорирование почты посредством скрытого ввода

```
<div class="form-group">
    <input type="hidden" name="user_id" value="{{ $user->id4 }}">
</div>
```

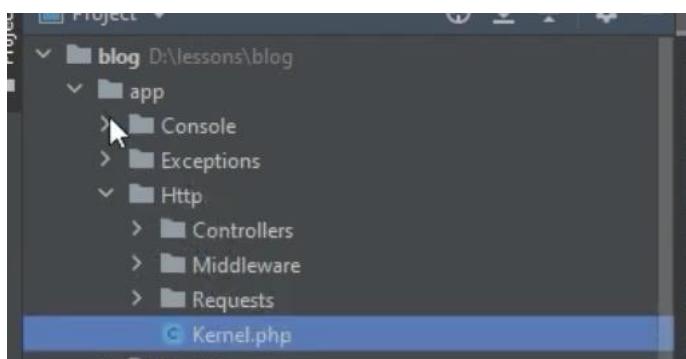
```
return [
    "name" => "required|string",
    "email" => "required|string|email|unique:users,email," . $this->user_id,
    "role" => "required|integer",
    "user_id" => "required|integer|exists:users,id",
];
```

Меняем в реквест и добавляем игнорирование

4.16 Создаем Middleware для админки и auth middleware, делаем кнопку выхода

```
D:\Lessons\blog>php artisan make:middleware AdminMiddleware
Middleware created successfully.
```

После создания посредника нужно обязательно его зарегать(придумать спец имя для использования в роутах)



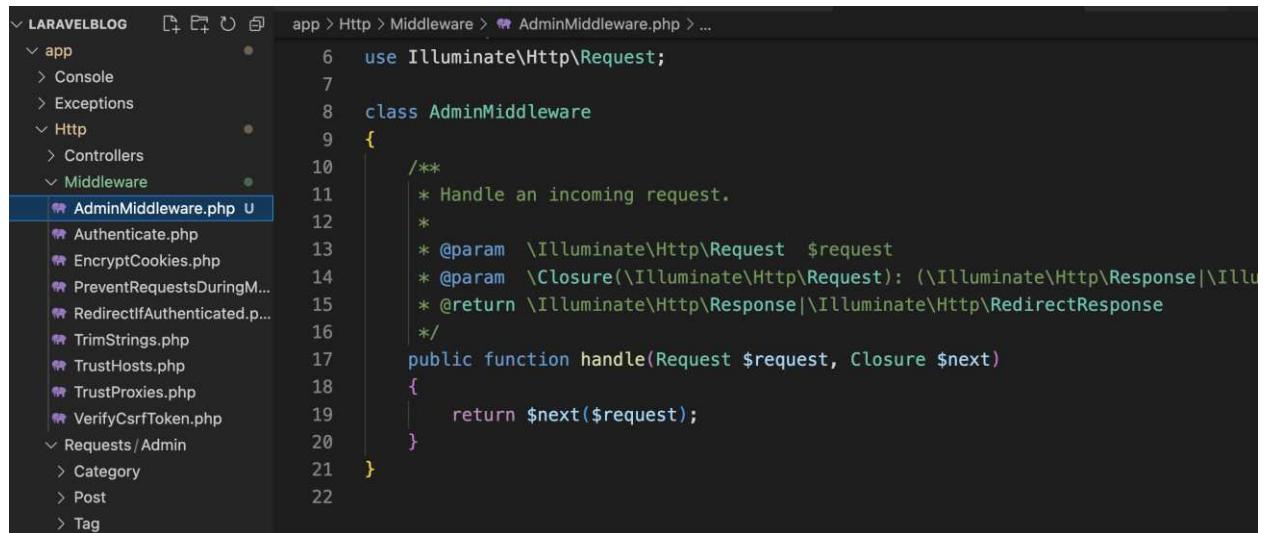
Переходим сюда

```
protected $routeMiddleware = [
    'auth' => \App\Http\Middleware\Authenticate::class,
    'auth.basic' => \Illuminate\Auth\Middleware\AuthenticateWithBasicAuth::class,
    'cache.headers' => \Illuminate\Http\Middleware\SetCacheHeaders::class,
    'can' => \Illuminate\Auth\Middleware\Authorize::class,
    'guest' => \App\Http\Middleware\RedirectIfAuthenticated::class,
    'password.confirm' => \Illuminate\Auth\Middleware\RequirePassword::class,
    'signed' => \Illuminate\Routing\Middleware\ValidateSignature::class,
    'throttle' => \Illuminate\Routing\Middleware\ThrottleRequests::class,
    'verified' => \Illuminate\Auth\Middleware\EnsureEmailIsVerified::class,
    'admin' => AdminMiddleware::class,
];
```

Задаем имя посреднику

```
Route::group(['namespace' => 'Admin', 'prefix' => 'admin', 'middleware' => 'admin'], function() {
    Route::group(['namespace' => 'Main'], function () {
        Route::get('uri: /', action: 'IndexController');
```

Задаем посредника роутам с админкой

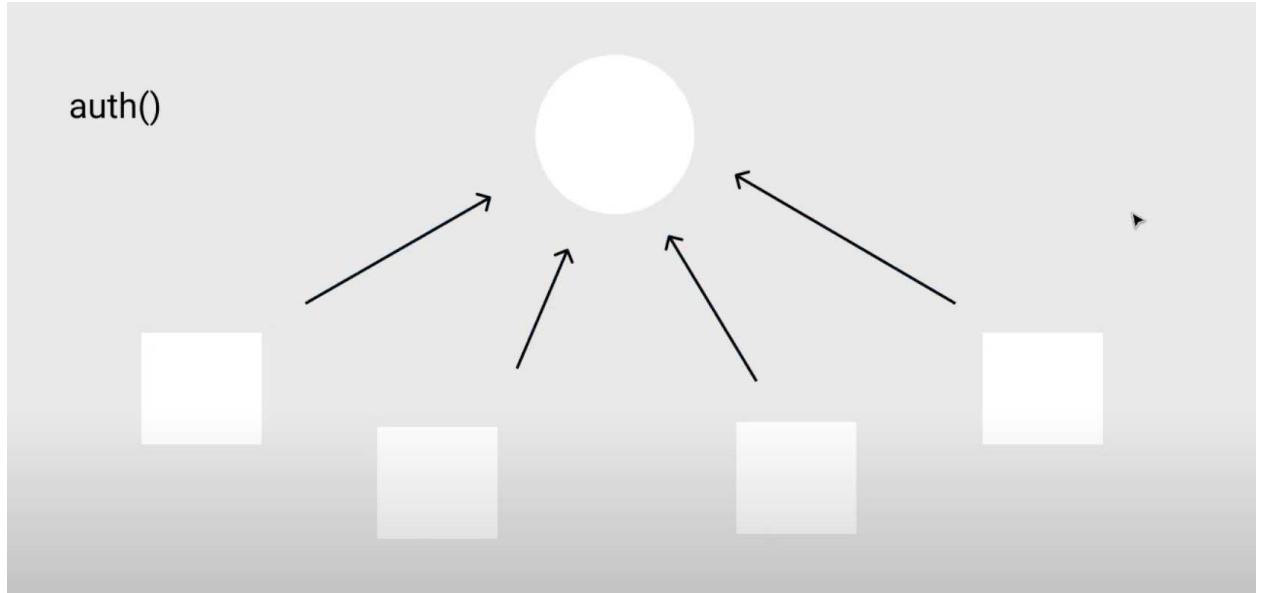


The screenshot shows a code editor with the file `AdminMiddleware.php` open. The file is located in the `LaravelBlog/app/Http/Middleware` directory. The code defines a middleware class `AdminMiddleware` that handles incoming requests.

```
use Illuminate\Http\Request;
class AdminMiddleware
{
    /**
     * Handle an incoming request.
     *
     * @param \Illuminate\Http\Request $request
     * @param \Closure(\Illuminate\Http\Request): (\Illuminate\Http\Response|\Illuminate\Http\RedirectResponse) $next
     */
    public function handle(Request $request, Closure $next)
    {
        return $next($request);
    }
}
```

Сам посредник (прежде чем получить ответ за запрос, он попадает в проводник)

4.17 Новая концепция auth



Круг – сервер
Квадраты – юзеры

У каждого запроса на сервер есть АВТОР, направляющий запрос, и мы можем взять запрос и посмотреть кто его отправил

Даже неавторизованный юзер отправляет запросы, тк он находится на сайте

В ларавел есть ХЕЛПЕР auth(), который не вернет никакого юзера, если человек не авторизовался, то в хелпер ничего не идет, но если авторизован, то в хелпер попадет пользователь отправивший запрос

У каждого запроса есть автор(пользователь) и чтобы мы смогли этот запрос просмотреть мы можем обратиться к хелперу auth

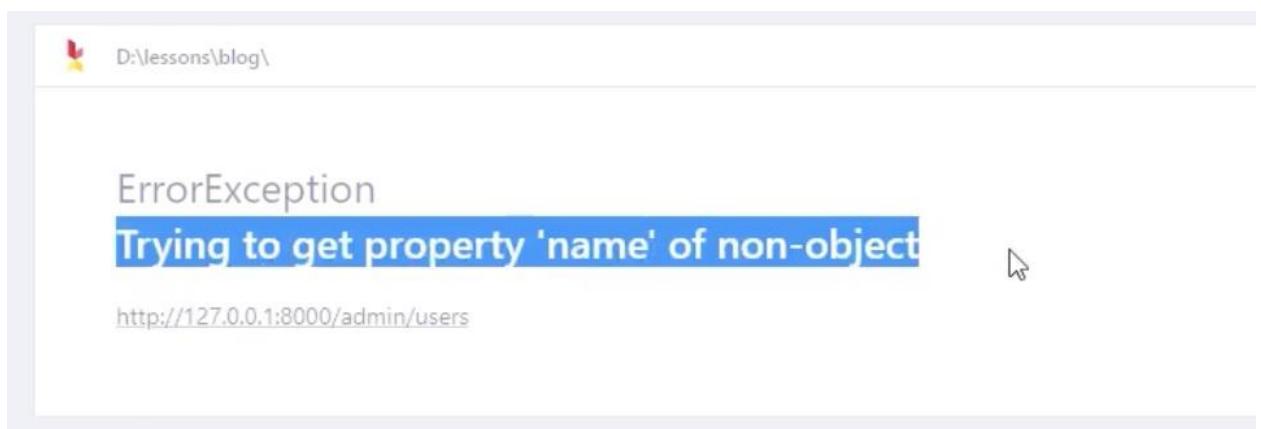
```
public function handle(Request $request, Closure $next)
{
    auth()->user()->name;

    return $next($request);
}
```

users

id	INTEGER (auto increment)
name	VARCHAR(255)
email	VARCHAR(255)
email_verified_at	DATETIME = NULL
password	VARCHAR(255)
remember_token	VARCHAR(100) = NULL
created_at	DATETIME = NULL
updated_at	DATETIME = NULL

