

Оглавление

Тема 1. Создание окружения	2
Тема 2. Разворачивание проекта	4
Тема 3. Работа с Git.....	6
Тема 4. ООП в PHP и в Laravel. Принципы ООП: абстракция, инкапсуляция, наследование, полиморфизм. Пространство имен. Создание абстрактного класса. Интерфейс.	7
Тема 5. Статические методы и свойства. Магические методы.	10
Тема 6. MVC. Структура проекта в Laravel. Папка Public	12
Тема 7. Основные понятия БД. Работа с БД в Laravel. Модель. Миграции.	13
Тема 8. DocBlock в PHP. Типизация в PHP. IDE Helper.....	14
Тема 9. Контроллеры. Создание контроллера и контроллера ресурса.....	15
Тема 10. Post и get запросы в PHP. Маршрутизация в Laravel. Виды маршрутов. Добавление маршрута.....	16
Тема 11. Шаблонизатор blade. Создание шаблона. Наследование шаблона. Структура шаблона. Laravel UI.....	17
Тема 12. Использование библиотеки Laravel-admin.	19

Тема 1. Создание окружения

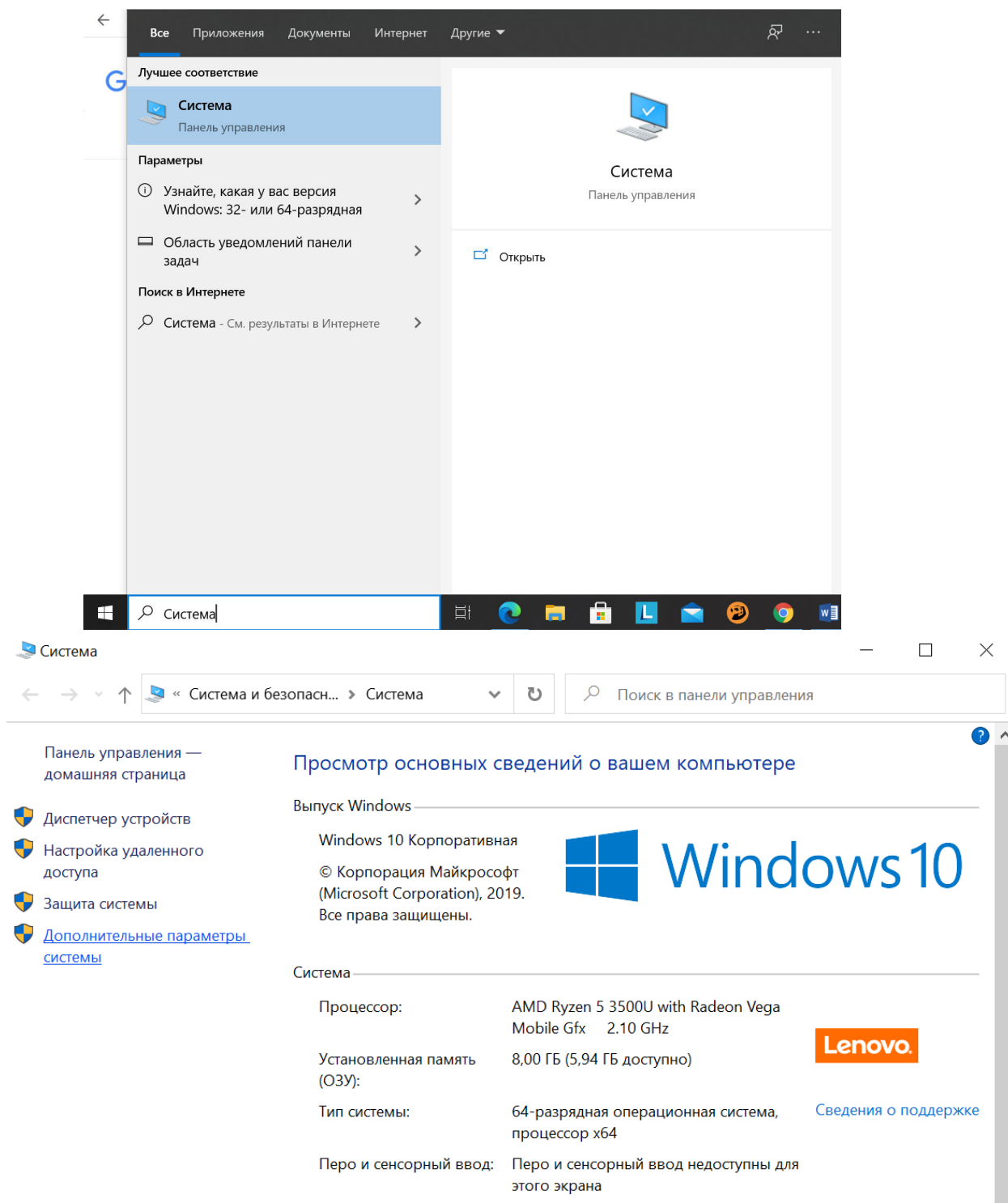
1.1 Установка IDE (PhpStorm или Visual Studio Code)

Visual Studio Code – бесплатный редактор.