Например тут мы обращаемся имени пользователя. Будет модель

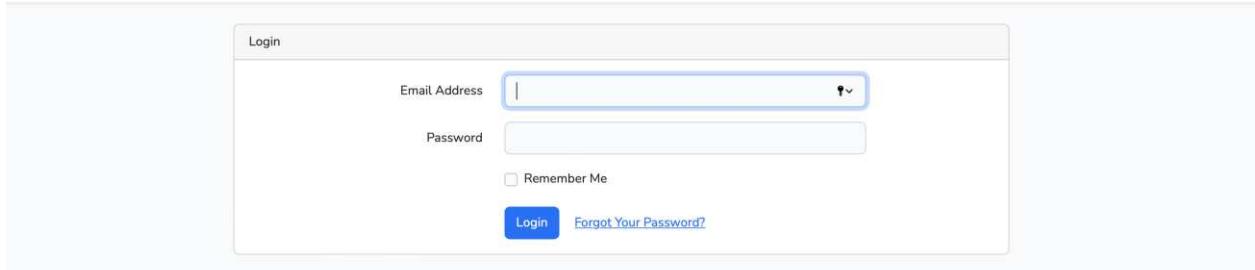


Если пользователь не авторизован то будет такая ошибка, тк хелпер вернул null

Чтобы не получать эту ошибку нужно подключить авторизацию

```
['namespace' => 'Admin', 'prefix' => 'admin', 'middleware' => ['auth', 'admin']], function() {
    group(['namespace' => 'Main'], function () {
        e::get('/', action: 'IndexController');
    });
}
```

Добавили к middleware ['auth'], благодаря которому мы проверяем авторизован юзер или нет



Получаем страницу авторизации, и если пользователь не авторизован его из админки выкинет

Далее если он авторизован то мы проверяем кто он вообще такой через посредника admin, и если он админ то мы его пропустим, а если нет то опять выкинем

```
public function handle(Request $request, Closure $next)
{
    dd(auth()->user()->name);
    return $next($request);
}
```

И теперь так мы можем получить имя юзера

```
public function handle(Request $request, Closure $next)
{
    if((int) auth()->user()->role !== User::ADMIN_ROLE){
        abort(404);
    }
    return $next($request);
}
```

Теперь можем сделать проверку на роль

```

<!-- Navbar -->
<nav class="main-header navbar navbar-expand navbar-white navbar-light">
    <!-- Left navbar links -->
    <div class="col-12 d-flex justify-content-between">
        <ul class="navbar-nav">
            <li class="nav-item">
                <a class="nav-link" data-widget="pushmenu" href="#" role="button"><i class="fas fa-bars"></i>
            </li>
        </ul>
        <ul class="navbar-nav">
            <li class="nav-item">
                <form action="{{route('logout')}}" method="POST">
                    @csrf
                    <input type="submit" value="Выйти" class="btn btn-outline-primary">
                </form>
            </li>
        </ul>
    </div>
</nav>

```

Кнопка выхода



4.18 Автогенерация пароля, отправка почты. Mailtrap и класс Mailable в Laravel

Напрямую через сайт сообщения на почту не отправляются, это уже давно прошлый век, все используют сторонние сервисы

```

MAIL_MAILER=smtp
MAIL_HOST=mailhog
MAIL_PORT=1025
MAIL_USERNAME=null
MAIL_PASSWORD=null
MAIL_ENCRYPTION=null
MAIL_FROM_ADDRESS=null
MAIL_FROM_NAME="${APP_NAME}"

```

В .env можно посмотреть конфигурацию почтового сервиса (локальная среда)

Нижеописанный сервис позволяет сымитировать работу стороннего сервиса на локальной среде

The screenshot shows the Mailtrap website's homepage. At the top, there's a navigation bar with links for Product, API, Pricing, Blog, Help, Log In, and Sign Up. Below the navigation is a section titled "Email Sandbox Service" with sub-headings: "Capture SMTP traffic from staging and dev environments", "Automate test flows and scenarios with flexible API", and "Analyze email content for spam score and validate HTML/CSS". A green "Sign Up For Free" button is prominently displayed. Below this, there are logos for various frameworks: RAILS, python, .NET, Symfony, and Laravel. The main content area features a large screenshot of the Mailtrap dashboard, which includes a sidebar with "Inboxes", "API", and "Billing" options, and a central panel showing a list of received emails and a detailed analysis of one specific message.

Для отправки писем на почту есть такой сайт

Авторизуемся

This screenshot shows the "Inboxes" section of the Mailtrap interface. On the left, a sidebar lists "Inboxes", "API", and "Billing". The main area is titled "Projects" and shows a single project named "My Project". Under "Inboxes", there is a list for "My Inbox" with the following details: Total Sent (2), Messages (0 / 0), Max size (50), and Last message (Empty). To the right, there are buttons for "Upgrade", "Add Project", and "Add Inbox".

Видим это

This screenshot shows the same "Inboxes" interface as above, but with a cursor pointing at the "MyInbox" link under the "Inboxes" heading. This indicates the user is about to click on it to view their messages.

Тык

My Inbox

SMTP Settings

Email Address

Auto Forward

Manual Forward

Team Members

SMTP / POP3  Reset Credentials 

Use these settings to send messages directly from your email client or mail transfer agent.

 Don't disclose your username or password as this may result in your inbox getting filled up with spam.

Show Credentials 

Integrations 

Laravel 7+



Laravel provides a clean, simple API over the popular SwiftMailer library.

With the default Laravel setup you can configure your mailing configuration by setting these values in the .env file in the root directory of your project.

```
MAIL_MAILER=smtp
MAIL_HOST=smtp.mailtrap.io
MAIL_PORT=2525
MAIL_USERNAME=a4ef222351af34
MAIL_PASSWORD=7950c40dd9bbf8
MAIL_ENCRYPTION=tls
```

Выбираем ларавел

```
MAIL_MAILER=smtp
MAIL_HOST=smtp.mailtrap.io
MAIL_PORT=2525
MAIL_USERNAME=de7c9a4070cc19
MAIL_PASSWORD=a9a2f048eb1ccb
MAIL_ENCRYPTION=tls
```

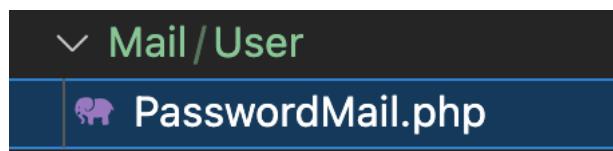
```
MAIL_MAILER=smtp
MAIL_HOST=smtp.mailtrap.io
MAIL_PORT=2525
MAIL_USERNAME=de7c9a4070cc19
MAIL_PASSWORD=a9a2f048eb1ccb
MAIL_ENCRYPTION=tls
MAIL_FROM_ADDRESS="hello@mail.ru"
MAIL_FROM_NAME="${APP_NAME}"
```

ОБЯЗАТЕЛЬНО ПЕРЕЗАПУСКАЕМ СЕРВЕР

Далее уже будем пользоваться средствами ларавел

```
● saint@MacBook-Pro-Artem LaravelBlog % php artisan make:mail User/PasswordMail -m mail.user.password
INFO Mail [app/Mail/User/PasswordMail.php] created successfully.
```

Создаем класс для отправки КОНЕРКТНОГО письма и также присвоим ему шаблон письма с помощью -т который мы оформим через blade



Создался класс отправки

```
create.blade.php
edit.blade.php
index.blade.php
show.blade.php
index.blade.php
> auth
< layouts
  admin.blade.php
  app.blade.php
  main.blade.php
< mail/user
  password.blade.php  u
< main
  main.blade.php
```

```
1 @component('mail::message')
2 # Introduction
3
4 The body of your message.
5
6 @component('mail::button', ['url' => ''])
7 Button Text
8 @endcomponent
9
10 Thanks,<br>
11 {{ config('app.name') }}
12 @endcomponent
```

И шаблон письма

Мы же будем отправлять пароль, а чтобы его отправить нужно сначала сгенерировать

```
class Service
{
    public function store($data)
    {
        $password = Str::random(10);
        $data['password'] = Hash::make($password);
        $user = User::FirstOrCreate(["email" => $data['email']], $data);
        Mail::to($data['email'])->send(new PasswordMail($password));
        return $user;
    }

    public function update($data, $user)
    {
        $user->update($data);
        return $user;
    }
}
```

Меняем сервис

```
public $password;

/**
 * Create a new message instance.
 *
 * @return void
 */
public function __construct($password)
{
    $this -> password = $password;
}
```

Меняем конструктор в классе письма

Убираем поле пароля в шаблоне создания

Убираем из реквестов пароли

```
@component('mail::message')
    Ваш пароль: {{$password}}
@endcomponent
```

Меняем шаблон

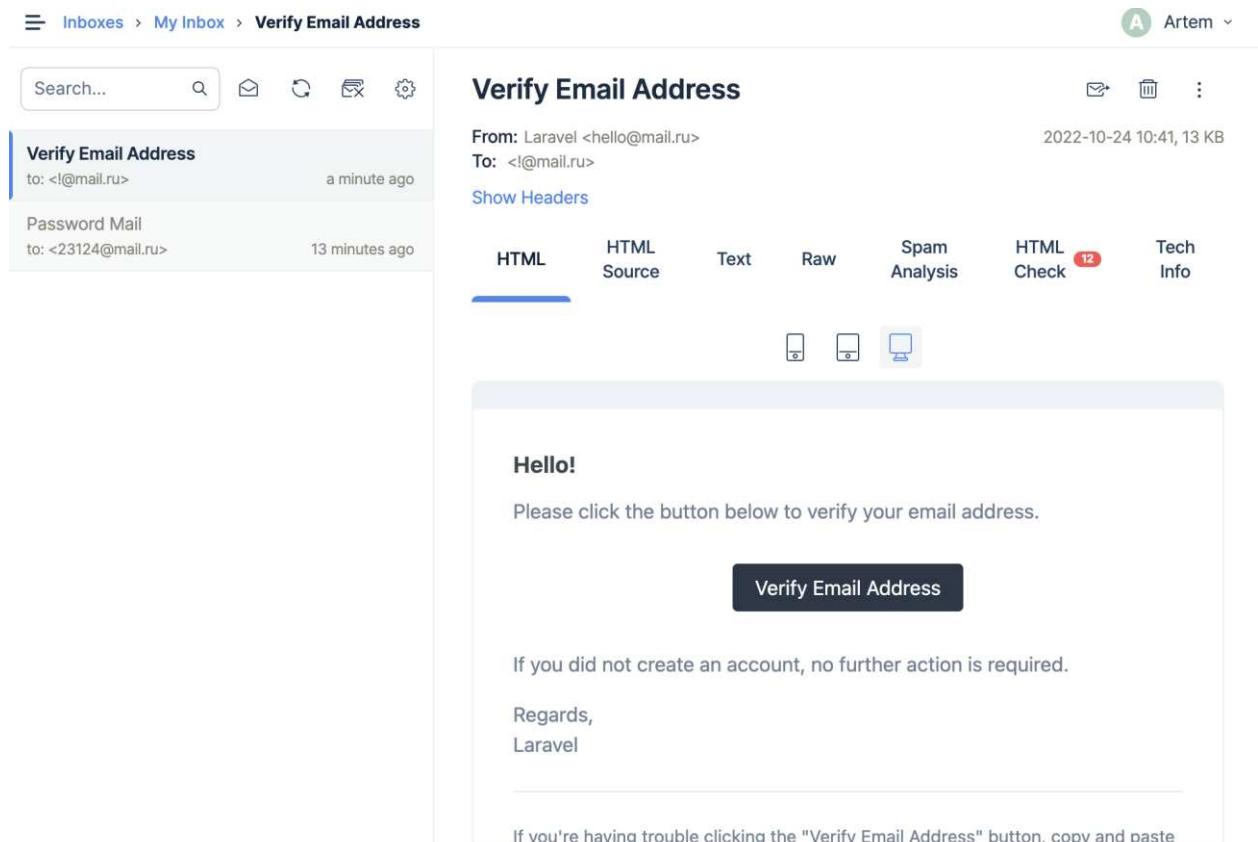
4.19 Отправление письма с подтверждением учетной записи Email verification

```
Auth::routes(['verify' => true]);
```

Включаем подтверждение почты в роуте

```
class User extends Authenticatable implements MustVerifyEmail
{
```

Имплементимся



The screenshot shows an email inbox with the following details:

- Header:** Verify Email Address
- From:** Laravel <hello@mail.ru>
- To:** <i@mail.ru>
- Date:** 2022-10-24 10:41, 13 KB
- Actions:** Delete, More options
- Content:** The email body contains a message:

Hello!
Please click the button below to verify your email address.

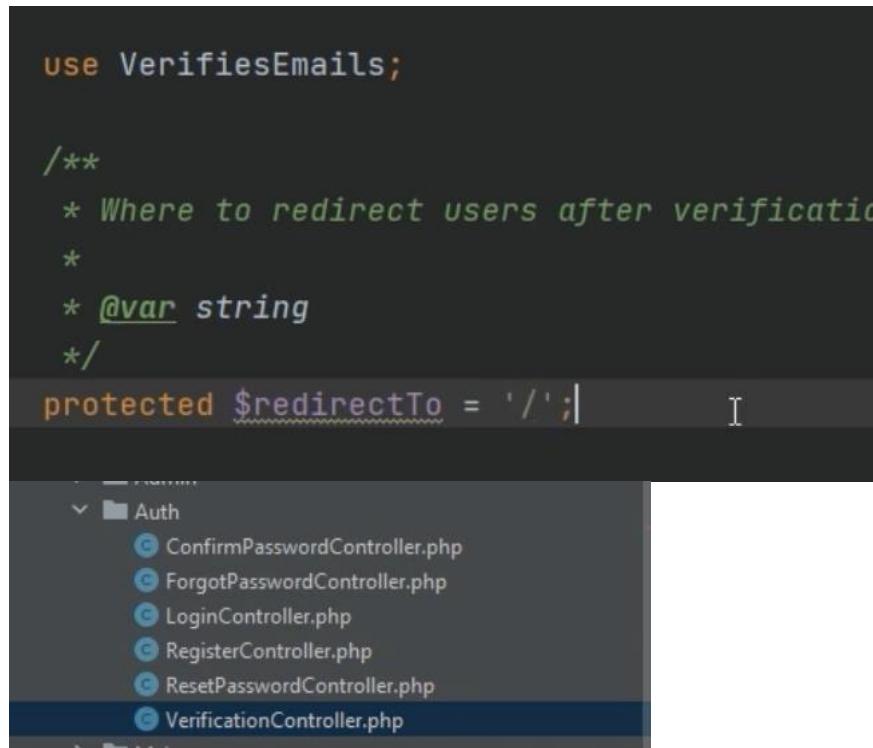
Verify Email Address

If you did not create an account, no further action is required.
Regards,
Laravel

Теперь при регистрации пользователя будет отправлять такое сообщение

```
'middleware' => ['auth', 'admin', 'verified']], f
```

При активации посредника страницы без подтверждения почты не будут доступны

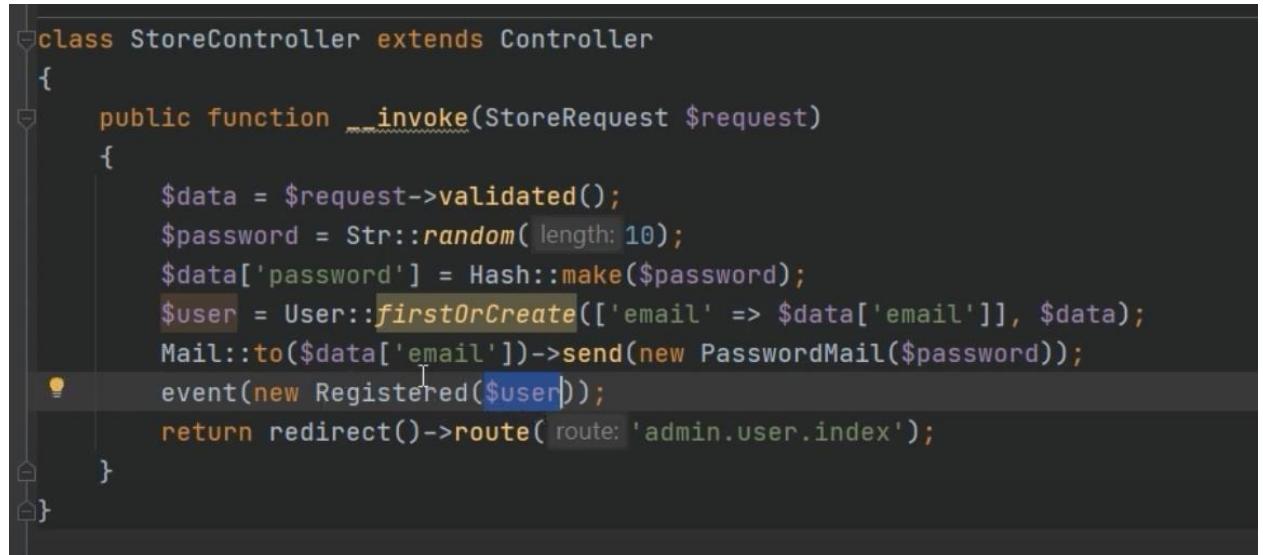


```
use VerifiesEmails;

/**
 * Where to redirect users after verification
 *
 * @var string
 */
protected $redirectTo = '/';
```

The screenshot shows a code editor with a sidebar containing a file tree. The tree has a single folder named 'Auth' which contains six files: ConfirmPasswordController.php, ForgotPasswordController.php, LoginController.php, RegisterController.php, ResetPasswordController.php, and VerificationController.php. The VerificationController.php file is currently open in the editor.

При подтверждении почты меняем страницу куда нас перекинут



```
class StoreController extends Controller
{
    public function __invoke(StoreRequest $request)
    {
        $data = $request->validated();
        $password = Str::random(length: 10);
        $data['password'] = Hash::make($password);
        $user = User::firstOrCreate(['email' => $data['email']], $data);
        Mail::to($data['email'])->send(new PasswordMail($password));
        event(new Registered($user));
        return redirect()->route('admin.user.index');
    }
}
```

Меняем сервис для отправки подтверждения письма через панель создания юзера

403 | THIS ACTION IS UNAUTHORIZED.

Если мы попробуем подтвердить аккаунт другого человека через свой token, то выдаст ошибку

4.20 Отправка писем с использованием очередей. Создаем свой Notification класс

Порядок действий

- 1) выполняем команду `php artisan queue:table` которая создаст миграцию
- 2) Запускаем миграцию `php artisan migrate` в базе данных будет создана новая таблица `Jobs`
- 3) Создаём свой класс отправки почты `php artisan make:notification SendVerifyWithQueueNotification` в папке `app` будет создана папка `Notifications` с файлом `SendVerifyWithQueueNotification.php`
- 4) Удаляем содержимое класса изменяя наследование на класс `VerifyEmail` и добавляем имплементацию, должно получится так:
`<?php namespace App\Notifications; use Illuminate\Bus\Queueable; use Illuminate\Contracts\Queue\ShouldQueue; use Illuminate\Notifications\Messages\MailMessage; use Illuminate\Auth\Notifications\VerifyEmail; class SendVerifyWithQueueNotification extends VerifyEmail implements ShouldQueue { use Queueable; }`
- 5) в модели `User` добавляем метод `public function sendEmailVerificationNotification() { $this->notify(new SendVerifyWithQueueNotification()); }`
- 6) в файле `.env` меняем значение параметра `QUEUE_CONNECTION` на `database` перезапустить сервер
- 7) для запуска действий находящихся в таблице `Jobs` нужно выполнить команду `php artisan queue:work`
- 8) для корректной ссылки в письме верификации в `.env` файле -
`APP_URL=http://127.0.0.1:8000/`
- 9) для использования очереди, при регистрации пользователя через админ панель, в файле `PasswordMail.php` (`app/Mail/User`) добавляем имплементацию
`class PasswordMail extends Mailable implements ShouldQueue`

4.21 Очереди. Проблематика

При регистрации нового пользователя проходит ОЧЕНЬ МНОГО ВРЕМЕНИ, это очень плохо.