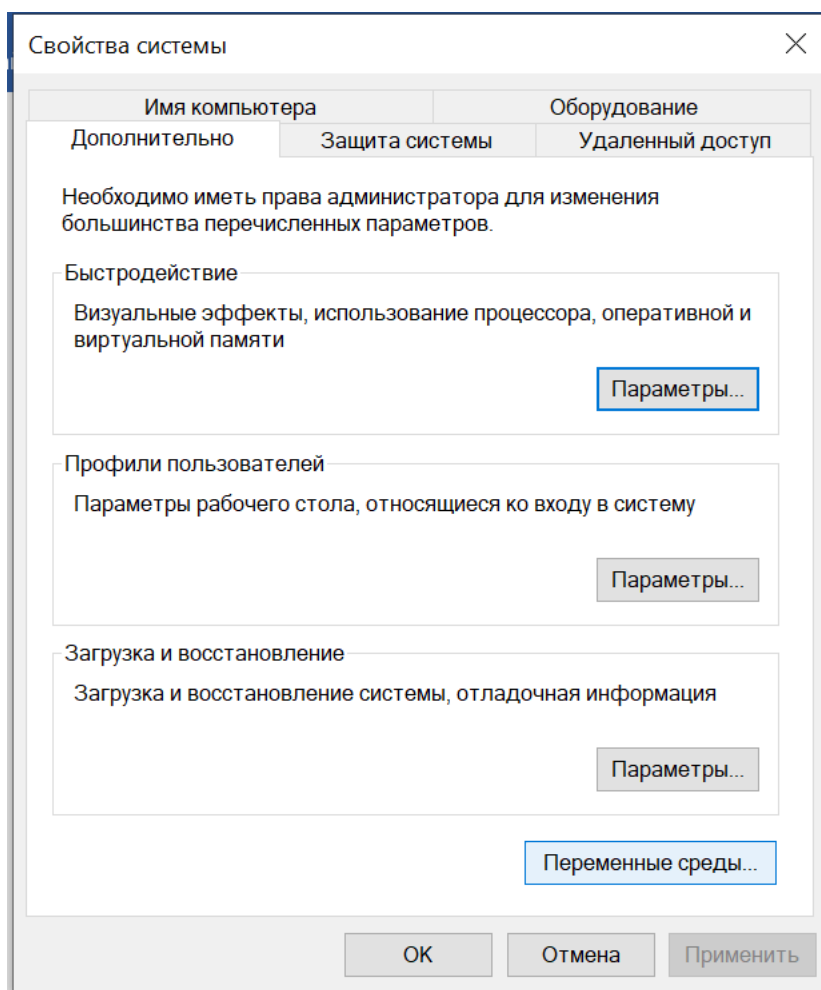
PhpStorm - возможно получение бесплатной студенческой лицензии.

1.2 Установка локального сервера (XAMPP)

1.3 Для Windows добавить путь к папке с php в переменные окружения



Перейти по ссылке «Дополнительные параметры системы».