ПОЧЕМУ это происходит?

Вся проблема в стороннем сервисе mailtrap. Тк он не находится в ларвел, то запрос идет куда-то на удаленный сервер, что занимает очень много времени

Веб-приложение конектится с сервисом – отдает ему письмо – страница перегружается

Много данных = огромная задержка

Это нужно ОПТИМИЗИРОВАТЬ С ПОМОЩЬЮ МЕХАНИЗМА ОЧЕРЕДЕЙ

По сути все действия и отправка запросов в сервис будут складироваться в таблицу бд, а пользователь сразу будет перенаправлен

Детальный разбор

```
queue
queue:batches-table
queue:clear
queue:failed
queue:failed-table
queue:flush
queue:forget
queue:listen
queue:monitor
queue:prune-batches
queue:prune-failed
queue:restart
queue:retry
queue:retry-batch
queue:table
queue:work
```

Если набрать php artisan(кью)

```
queue:table
```

Конкретно нас интересует эта команда, потому что с помощью нее и работают очереди

```
● saint@MacBook-Pro-Artem LaravelBlog % php artisan queue:table
```

```
INFO Migration created successfully.
```

Создает миграцию в которую будут помещаться действия по работе с очередями

```
return new class extends Migration
{
    /**
     * Run the migrations.
     *
     * @return void
     */
    public function up()
    {
        Schema::create('jobs', function (Blueprint $table) {
            $table->bigIncrements('id');
            $table->string('queue')->index();
            $table->longText('payload');
            $table->unsignedTinyInteger('attempts');
            $table->unsignedInteger('reserved_at')->nullable();
            $table->unsignedInteger('available_at');
            $table->unsignedInteger('created_at');
        });
    }
}
```

Так мы разгружаем работу сайта

Потом с помощью специальной команды мы это дело запустим (связи со сторонними сервисами)

```
● saint@MacBook-Pro-Artem LaravelBlog % php artisan migrate
```

```
INFO Running migrations.
```

```
2022_10_26_164534_create_jobs_table ..... 69ms DONE
```

в базе данных будет создана новая таблица Jobs

Задача:

- 1) Оптимизировать регистрацию (сделать так чтобы она проходила в фоне и не было загрузки), далее после запуска необходимой команды все JOBSы будут реализовываться

Для этого нужно создать свой вариант отправки почты

```
● saint@MacBook-Pro-Artem LaravelBlog % php artisan make:notification SendWithQueueNotification
[INFO] Notification [app/Notifications/SendWithQueueNotification.php] created successfully.
```

Создадим класс нотификации (уведомление)

```
<?php

namespace App\Notifications;

use Illuminate\Bus\Queueable;
use Illuminate\Contracts\Queue\ShouldQueue;
use Illuminate\Notifications\Messages\MailMessage;
use Illuminate\Notifications\Notification;

class SendWithQueueNotification extends Notification implements ShouldQueue
{
    use Queueable;

}
```

Заходим в класс и удаляем там всю реализацию и имплементимся от другого класса для реализации очередей

```
class User extends Authenticatable implements MustVerifyEmail
{
```

Переходим в модель user и тк мы реализуем класс mustverifyemail, то у нас появляется новый метод

```
public function sendEmailVerificationNotification()
{
}
```

С помощью этого метода мы можем перезаписать письмо по умолчанию

```

public function sendEmailVerificationNotification()
{
    $this->notify(new SendWithQueueNotification);
}

```

Так мы будем отправлять другой шаблон почты

```

class SendWithQueueNotification extends VerifyEmail implements ShouldQueue
{
    use Queueable;

}

```

Теперь у класса мы должны сменить наследование, тк там содержаться методы отправки письма, конфигурации, верификации и тд и тп

```

18      BROADCAST_DRIVER=log
19      CACHE_DRIVER=file
20      FILESYSTEM_DISK=local
21      QUEUE_CONNECTION=database
22      SESSION_DRIVER=file
23      SESSION_LIFETIME=120
24

```

В .env меняем подключение очереди с sync на database

Теперь страницы с регистрацией пользователя срабатывают моментально

НО ПИСЬМО НЕ ПРИШЛО

+ Параметры	id	queue	payload	attempts	reserved_at	available_at	created_at
<input type="checkbox"/> Изменить <input type="checkbox"/> Копировать <input type="checkbox"/> Удалить 1 default <input type="checkbox"/> Отметить все <input type="checkbox"/> С отмеченными: <input type="checkbox"/> Изменить <input type="checkbox"/> Показать все Количество строк: 25 <input type="checkbox"/> Фильтровать Использование результатов запроса <input type="checkbox"/> Печать <input type="checkbox"/> В буфер обмена <input type="checkbox"/> Экспорт <input type="checkbox"/> Отобразить			{"uuid": "7ee4a63a-cdc7-4ce2-b1e5-0bf14c923274", "displayName": "App\\Notifications\\SendWithQueueNotification", "job": "Illuminate\\Queue\\CallQueuedHandler@call", "maxTries": null, "maxExceptions": null, "failOnTimeout": false, "backoff": null, "timeout": null, "retryUntil": null, "data": {"commandName": "Illuminate\\Notifications\\SendQueuedNotifications", "command": "0:48:\\Illuminate\\Notifications\\SendQueuedNotifications", "queue": "default"}, "attempts": 1, "reserved_at": null, "available_at": 1666804721, "created_at": 1666804721}	1	null	1666804721	1666804721

Но в таблицу пришла какая-то хрень, а именно реализация отправление письма (запрос ушел в очередь)

Реализация которая должна была произойти сохранилась в виде определенных данных в таблице

```
$ php artisan queue:work
```

Теперь с помощью специальной команды мы можем запустить исполнение (пойдет по порядку очереди)

```
saint@MacBook-Pro-Artem LaravelBlog % php artisan queue:work
INFO Processing jobs from the [default] queue.
2022-10-26 17:22:17 App\Notifications\SendWithQueueNotification ..... 3,420.05ms DONE
```

Теперь письмо пришло, а таблица очистилась (полезная нагрузка ушла)

НО ПРИ ПОДТВЕРЖДЕНИИ МЫ ПОЛУЧИМ ОШИБКУ, тк теперь у нас запрос идет не на 127.0.0.1:8000 а на локал хост, а сервис отправки письма такое не понимает

5 APP_URL=http://127.0.0.1:8000

Поменяли адрес И ПЕРЕЗАГРУЗИЛИ СЕРВЕР

2) Оптимизируем добавление пользователя через админпанель

```
class StoreController extends Controller
{
    public function __invoke(StoreRequest $request)
    {
        $data = $request->validated();
        $password = Str::random(length: 10);
        $data['password'] = Hash::make($password);
        $user = User::firstOrCreate(['email' => $data['email']], $data);
        Mail::to($data['email'])->send(new PasswordMail($password));
        event(new Registered($user));
        return redirect()->route(route: 'admin.user.index');
    }
}
```

Перейдем в сторконтроллер и посмотрим на евент который запускает наш метод `SendVerifyWithQueueNotification` а

```
Mail::to($data['email'])->send(new PasswordMail($password));
```

отправляет новый класс в котором реализована очередь

```
class PasswordMail extends Mailable
{
    use Queueable, SerializesModels;
```

осталось добавить реализацию

```
class PasswordMail extends Mailable implements ShouldQueue
```

```
saint@MacBook-Pro-Artem LaravelBlog % php artisan queue:work
```

```
INFO Processing jobs from the [default] queue.
```

```
2022-10-26 17:32:03 App\Notifications\SendWithQueueNotification ..... 2,646.50ms DONE
2022-10-26 17:39:30 App\Mail\User>PasswordMail ..... 2,999.71ms DONE
2022-10-26 17:39:33 App\Notifications\SendWithQueueNotification ..... 914.57ms DONE
```

Теперь можно смотреть на запросы

На проде все будет работать шикарно, тк прод сервер никогда не выключается

На проде, при выгрузке сайта запускается СУПЕРВИЗИОР который выполняет по сути php artisan queue:work и супервизор тоже никогда не спит и очереди будут работать постоянно

!!!ОЧЕРЕДИ МОЖНО ИСПОЛЬЗОВАТЬ ВООБЩЕ ВЕЗДЕ (контроллеры, миграции, запросы и тд и тп), но не всегда это имеет смысл

!!!В ОЧЕРЕДЬ МОЖНО ЗАНЕСТИ РЕАЛИЗАЦИЮ ЛЮБОГО ДЕЙСТВИЯ

!!!ОЧЕРЕДЬ ОТРАБАТЫВАЕТ НА ФОНЕ, ЮЗЕР НЕ ВИДИТ РЕАЛИЗАЦИИ И ЗАГРУЗКИ

!!!в очереди имеет смысл отправлять запросы со сторонних сервисов, которые срабатывают не мгновенно

4.22 Создаем класс Job. Пример работы с очередями любого алгоритма/действия

В прошлом параграфе говорилось о том что любо действие можно реализовать посредством очереди. Рассмотрим как это делать.

```
class Service
{
    public function store($data)
    {
        $password = Str::random(10);
        $data['password'] = Hash::make($password);
        $user = User::FirstOrCreate(["email" => $data['email']], $data);
        Mail::to($data['email'])->send(new PasswordMail($password));
        event(new Registered($user));
        return $user;
    }
}
```

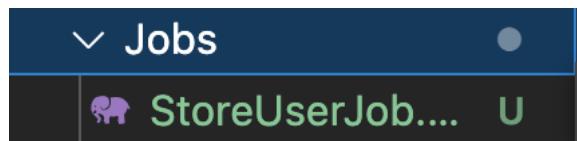
Засуним реализацию добавление пользователя в очередь и выполниться она будет только при запуске queue:work