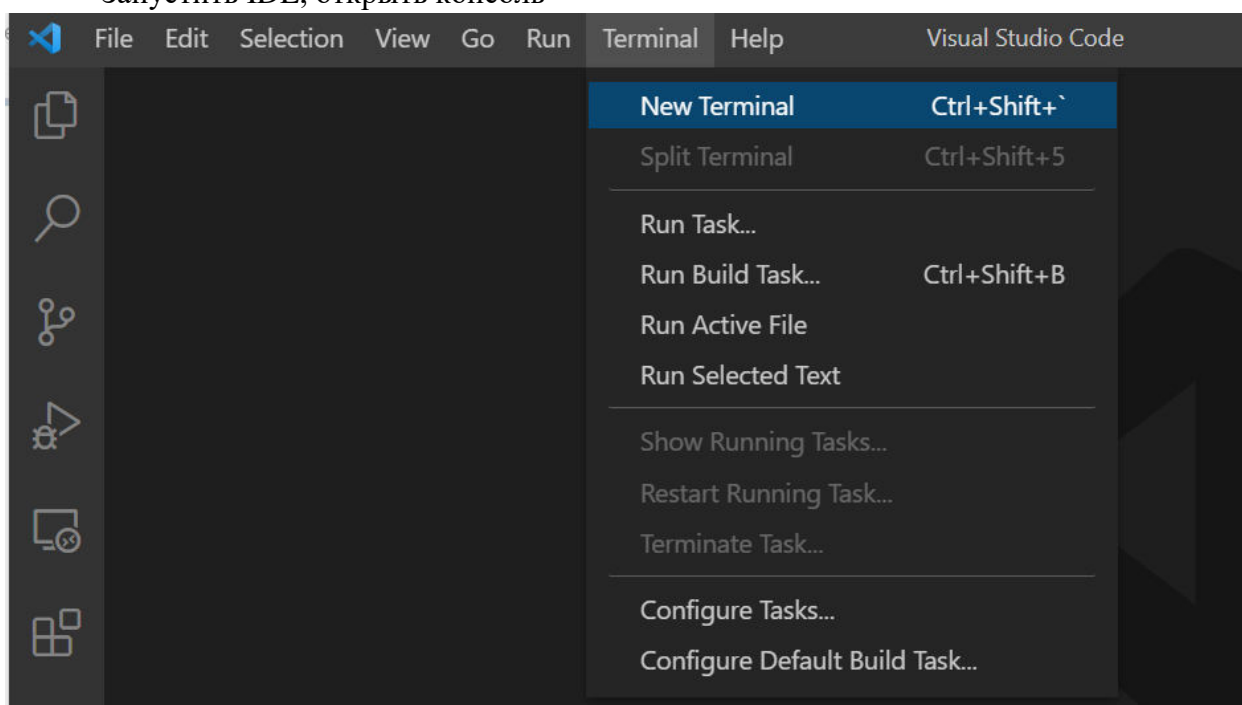


Нажать Переменные среды

Изменить переменную среды Path, добавить путь к папке с php (в папке хатрр).

1.4 Проверка версии php

Запустить IDE, открыть консоль



В консоли ввести: **php --version**

1.5 Добавление проекта в XAMPP

В папке хампп перейти в директорию htdocs, создать папку laravel. Перейти в созданную папку laravel, в ней создать папку проекта – project.local (project – заменить на свое название проекта латинскими буквами).

1.6 Установить пакетный менеджер composer с сайта getcomposer.org/download/

Command-line installation

To quickly install Composer in the current directory, run the following script in your terminal. To automate the installation, use [the guide on installing Composer programmatically](#).

```
php -r "copy('https://getcomposer.org/installer', 'composer-setup.php');"
php -r "if (hash_file('sha384', 'composer-setup.php') === '756890a4488ce9024fc62c56153228907f1545c228516cbf63f8
php composer-setup.php
php -r "unlink('composer-setup.php');"
```

В консоли перейти в папку проекта, используя команду cd. Последовательно ввести команды из списка.

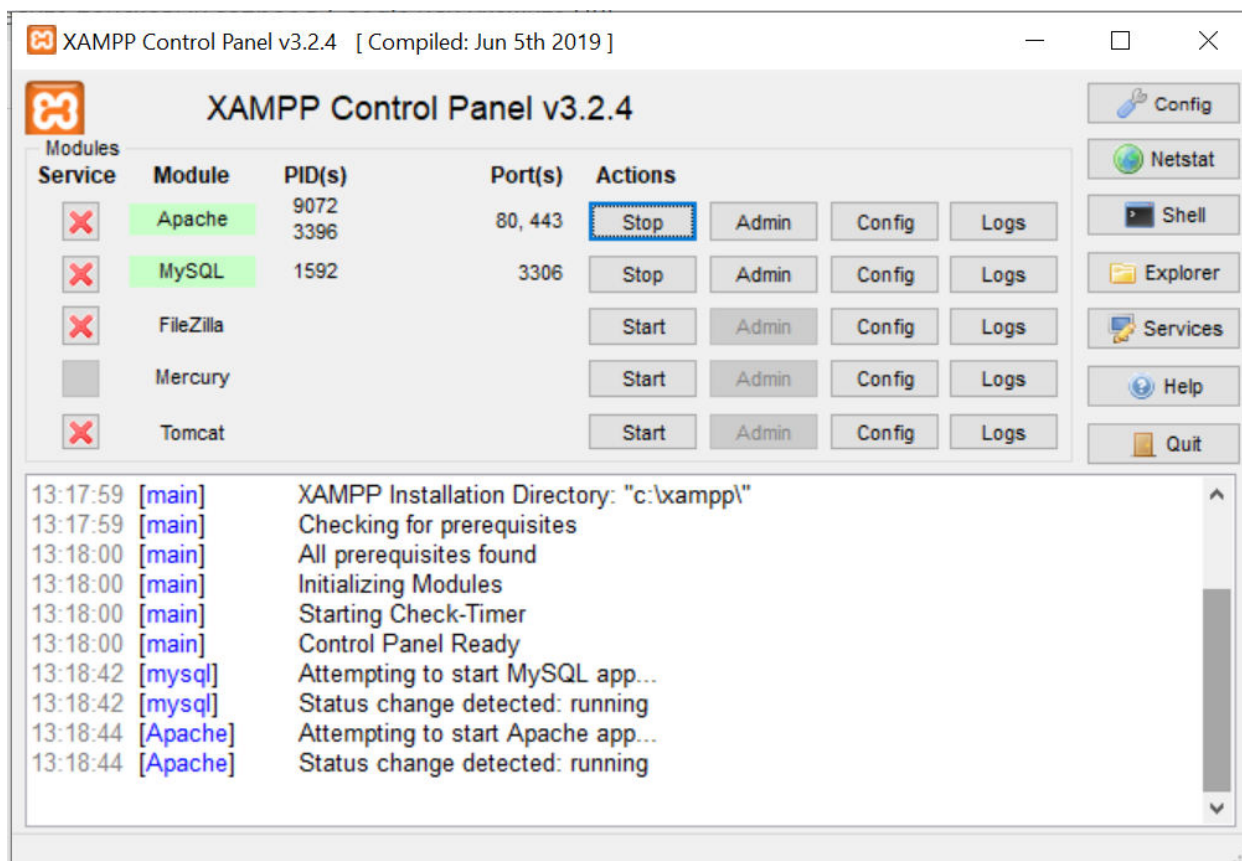
1.7 Работа с пакетным менеджером composer.