В артизан есть такая функция

Эта функция создает класс, куда можно записать любое действие и после queue:work оно выполнится

```
● saint@MacBook-Pro-Artem LaravelBlog % php artisan make:job StoreUserJob
INFO  Job [app/Jobs/StoreUserJob.php] created successfully.
```



```

class StoreUserJob implements ShouldQueue
{
    use Dispatchable, InteractsWithQueue, Queueable, SerializesModels;

    /**
     * Create a new job instance.
     *
     * @return void
     */
    public function __construct()
    {
        //
    }

    /**
     * Execute the job.
     *
     * @return void
     */
    public function handle()
    {
        //
    }
}

```

Все что мы укажем в handle сохранится в таблице jobs

```

public function handle()
{
    $password = Str::random(10);
    $data['password'] = Hash::make($password);
    $user = User::FirstOrCreate(['email' => $data['email']], $data);
}

```

Но теперь у нас недоступны переменные

```

public function __construct($data)
{
    $this->data = $data;
}

```

Передадим в конструктор дату и проинициализируем ее через указатель this

```
class StoreUserJob implements ShouldQueue
{
    use Dispatchable, InteractsWithQueue, Queueable, SerializesModels;

    public $data;
}
```

Создаем св-во data (у меня работает только с публич)

```
$password = Str::random(10);
$this->data['password'] = Hash::make($password);
$user = User::FirstOrCreate(["email" => $this->data['email']], $this->data);
Mail::to($this->data['email'])->send(new PasswordMail($password));
event(new Registered($user));
```

```
class Service
{
    public function store($data)
    {
        StoreUserJob::dispatch($data);
    }
}
```

В сервисе так мы можем вызвать наш job

```
namespace App\Jobs;

use App\Mail\User>PasswordMail;
use App\Models\User;
use Illuminate\Auth\Events\Registered;
use Illuminate\Bus\Queueable;
use Illuminate\Contracts\Queue\ShouldBeUnique;
use Illuminate\Contracts\Queue\ShouldQueue;
use Illuminate\Foundation\Bus\Dispatchable;
use Illuminate\Queue\InteractsWithQueue;
use Illuminate\Queue\SerializesModels;
use Illuminate\Support\Facades\Hash;
use Illuminate\Support\Facades\Mail;
use Illuminate\Support\Str;
```

Не забываем перенести все зависимости

!!!ПЕРЕЗАПУСКАЕТ СЕРВЕР И ДЖОБЫ

+ Параметры	Id	queue	payload	attempts	reserved_at	available_at	created_at
<input type="checkbox"/> Изменить <input type="checkbox"/> Копировать <input type="checkbox"/> Удалить	1	default	{"uuid":"7536b125-0dba-461c-b5b2-85a1a510e252","di...	0	NULL	1666820881	1666820881

Получаем запрос в таблицу очереди

```
o saint@MacBook-Pro-Artem LaravelBlog % php artisan queue:work
INFO Processing jobs from the [default] queue.

2022-10-26 21:49:55 App\Jobs\StoreUserJob ..... 185.33ms DONE
2022-10-26 21:49:55 App\Mail\User\PasswordMail ..... 3,519.39ms DONE
2022-10-26 21:49:58 App\Notifications\SendWithQueueNotification ..... 1,019.92ms DONE
```

После запуска команды все работает и пользователь создается

!! ТАК МОЖНО ДЕЛАТЬ ИМПОРТ С ЭКСЕЛЬ ТАБЛИЦ, ИНТЕГРАЦИИ СО СТОРОННИМИ СЕРВИСАМИ И ТД И ТП, так система грузиться не будет, и не будет вечной загрузки

4.23 Делаем общие косметические изменения по дизайну и редиректам

Расставим карточки

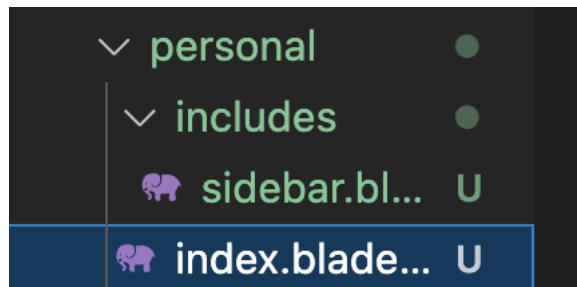
```
class IndexController extends Controller
{
    public function __invoke()
    {
        $data = [];
        $data['usersCount'] = User::all()->count();
        $data['postsCount'] = Post::all()->count();
        $data['categoriesCount'] = Category::all()->count();
        $data['tagsCount'] = Tag::all()->count();
        return view('admin.main.index', compact('var_name: 'data'));
    }
}
```

4.24 Создаем личный кабинет пользователя

- 1) Шаблон личного кабинета

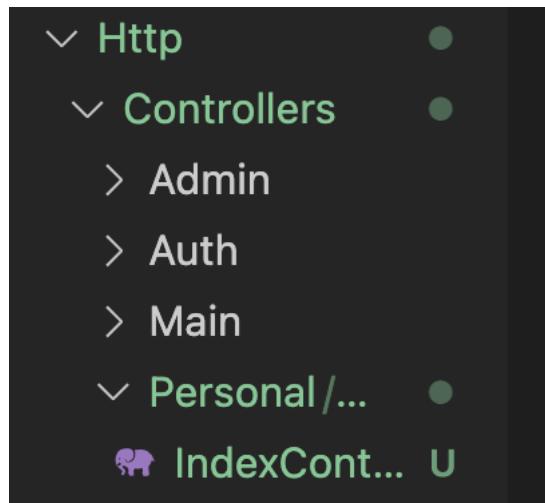
```
Route::group(['namespace' => 'App\Http\Controllers\Personal', 'prefix' => 'personal', 'middleware' => ['auth', 'verified']], function () {  
    Route::get('/', 'IndexController')->name('personal.main.index');  
});
```

Создаем роуты для лк

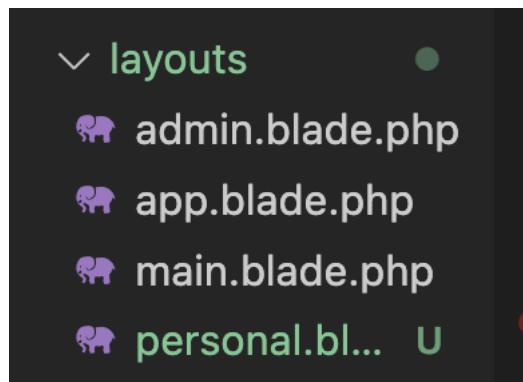


Создаем view

Меняем там все что хотим

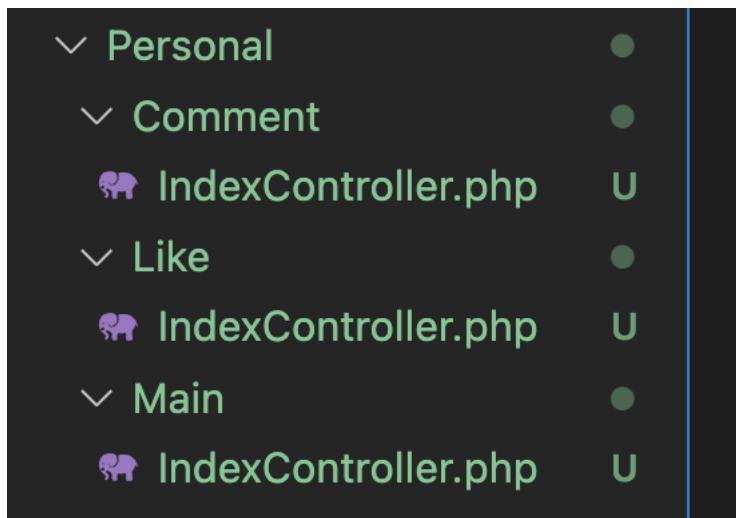


Создаем контроллеры



```
@extends('layouts.personal')
@section('content')
```

Меняем шаблоны инклуд



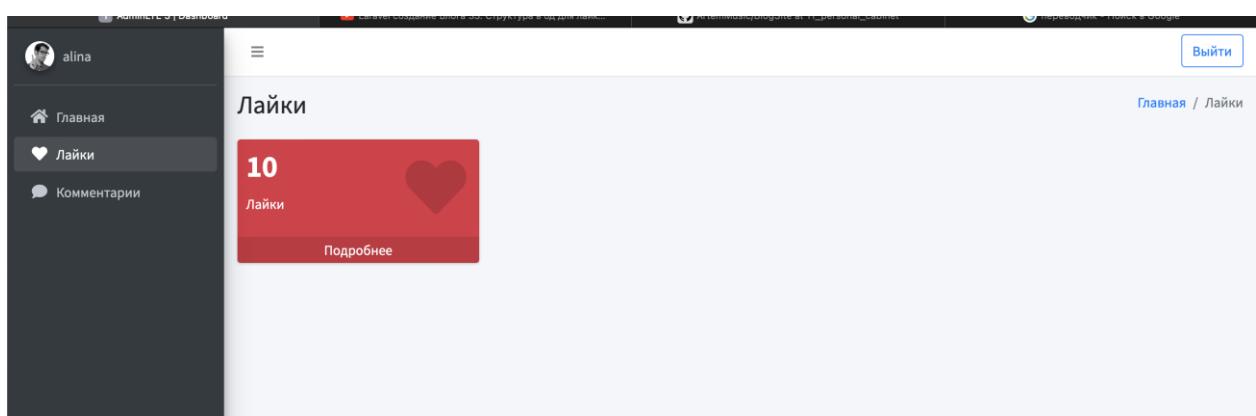
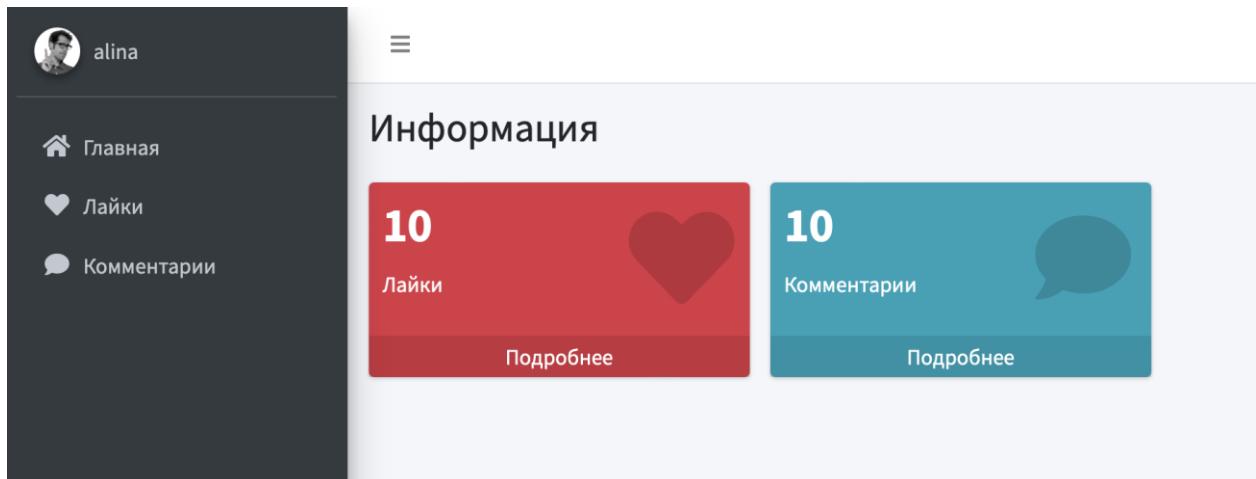
```
//Личный кабинет
Route::group(['namespace' => 'App\Http\Controllers\Personal', 'prefix' => 'personal'],
    [
        //Лайки
        Route::group(['namespace' => 'Like', 'prefix' => 'likes'], function () {
            Route::get('/', 'IndexController')->name('personal.like.index');
        });

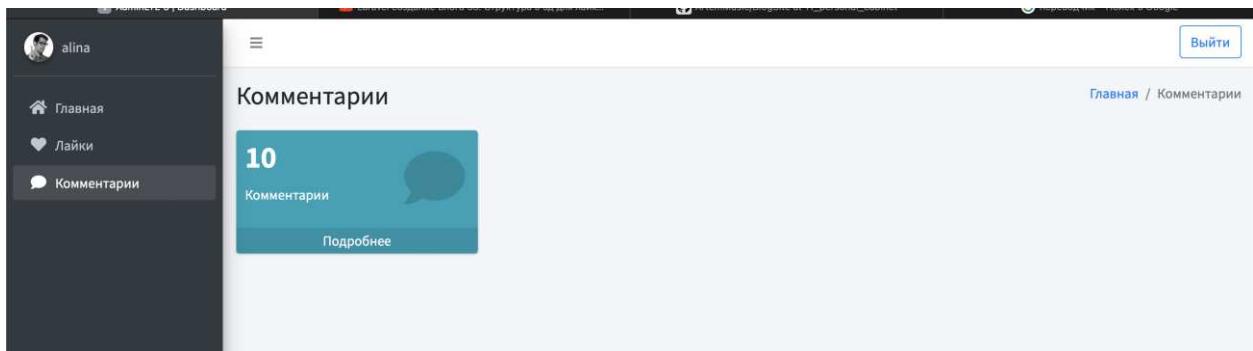
        //Комментарии
        Route::group(['namespace' => 'Comment', 'prefix' => 'comments'], function () {
            Route::get('/', 'IndexController')->name('admin.comment.index');
        });
    ]
);
```

Настраиваем контроллеры и роуты

```
personal
└── comment
    └── index.blade.php
includes
└── sidebar.blade.php
like
└── index.blade.php
└── index.blade.php
└── welcome.blade.php
```

Создадим view





В конечном итоге нужно чтобы было это

4.25 Структура в бд для лайков и их отображение в личном кабинете

Отношение посты-юзеры будет многое ко многим (у постов много лайков от юзеров и у юзеров много лайкнутых постов) для этого нужно создать таблицу объединения юзеров и постов

```
○ saint@MacBook-Pro-Artem LaravelBlog % php artisan make:model PostUserLike -m
    INFO Model [app/Models/PostUserLike.php] created successfully.
    INFO Created migration [2022_10_29_105719_create_post_user_likes_table].
    |
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class PostUserLike extends Model
{
    use HasFactory;
    protected $table = "post_user_likes";
    protected $guarded = false;
}
```

```

public function up()
{
    Schema::create('post_user_likes', function (Blueprint $table) {
        $table->id();
        $table->unsignedBigInteger('post_id');
        $table->unsignedBigInteger('user_id');
        $table->timestamps();

        $table->index('post_id', 'pul_post_idx');
        $table->index('user_id', 'pul_user_idx');

        $table->foreign('post_id', 'pul_post_fk')->on('posts')->references('id');
        $table->foreign('user_id', 'pul_user_fk')->on(['users'])->references('id');
    });
}

```

```

● saint@MacBook-Pro-Artem LaravelBlog % php artisan migrate
  INFO  Running migrations.
  2022_10_29_105719_create_post_user_likes_table ..... 190ms DONE
○ saint@MacBook-Pro-Artem LaravelBlog %

```

Далее в индекс вставляем шаблон индекса как у категорий админки

Но тут возникает проблема, как сделать так чтобы пользователь видел только посты которые ОН лайкнул?

```

class IndexController extends Controller
{
    public function __invoke()
    {
        auth()->user();
        return view('personal.like.index');
    }
}

```

А вот так

Так мы вытаскиваем конкретного юзера

```

class IndexController extends Controller
{
    public function __invoke()
    {
        $posts = auth()->user()->likedPosts();
        return view('personal.like.index', compact('posts'));
    }
}

```

Так мы вытаскиваем лайкнутые посты через модель юзер

```

public function likedPosts()
{
    return $this->belongsToMany(Post::class, 'post_user_likes', 'user_id', 'post_id');
}

```

Отношения многие ко многим чтобы вытащить посты

```

Illuminate\Database\Eloquent\Collection {#1287 ▼ // app/Http/Controllers/Personal/Like/IndexController.php:12
#items: array:1 [▼
  0 => App\Models\Post {#1306 ▼
    #connection: "mysql"
    #table: "posts"
    #primaryKey: "id"
    #keyType: "int"
    +incrementing: true
    #with: []
    #withCount: []
    +preventsLazyLoading: false
    #perPage: 15
    +exists: true
    +wasRecentlyCreated: false
    #escapeWhenCastingToString: false
    #attributes: array:9 [▶]
    #original: array:11 [▶]
    #changes: []
    #casts: array:1 [▶]
    #classCastCache: []
    #attributeCastCache: []
    #dates: []
    #dateFormat: null
    #appends: []
    #dispatchesEvents: []
    #observables: []
    #relations: array:1 [▶]
    #touches: []
    +timestamps: true
    #hidden: []
    #visible: []
    #fillable: []
    #guarded: false
    #forceDeleting: false
  }
]
#escapeWhenCastingToString: false
}

```

Вот что приходит

```
class DeleteController extends Controller
{
    public function __invoke(Post $post)
    {
        auth()->user()->likedPosts()->detach($post->id);
        return redirect()->route(['personal.main.index']);
    }
}
```

Так реализуется удаление

Разница тут в том, что мы не удаляем пост, а открепляем его
Тут мы также обращаемся к пользователю к методу likesposts, но уже
запросом в базу данных с помощью (). Далее открепляем пост и
перенаправляем на индекс

4.26 Структура в бд для комментариев и их отображение в личном кабинете

Отношения комментарии-посты – один ко многим

Отношения юзер комментарии – один ко многим

При создании модели комментарии внутри мы можем прописать сразу
отношения и с постами и с пользователями

```
class Comment extends Model
{
    use HasFactory;
    protected $table = "comments";
    protected $guarded = false;
}
```

```
public function up()
{
    Schema::create('comments', function (Blueprint $table) {
        $table->id();
        $table->unsignedBigInteger('user_id');
        $table->unsignedBigInteger('post_id');
        $table->text('message');

        $table->index('user_id', 'comments_user_idx');
        $table->index('post_id', 'comments_post_idx');

        $table->foreign('post_id', 'comments_post_fk')->on('posts')->references('id');
        $table->foreign('user_id', 'comments_user_fk')->on(['users'])->references('id');

        $table->timestamps();
    });
}
```

```
public function comments()
{
    return $this->hasMany(Comment::class, 'user_id', 'id');
}
```

В МОДЕЛИ ЮЗЕР

```
<?php

namespace App\Http\Controllers\Personal\Comment;

use App\Http\Controllers\Controller;

class IndexController extends Controller
{
    public function __invoke()
    {
        $comments = auth()->user()->comments;
        return view('personal.comment.index', compact('comments'));
    }
}
```

Далее делаем вью

Комментарии

ID	Название	Действия
----	----------	----------

1	122121	 
---	--------	---

```
//Комментарии
Route::group(['namespace' => 'Comment', 'prefix' => 'comments'], function () {
    Route::get('/', 'IndexController')->name('personal.comment.index');
    Route::patch('/{comment}/edit', 'EditController')->name('personal.comment.edit');
    Route::get('/{comment}', 'IndexController')->name('personal.comment.update');
    Route::delete('/{comment}', 'DeleteController')->name('personal.comment.delete');
});
```

```
class DeleteController extends Controller
{
    public function __invoke(Comment $comment)
    {
        $comment->delete();
        return redirect()->route('personal.comment.index');
    }
}
```

```
class EditController extends Controller
{
    public function __invoke(Comment $comment)
    {
        return view('personal.comment.edit', compact('comment'));
    }
}
```