В папке проекта найти файл composer.phar. Проверить локальную установку пакетного менеджера – в консоли ввести: **php composer.phar --version**

Тема 2. Разворачивание проекта

2.1 Работа с phpMyAdmin

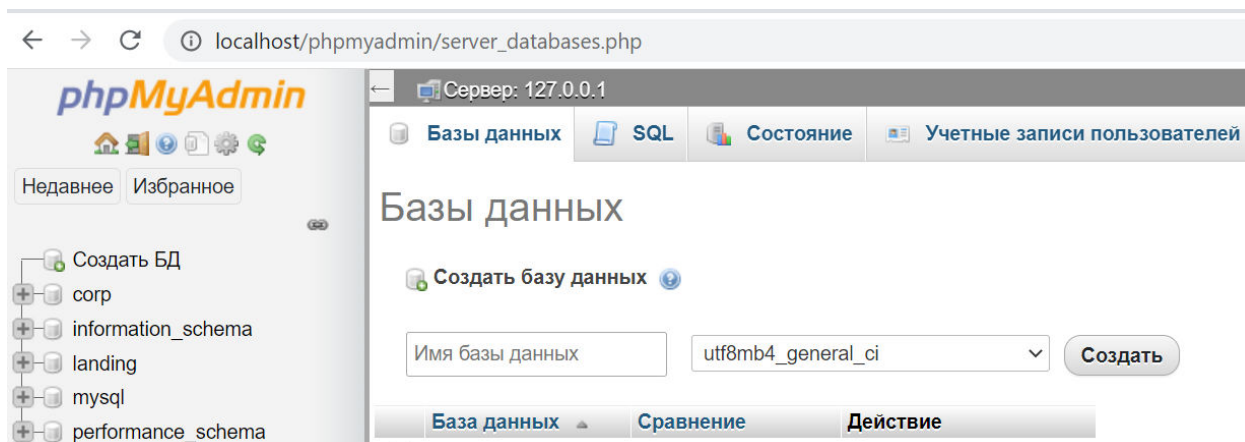
Открыть панель управления XAMPP. Запустить сервер Apache и MySQL.



Перейти по ссылке <http://localhost/phpmyadmin/>.

2.2 Добавление БД

Используя локальный сервер ХАММР и phpMyAdmin создать новую базу данных



Использовать кодировку utf8mb4_general_ci. Имя базы данных может содержать латинские буквы, символы подчеркивания. Перезапустить сервер.

2.3 Настройка проекта в ХАММР – файл с виртуальными хостами

Открыть файл с виртуальными хостами

xampp/apache/conf/extra/httpd-vhosts.conf

Добавить настройки для своего проекта (project – заменить на свое название проекта)

```

2 <VirtualHost *:80>
3     ##ServerAdmin webmaster@dummy-host2.example.com
4     DocumentRoot "C:/xampp/htdocs/laravel/project.local/public"
5     ServerName project.local
6     ServerAlias www.project.local
7
8     <Directory /xampp/htdocs/project.local/public>
9         Options Indexes FollowsymbLinks MultiViews
10        AllowOverride All
11        Order Allow,Deny
12        Allow from all
13    </Directory>
14
15    ErrorLog "logs/project-error.log"
16    CustomLog "logs/project-access.log" common
17 </VirtualHost>

```

2.4 Настройка хоста в Windows – файл Windows/System32/drivers/etc/hosts

В файл

Windows/System32/drivers/etc/hosts

добавить строчку

127.0.0.1 project.local // здесь должно быть имя проекта, которое было указано в ServerName в виртуальном хосте

```

14 # For example:
15 #
16 #      102.54.94.97      rhino.acme.com      # source server
17 #      38.25.63.10      x.acme.com          # x client host
18
19 # localhost name resolution is handled within DNS itself.
20 #   127.0.0.1          localhost
21 #   ::1                localhost
22 127.0.0.1          project.local

```

2.5 Папка public/index.html

В папке с проектом создать папку public, в нее добавить файл index.html с любым тестовым сообщением. Перезапустить сервер. В браузере запустить проект (Например: <http://project.local/> вместо “project” – свое название проекта). Проверить вывод тестового сообщения в браузере.

2.6 Выбор версии Laravel, разворачивание проекта через composer

В консоли перейти в папку проекта и выполнить команду

php composer.phar create-project --prefer-dist laravel/laravel:^9.0

Версия laravel

Перенести содержимое папки laravel в папку проекта на один уровень с папкой public. Пустую папку laravel удалить из проекта.

Открыть папку проекта через IDE.

2.7 Изучить структуру файла composer.json, выбрать версию php, используя тильду (~).