Для едит нужно сделать еще и выю

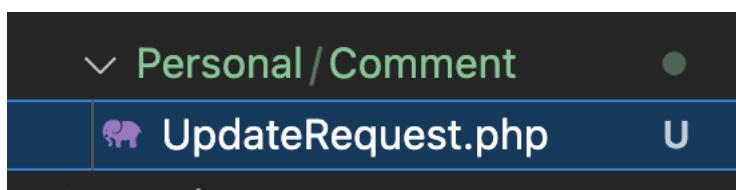
Комментарии

123

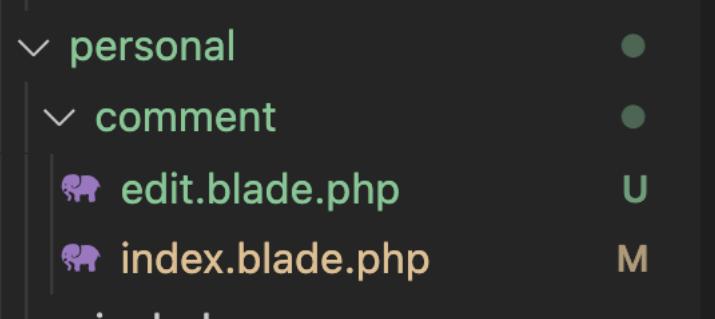
Изменить

Сбросить

```
class UpdateController extends Controller
{
    public function __invoke(UpdateRequest $request, Comment $comment)
    {
        $data=$request->validated();
        $comment->update($data);
        return redirect()->route('personal.comment.index');
    }
}
```



```
* @return array<string, mixed>
*/
public function rules()
{
    return [
        "message"=>'required|string'
    ];
}
```



4.27 Вывод ленты постов. Случайные посты. Популярные посты. Пагинация

См коммит 12 на гите

A screenshot of a code editor showing the 'Post.php' file. The file is located at 'app/Models/Post.php'. The code defines a Model for 'posts' with various methods like 'tags()', 'category()', and 'likedUsers()'. The code editor has tabs for other files like 'AppServiceProvider.php', 'Post.php', 'IndexController.php', and 'verify.b'. The code is syntax-highlighted with numbers on the left.

```

public function sendEmailVerificationNotification()
{
    $this->notify(new SendWithQueueNotification);
}

public function likedPosts()
{
    return $this->belongsToMany(Post::class, 'post_user_likes', 'user_id', 'post_id');
}

public function comments()
{
    return $this->hasMany(Comment::class, 'user_id', 'id');
}

```

```

namespace App\Http\Controllers\Main;

use App\Http\Controllers\Controller;
use App\Models\Post;

class IndexController extends Controller

    public function __invoke()
    {
        // $this -> authorize('view', auth()->user());
        $posts = Post::paginate(6);
        $randomPosts = Post::get()->random(4);
        $likedPosts = Post::withCount('likedUsers')->orderBy('liked_users_count', 'DESC')->get()->take(4); //All posts with more than 1 like
        return view('main.index', compact('posts', 'randomPosts', 'likedPosts'));
    }

```

```

<h3 class="edica-page" data-aos="fade-up">Последние посты</h3>
<br>
<div class="row">
    @foreach($posts as $post)
        <div class="col-md-4 fetured-post blog-post" data-aos="fade-right">
            <div class="blog-post-thumbnail-wrapper">
                
            </div>
            <p class="blog-post-category">{{$post->category->title}}</p>
            <a href="#" class="blog-post-permalink">
                <h6 class="blog-post-title">{{$post->title}}</h6>
            </a>
        </div>
    @endforeach
</div>

```

```
<div class="row">
<div class="mx-auto" style="margin-top: -80px;">
    {{$posts->links()}}
</div>
</div>
```

```
<h3 class="edica-page" data-aos="fade-up">Случайные посты</h3>
<br>
<div class="row">
    <div class="col-md-8">
        <section>
            <div class="row blog-post-row">
                @foreach($randomPosts as $randpost)
                    <div class="col-md-6 blog-post" data-aos="fade-up">
                        <div class="blog-post-thumbnail-wrapper">
                            
                        <p class="blog-post-category">{{$randpost->category->title}}<
                        <a href="#" class="blog-post-permalink">
                            <h6 class="blog-post-title">{{$randpost->title}}</h6>
                        </a>
                    </div>
                @endforeach
            </div>
        </section>
    </div>
</div>
```

```
<div class="widget widget-post-list">
    <h5 class="widget-title">Популярные посты</h5>
    <ul class="post-list">
        @foreach($likedPosts as $likedPost)
            <li class="post">
                <a href="#" class="post-permalink media">
                    
                    <div class="media-body">
                        <h6 class="post-title">{{$likedPost->title}}</h6>
                    </div>
                </a>
            </li>
        @endforeach
    </ul>
</div>
```

4.28 Carbon даты. Добавляем схожие посты (одиночные страницы для поста)

```
Illuminate\Support\Carbon @1631287809 {#333 ▾
  #endOfTime: false
  #startOfTime: false
  #constructedObjectId: "0000000186ce0b5000000005dcf7e9e"
  #localMonthsOverflow: null
  #localYearsOverflow: null
  #localStrictModeEnabled: null
  #localHumanDiffOptions: null
  #localToStringFormat: null
  #localSerializer: null
  #localMacros: null
  #localGenericMacros: null
  #localFormatFunction: null
  #localTranslator: null
  #dumpProperties: array:3 [▶]
  #dumpLocale: null
  date: 2021-09-10 15:30:09.0 UTC (+00:00)
}
```

```
class ShowController extends Controller
{
    public function __invoke(Post $post)
    {
        $data = Carbon::parse($post->created_at);
        return view('post.show', compact('post', 'data'));
    }
}
```

```
class ShowController extends Controller
{
    public function __invoke(Post $post)
    {
        $date = Carbon::parse($post->created_at);
        dd($date->format('Y-m-d'));
        return view('post.show', compact('var_name: 'post'));
    }
}
```

```
 {{ $date->format('F') }}February 1,
="300">
```

```
class ShowController extends Controller
{
    public function __invoke(Post $post)
    {
        Carbon::setLocale('ru_RU');
        $data = Carbon::parse($post->created_at);
        return view('post.show', compact('post', 'data'));
    }
}
```

```
/*
public function boot()
{
    Carbon::setLocale('ru_RU');
    Paginator::useBootstrap();
}
```

```
class ShowController extends Controller
{
    public function __invoke(Post $post)
    {
        $date = Carbon::parse($post->created_at);
        return view('post.show', compact('post', 'date'));
    }
}
```

```
>{{ $date->translatedFormat('F') }} {{ $date->format('d') }}, 20{{ $date-
    'Y' }}
```

```
 {{ $post->comments->count() }} Комментариев</p>
```

```
public function comments()
{
    return $this->hasMany(related: Comment::class, foreignKey: 'post_id', localKey: 'id');
}
```

34234

октябрь 20, 2022 17:58 • О Комментариях

```
<div class="col-lg-9 mx-auto" data-aos="fade-up">
|   {!! $post->content !!}
</div>
```

```

    <h2 class="section-title mb-5" data-aos="fade-up">Оставить комментарий</h2>
    <form action="/" method="post">
        <div class="row">
            <div class="form-group col-12" data-aos="fade-up">
                <label for="message" class="sr-only">Комментарий</label>
                <textarea name="message" id="message" class="form-control" placeholder="Комментарий" rows="3"></textarea>
                <input type="hidden" value="{{ $post->id }}" name="post_id">
            </div>
        </div>
        <div class="row">
            <div class="col-12" data-aos="fade-up">
                <input type="submit" value="Отправить" class="btn btn-warning">
            </div>
        </div>
    </form>
</section>

@csrf
```

Тут мы в скрытом поле передаем наш пост и так мы можем создать отдельный круд для постов

```
//Отдельный Пост
Route::group(['namespace' => 'App\Http\Controllers\Post', 'prefix' => 'posts'], function()
{
    Route::get('/', 'IndexController')->name('post.index');
    Route::get('/{post}', 'ShowController')->name('post.show');

    //post/{id}/comments
    Route::group(['namespace' => 'Comment', 'prefix' => '/{post}/comments'], function()
    {
    });
});
```

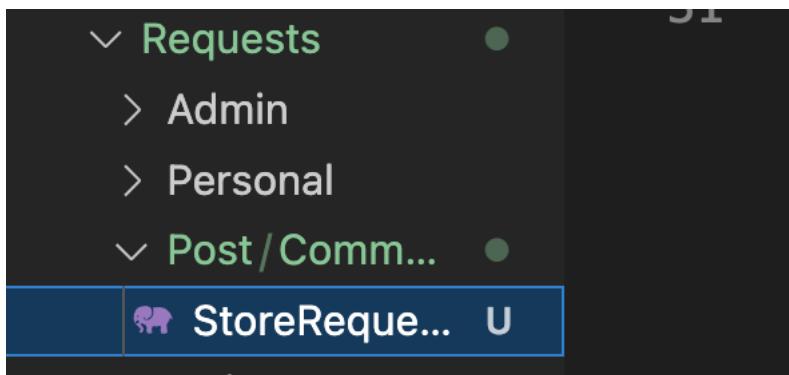
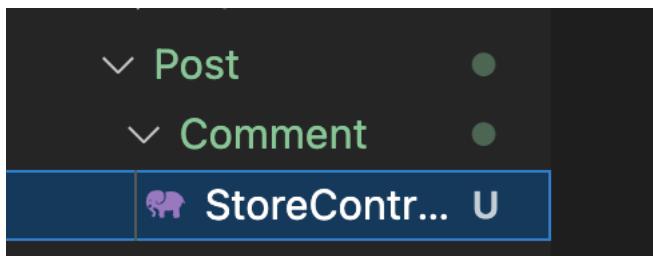
А вот так мы можем реализовать добавление комментариев путем вложенного роута (оч крутая и полезная штука) – nested route

```
//post/{id}/comments
Route::group(['namespace' => 'Comment', 'prefix' => '/{post}/comments'], function()
{
    Route::post('/', 'StoreController')->name('post.comment.store');
});
```

И теперь пойдет все вот так

Это очень удобно и нам не надо создавать всякую кашу

Далее создаем контроллеры и реквесты



```
<form action="{{route('post.comment.store', $post->id)}}> method="post">
```

```
class StoreController extends Controller
{
    public function __invoke(Post $post, StoreRequest $request)
    {
        $data = $request->validated();
        dd($data);
    }
}
```

```
class StoreController extends Controller
{
    public function __invoke(Post $post, StoreRequest $request)
    {
        $data = $request->validated();
        $data['user_id'] = auth() -> user() -> id;
        $data['post_id'] = $post->id;
        Comment::create($data);

        return redirect() -> route('post.show', $post->id);
    }
}
```

Тут мы получаем наш message, но для коммента нам нужен еще и айди юзера и айди поста, так мы их получаем и создаем коммент.