2.8 Файл .env – локальные настройки проекта

В файле .env задать локальные настройки проекта

```

DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=db_name //имя базы данных
DB_USERNAME=root
DB_PASSWORD=

```

2.9 Обновить страницу проекта в браузере: Вывод заставки “Laravel”

Тема 3. Работа с Git

3.1 Установить Git.

Проверить установку – в консоли ввести команду **git --version**

3.2 Создание Git-репозитория

Создать аккаунт на одном из ресурсов: github.com, gitlab.com, bitbucket.org.

После авторизации под своим аккаунтом создать репозиторий.

3.3 Связь проекта с удаленным репозиторием

git remote add origin *ссылка на созданный репозиторий*

3.4 Первый коммит

Проверить изменения в папке проекта – ввести в командной строке **git status**

Зафиксировать изменения

git add .

Первый commit

git commit -m “First commit”

Отправить изменения в удаленный репозиторий

git push -u origin master

3.5 Ветвление в Git

Создать новую ветку develop. Использовать команду

git checkout -b develop

Проверить переключение на ветку develop через **git status**

От ветки develop отвести ветку feature для добавления нового функционала в проект.

3.7 Файл .gitignore. Изучить содержимое файла .gitignore. Понять назначение этого файла.

Тема 4. ООП в PHP и в Laravel. Принципы ООП: абстракция, инкапсуляция, наследование, полиморфизм. Пространство имен. Создание абстрактного класса. Интерфейс.

ООП (объект-ориентированное программирование) – подход в программировании, в котором весь код представляется в виде объектов, каждый из которых является экземпляром определенного класса.

Принципы ООП:

- абстракция
- инкапсуляция
- наследование
- полиморфизм

Абстракция.

Создание концептов (классов), содержащих минимальный набор свойств, достаточных для создания реализации (объект)

```
class Person {
    //Объявление переменных класса
    private string $name;
    private int $age;
    private string $country;

    //Инициализация объекта класса
    public function __construct(string $name, int $age, string $country)
    {
        $this->name = $name; //Обращение к свойству класса и передача значения
        $this->age = $age;
        $this->country = $country;
    }

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

//Создание экземпляра класса
$person = new Person('Alex', 20, 'Russia');
```

Инкапсуляция

Разграничение доступа к свойствам и методам класса, что обеспечивает сокрытие реализации.

Модификаторы доступа:

- *public* – доступ в общем контексте
- *protected* – доступ в контексте класса и его наследников
- *private* – доступ только в контексте класса

А также размещение общего функционала внутри единой сущности.

Наследование

Определение нового класса, на основе уже имеющегося, с заимствованием функционала


```
class Figure
{
    protected string $name;

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

    public function getSquare(): float|int
    {
        return 0;
    }
}
```

```
class Rectangle extends Figure
{
    protected string $name = "rectangle";

    protected int|float $width = 5;
    protected int|float $length = 10;

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

    public function getSquare(): float|int
    {
        return $this->length * $this->width;
    }
}
```

Полиморфизм

Использование объектов с одинаковым интерфейсом, без информации о структуре объекта. Реализуется через:

- интерфейсы
- абстрактные классы

Интерфейсы содержат только описание методов (нет свойств и реализации методов)

```
interface FigureInterface
{
    public function getName();
}
```

```
interface FigureMathInterface
{
    public function getSquare();
}
```

```
class Figure implements FigureInterface, FigureMathInterface
{
    protected $name;

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

    public function getSquare()
    {
        return 0;
    }
}
```

Абстрактные классы

- нельзя создавать объект от абстрактного класса
- может наследоваться от других классов
- может реализовывать интерфейсы
- класс должен быть определён как абстрактный, если содержит хотя бы один абстрактный метод
- абстрактные методы - методы без реализации (как в интерфейсах)

```

abstract class Figure
{
    protected $name;

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

    abstract public function getSquare();
}

```

Namespace – пространство имен, используется для:

- смыслового разграничения функций и классов
- устранения конфликтов функций или классов с одинаковыми именами
- сокращения названия функций или классов

```

namespace App\Http\Controllers;

use Illuminate\Foundation\Auth\Access\AuthorizesRequests;
use Illuminate\Foundation\Bus\DispatchesJobs;
use Illuminate\Foundation\Validation\ValidatesRequests;
use Illuminate\Routing\Controller as BaseController;

class Controller extends BaseController
{
    use AuthorizesRequests, DispatchesJobs, ValidatesRequests;
}

```

Ключевое слово **use** используется для доступа к переменным и методам другого класса.

Реализация в Laravel

<https://laravel.com/docs/11.x/controllers>

<https://laravel.su/docs/11.x/controllers>

Задание

Создать контроллер, который описывает абстрактный класс BaseController. Этот класс BaseController должен наследовать класс Controller.

Записать изменения на git в созданную ветку feature.

Тема 5. Статические методы и свойства. Магические методы.