НЕАВТОРИЗОВАННОМУ ЮЗЕРУ ВЫДАСТ ОШИБКУ, а авторизованному создаст комментарий

Далее сделаем вывод комментариев

```
<section class="comment-list">
    @foreach($post->comments as $comment)
        <div class="comment-text">
            <span class="username">
                <div>{{$comment->user->name}}</div>
                <span class="text-muted float-right">data</span>
            </span><!-- /.username -->
            $comment->message
        </div>
    @endforeach
</section>
```

Так можно сделать вывод (но будет ошибка тк нет отношений comment-юзер)

```
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Comment extends Model
{
    use HasFactory;
    protected $table = "comments";
    protected $guarded = false;

    public function user()
    {
        return $this->belongsTo(User::class, 'user_id', 'id');
    }
}
```

```
<section class="comment-list">
    @foreach($post->comments as $comment)
        <div class="comment-text">
            <span class="username">
                <div>{{$comment->user->name}}</div>
                <span class="text-muted float-right">{{$comment->created_at}}</span>
            </span><!-- /.username -->
            {{$comment->message}}
        </div>
    @endforeach
</section>
```

Выводим

артем
111

2022-11-02 15:48:23

Далее обернем дату, но чтобы не писать всякую хрень с загромождением обернем дату вот так:

Будем использовать ГЕТТЕРЫ

В ларавел с помощью геттеров можно создавать ДИНАМИЧЕСКИЕ АТРИБУТЫ

```
public function getDateAsCarbonAttribute()
{
    return 1111;
}
```

В начале get и в конце attribute ОБЯЗАТЕЛЬНО

```
<section class="comment-list">
    @foreach($post->comments as $comment)
        @dd($comment->dateAsCarbon)
```

Теперь так можно вызывать атрибут

```
public function getDateAsCarbonAttribute()
{
    return Carbon::parse($this->created_at);
}
```

Делаем конвертацию даты

```
<section class="comment-list">
    @foreach($post->comments as $comment)
        <div class="comment-text">
            <span class="username">
                <div>{{$comment->user->name}}</div>
                <span class="text-muted float-right">{{$comment->dateAsCarbon->diffForHumans()}}</span>
            </span><!-- /.username -->
            {{$comment->message}}
        </div>
    @endforeach
</section>
```

ТЕПЕРЬ МЫ МОЖЕМ СДЕЛАТЬ вывод даты через метод карбона который работает только с датами обернутыми в карбон

artem

111

42 минуты назад

```
<section class="comment-list mb-5">
    <h3 class="mb-4">Комментарии({{$post->comments->count()}})</h3>
    @foreach($post->comments as $comment)
        <div class="comment-text mb-3">
            <span class="username">
                <div><b>{{$comment->user->name}}</b></div>
                <span class="text-muted float-right">{{$comment->dateAsCarbon->diffForHumans()}}</span>
            </span><!-- /.username -->
            {{$comment->message}}
        </div>
    @endforeach
</section>
```

Финальный вариант

Комментарии(4)

artem

111

48 минут назад

artem

efewgyu

6 минут назад

artem

100% чурка

2 минуты назад

artem

21

32 секунды назад

Оставить комментарий

Комментарий

НО форма комментариев нужна только для авторизированных пользователей

Воспользуемся новой директивой blade @auth()

```
@auth()
<section class="comment-section">
    <h3 class="comment-section__title">Comments</h3>
    @foreach($comments as $comment)
        <div class="comment">
            <span class="comment__author">{{ $comment->user->name }}</span>
            <div><b>{{ $comment->content }}</b></div>
            <span class="comment__date">{{ $comment->created_at }}</span>
        </div>
    @endforeach
</section>
@endauth()
```

```
@auth()
<a class="nav-link" href="{{ route('personal.main.index') }}">Войти</a>
@endauth
@guest()
<a class="nav-link" href="{{ route('personal.main.index') }}">Личный кабинет</a>
@endguest
```

Так можно сделать и войти-личный кабинет

4.30 Добавление лайков к постам. Бонусом - вывод постов по категориям

Создадим вот так:



СНЮС



Сафонов. Заведет ли пузырь карту?

```
<h3 class="edica-page" data-aos="fade-up">Последние посты</h3>
<br>
<div class="row">
    @foreach($posts as $post)
        <div class="col-md-4 fetured-post blog-post" data-aos="fade-right">
            <div class="blog-post-thumbnail-wrapper">
                <a href="{{route('post.show', $post->id)}}>
                <p class="blog-post-category">$post->category->title</p>
                <form action="#">
                    <button type="submit" class="border-0 bg-transparent">
                        <i class="far fa-heart"></i>
                    </button>
                </form>
            </div>
            <a href="{{route('post.show', $post->id)}}" class="blog-post-permalink">
                <h6 class="blog-post-title">$post->title</h6>
            </a>
        </div>
    @endforeach
</div>
```

```

//Отдельный Пост
Route::group(['namespace' => 'App\Http\Controllers\Post', 'prefix' => 'posts'], function()
{
    Route::get('/', 'IndexController')->name('post.index');
    Route::get('/{post}', 'ShowController')->name('post.show');

    //post/{id}/comments
    Route::group(['namespace' => 'Comment', 'prefix' => '/{post}/comments'], function()
    {
        Route::post('/', 'StoreController')->name('post.comment.store');
    });
});

//post/{id}/likes
Route::group(['namespace' => 'Like', 'prefix' => '/{post}/likes'], function()
{
    Route::post('/', 'StoreController')->name('post.like.store');
});

```

Создадим вложенный роутинг

Делаем контроллер для нажатия на сердечко

```

<?php

namespace App\Http\Controllers\Post\Like;

use App\Http\Controllers\Controller;
use App\Models\Post;

class StoreController extends Controller
{
    public function __invoke(Post $post)
    {
        dd($post->id);
    }
}

<form action="{{route('post.like.store', $post->id)}}" method="POST">
    @csrf

```

```

class StoreController extends Controller
{
    public function __invoke(Post $post)
    {
        auth()->user()->likedPosts()->toggle($post->id);
        return redirect()->route(['post.index']);
    }
}

```

Меняем контроллер. Тут появился новый метод, который по сути является переключателем. При первом тыке он вносит в бд значения, а при втором (если знач уже есть) убирает значение из бд

Далее нужно сделать проверку на лайк(поставлен ли он)

```

<form action="{{route('post.like.store', $post->id)}}" method="POST">
    @csrf
    <button type="submit" class="border-0 bg-transparent">
        @auth()
            @if(auth()->user()->likedPosts->contains($post->id))
                <i class="fas fa-heart"></i>
            @else
                <i class="far fa-heart"></i>
            @endif
        @endauth()
    </button>
</form>

```

Данной строчкой мы проверяя есть ли наш пост в списке лайкнутых пользователем

auth()->user()->likedPosts

вернем нам коллекцию постов, а метод contains проверит есть ли в нем наш айди

<i class="fa{{ auth()->user()->likedPosts->contains(\$post->id) ? 's' : 'r' }} fa-heart"><,

Можно вообще сделать вот так

Далее нужно сделать отображение лайков на странице одного поста

```
@if($relatedPosts->count() > 0)
<section class="related-posts">
    <h2 class="section-title mb-4" data-aos="fade-up">Схожие посты</h2>
    <div class="row">
        @foreach($relatedPosts as $relatedPost)
            <div class="col-md-4" data-aos="fade-right" data-aos-delay="100">
                <a href="{{route('post.show', $relatedPost->id)}}>{{$relatedPost->category_id}}</p>
                <a href="{{route('post.show', $relatedPost->id)}}>
                    <h5 class="post-title">{{$relatedPost->title}}</h5>
                </a>
            </div>
        @endforeach
    </div>
@endif
```

Сделаем проверку на схожие посты (просто писать \$relatedPosts нельзя тк проверка на коллекции всегда вернет 1)

Правда ли что Константин чурка, или же это просто вымыслы?

да, правда



Комментарии (6)

артем

Сделаем вот такой значок

```
<div class="row mt-4">
    <div class="col-lg-9 mx-auto">
        <section class="py-3">
            <form action="{{route('post.like.store', $post->id)}}" method="POST"
                @csrf
                <button type="submit" class="border-0 bg-transparent">
                    @auth()
                    @if(auth()->user()->likedPosts->contains($post->id))
                        <i class="fas fa-heart"></i>
                    @else
                        <i class="far fa-heart"></i>
                    @endif
                    @endauth()
                </button>
            </form>
        </section>
        @if($relatedPosts->count() > 0)
```

Но при нажатии кнопки лайка на странице show нас перекинет на главную страницу. Что делать?

```
class StoreController extends Controller
{
    public function __invoke(Post $post)
    {
        auth()->user()->likedPosts()->toggle($post->id);
        return redirect()->back();
    }
}
```

А просто сделаем редирект бэк

Далее добавим кол-во лайков

```

class Post extends Model
{
    use HasFactory;
    use SoftDeletes;
    protected $table = "posts";
    protected $guarded = false;

    protected $withCount = ['LikedUsers'];
}

```

Прописываем в модели пост вот такое св-во

```

<form action="{{route('post.like.store', $post->id)}}" method="POST">
    @csrf
    <span>$post->liked_users_count</span>
    <button type="submit" class="border-0 bg-transparent">

```

И теперь у нас доступно новое св-во liked_users_count

```

<div class="d-flex justify-content-between">
    <p class="blog-post-category">$post->category->title</p>
    @auth()
        <form action="{{route('post.like.store', $post->id)}}" method="POST">
            @csrf
            <span>$post->liked_users_count</span>
            <button type="submit" class="border-0 bg-transparent">
                @if(auth()->user()->likedPosts->contains($post->id))
                    <i class="fas fa-heart"></i>
                @else
                    <i class="far fa-heart"></i>
                @endif
            </button>
        </form>
    @endauth()
    @guest()
        <div>
            <span>$post->liked_users_count</span>
            <a href="{{route('login')}}" style="color:black;"><i class="far fa-heart"></i></a>
        </div>
    @endguest
</div>

```

Реализуем проверку на гостя-юзера