Статические методы и свойства

- позволяют обращаться к переменной класса без создания экземпляра класса
- доступны в контексте класса, но можно использовать и в контексте объекта
- ключевое слово **self** для доступа к статике внутри класса

```

class Circle extends Figure
{
    const PI = 3.14;
    public $name = 'circle';
    public static $type = 'geometry';
    public $radius = 0;

    public function getSquare()
    {
        return self::PI * ($this->radius**2);
    }

    public static function create($radius)
    {
        $instance = new self;
        $instance->radius = $radius;

        return $instance;
    }
}

$circle = Circle::create(2);
echo Circle::type;

```

Магические методы

Определяют поведения класса и объекта в определенных ситуациях:

- 1) создание/уничтожение объекта
- 2) вызов несуществующих свойств или методов
- 3) сериализация и десериализация
- 4) преобразование в строку
- 5) клонирование
- 6) вызов объекта как функции

```

class A {
    protected $b

    public function __construct(B $b)
    {
        $this->b = $b;
    }

    public function foo()
    {
        $this->someMethod();
    }
}

```

Вызов `__construct()` при создании объекта
 Использование для внедрения зависимостей
 Здесь класс **A** использует переменную класса **B**, которая передается через конструктор (**B \$b**).

Статические методы и свойства в Laravel

Пример использования статических методов для класса **Route** в файле `web.php`

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

Route::group(['namespace' => 'App\Http\Controllers\Blog',
'prefix' => 'blog'], function() {
    Route::resource('posts', 'PostController')->names('blog.
posts');
});

Route::resource('rest', RestTestController::class);

Auth::routes();

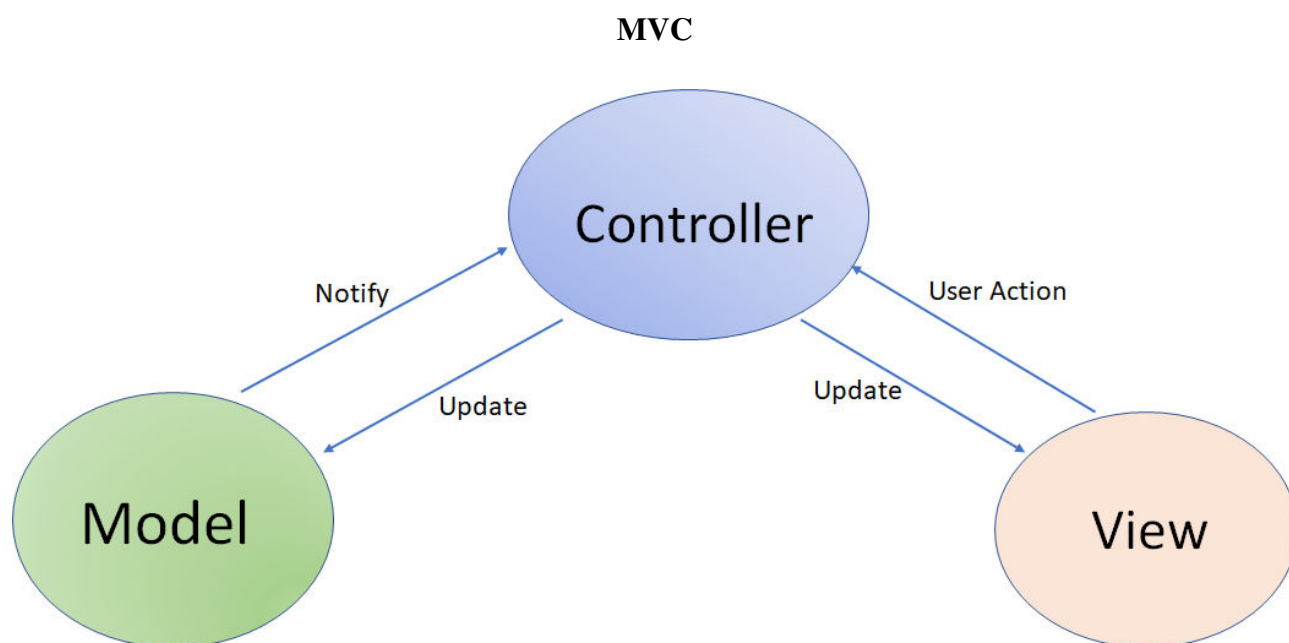
Route::get('/home',
[App\Http\Controllers\HomeController::class, 'index'])->name
('home');
```

Тема 6. MVC. Структура проекта в Laravel. Папка Public

MVC (Model View Controller)

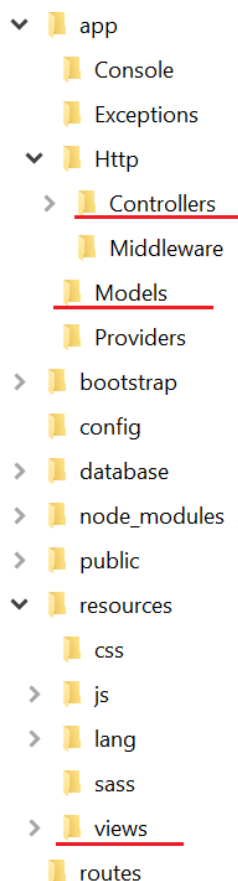
Цель – разделить весь код на 3 основных составляющих.

Чтобы эти составляющие как можно меньше влияли друг на друга.



Структура фреймворка Laravel

- **Модель (Model)** — это База Данных, объекты на которые проецируется БД, методы для работы с БД (папка app/Models)
- **Контроллер (Controller)** – получает запросы от пользователя, обрабатывает их, извлекает данные из БД и отправляет во View (папка app/Http/Controllers)
- **Представление (View)** – это html, шаблоны (папка resources/views)



Папка **public** – запускается файл index.php, поступают запросы, содержит картинки, js, css, которые будут доступны пользователям

.htaccess - этот файл содержит правила, по которым перенаправляет запросы на фреймворк

routes – правила по которым Laravel перенаправляет запросы в контроллеры (routes/web.php)

Задание

В папку app/ Models добавить папку для хранения моделей предметной области проекта.

Тема 7. Основные понятия БД. Работа с БД в Laravel. Модель. Миграции.

Основные понятия БД: поле, запись, первичный ключ, внешний ключ, индекс, таблица, нормальные формы, связи между таблицами.

Модель в Laravel представляет собой класс, который описывает определенную сущность.

<https://laravel.com/docs/11.x/eloquent#defining-models>

<https://laravel.su/docs/11.x/eloquent#generaciia-klassev-modeli>

Пример:

```
namespace App\Models;

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

class BlogPost extends Model
{
    use HasFactory;
    use SoftDeletes;
}
```

Создание модели

Задание:

Создать модель для сущности Новости (News)

php artisan make:model News -m

Флаг **-m** указывает на создание миграции.

Models – папка в каталоге app в которую будет добавлена модель.

В результате создается модель в папке app/Models и файл миграции для таблицы news в папке database/migrations.

Миграции используются, чтобы без потерь обновить структуру базы данных при изменении модели.

<https://laravel.com/docs/11.x/migrations>

<https://laravel.su/docs/11.x/migrations>

Файлы миграций содержат информацию, какие поля нужно создать.

С помощью миграций можно создать таблицу и изменить структуру таблиц - функция up(). Или откатить изменения – функция down().

Задание:

Добавить поля для таблицы news в файле миграций по примеру таблицы users.

Поля таблицы: id, title, preview, content, is_active, published_at, используются timestamps() и softDeletes(). Для полей выбрать подходящие типы данных

<https://laravel.com/docs/11.x/migrations#columns>

<https://laravel.su/docs/11.x/migrations#columns>

Выполнить миграции

php artisan migrate

Заполнить данные в таблице news. (Через phpMyAdmin).

Тема 8. DocBlock в PHP. Типизация в PHP. IDE Helper.

DocBlock в PHP

Для описания переменных класса News необходимо добавить в файл модели многострочный комментарий — DocBlock, который должен быть составлен по определенным [правилам](#).

Для генерации описания может использоваться пакет [Laravel IDE Helper Generator](#).

Необходимо установить пакет по инструкции, использовать локально установленный composer (через composer.phar):

php composer.phar require --dev barryvdh/laravel-ide-helper

После установки использовать как описано в разделе документации «Automatic PHPDocs for models», затем при необходимости скорректировать DocBlock, указать для переменных типы данных.

Тема 9. Контроллеры. Создание контроллера и контроллера ресурса.

Создание контроллера в Laravel

<https://laravel.com/docs/11.x/controllers>

<https://laravel.su/docs/11.x/controllers>

Создание контроллера

php artisan make:controller Blog/BaseController

Создание контроллера ресурса

php artisan make:controller Blog/PostController --resource

Контроллер ресурса будет содержать метод для каждой доступной операции с ресурсами.

Действия, обрабатываемые контроллером ресурсов

Тип	URI	Действие	Имя маршрута
GET	/blog/posts	index	blog.posts.index
GET	/blog/posts/create	create	blog.posts.create
POST	/blog/posts	store	blog.posts.store
GET	/blog/posts/{post}	show	blog.posts.show
GET	/blog/posts/{post}/edit	edit	blog.posts.edit
PUT/PATCH	/blog/posts/{post}	update	blog.posts.update
DELETE	/blog/posts/{post}	destroy	blog.posts.destroy

Задание

Добавить контроллер ресурса для обработки вывода Новостей – NewsController. NewController должен наследовать созданный ранее BaseController.

Заполнить методы **index()** – для вывода списка новостей и **show()** – для отдельной новости.

Ниже пример метода контроллера.

Пример метода для вывода списка постов блога.

```
public function index()
{
    $items = BlogPost::all();

    return view('blog.posts.index', compact('items'));
}
```

Здесь выбираются все поля и все записи сущности BlogPost. Затем переменная с данными передается в шаблон, который находится в файле `resources/views/blog/posts/index.blade.php`.

Для доступа к данным модели необходимо использовать ключевое слово **use**

Пример

```
use App\Models\BlogCategory;
```

Для выборки и вывода данных использовать методы **findOrFail()** для метода **show()**, **all()**, **select()** и другие из раздела <https://laravel.com/docs/11.x/eloquent>,

<https://laravel.su/docs/11.x/eloquent>

Пример.

```
/**
 * Show the form for editing the specified resource.
 *
 * @param int $id
 * @return \Illuminate\Http\Response
 */
public function edit($id)
{
    $item = BlogCategory::findOrFail($id);
    $categoryList = BlogCategory::all();

    return view('blog.admin.category.edit',
        compact('item', 'categoryList'));
}
```

Тема 10. Post и get запросы в PHP. Маршрутизация в Laravel. Виды маршрутов. Добавление маршрута.

GET и POST запросы в php

GET запрос

- запрос может только получать ресурсы
- параметры запроса видны пользователю в адресной строке
- параметры могут быть ограничены адресной строкой

POST запрос

- запрос может менять ресурсы
- параметры запроса явно не видны пользователю
- параметры не ограничены

Виды маршрутов

- **get**
- **post**
- **put**
- **patch**
- **delete**
- **options**

Пример создания маршрута:

```
use App\Http\Controllers\HomeController;
```

```
Route::get('/home', [HomeController::class, 'index'])->name('home');
```

Пример маршрута для контроллера ресурса

```
Route::group(['namespace' => '\App\Http\Controllers\Blog', 'prefix' => 'blog'], function() {
    Route::resource('posts', 'PostController')->names('blog.posts');
});
```

Этот маршрут описывает переход по ссылке /blog/posts относительно корня.

Параметры для группы маршрутов задаются через group().

Этот маршрут нужно добавить в файл routes/web.php

Задание

Добавить маршрут для контроллера новостей – NewsController.

<https://laravel.com/docs/11.x/routing>

<https://laravel.com/docs/11.x/controllers#resource-controllers>

Тема 11. Шаблонизатор blade. Создание шаблона. Наследование шаблона. Структура шаблона. Laravel UI.

Представление (View) в Laravel основано на использовании шаблонизатора Blade - <https://laravel.com/docs/11.x/blade>

<https://laravel.su/docs/11.x/blade>

Директивы шаблонизатора

Директива @section определяет содержимое секции,

@yield используется для отображения содержимого данной секции

Наследование шаблона

@extends('app') показывает какой шаблон наследуется

@section('content') определяет часть шаблона, заданную в области @yield('content') шаблона.

Подключение стилей

```
@section('css')
```

```
<link href="{{ asset('/css/app.css') }}" rel="stylesheet">
```

```
@endsection
```

asset(...) превращает относительные пути в абсолютные.

Пример минимальной структуры родительского шаблона

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">

  <title>@section('title') Имя проекта @show</title>
</head>
<body>
  <main class="content">
    @yield('content')
  </main>
</body>
</html>
```

Директивы дочернего шаблона

@parent используется для дополнения, а не перезаписи содержимого соответствующей области родительского макета.

Пример дочернего шаблона

```
@extends('layouts.app')

@section('content')
  <table>
    @foreach($items as $item)
      <tr>
        <td>{{ $item->id }}</td>
        <td>{{ $item->title }}</td>
        <td>{{ $item->created_at }}</td>
      </tr>
    @endforeach
  </table>
@endsection
```

Пример вывода данных из шаблона resources/views/welcome.blade.php

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

Laravel UI

Для генерации [базового](#) родительского шаблона может использоваться пакет [Laravel UI](#).

Установка пакета

php composer.phar require laravel/ui

php artisan ui bootstrap --auth

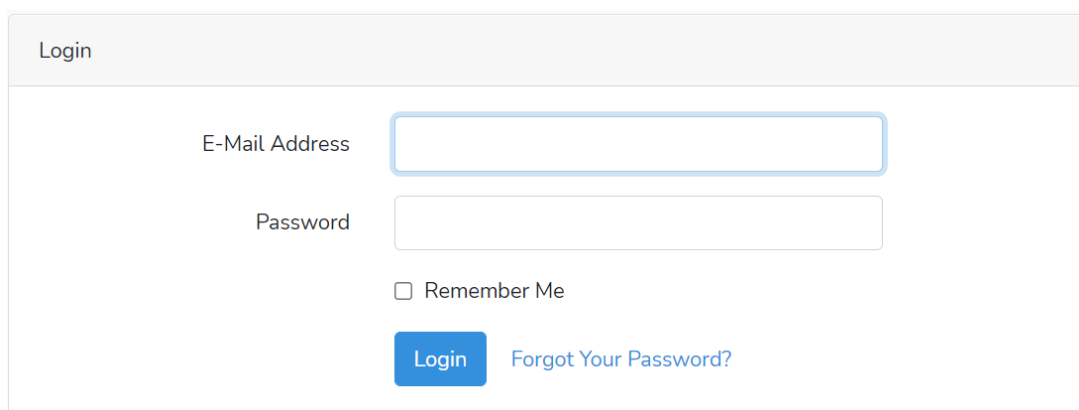
Для подключения встроенных стилей библиотеки bootstrap и сборки проекта необходимо скачать и установить node.js. После установки в консоли выполнить команды

npm install

npm run dev

После выполнения сборки проекта проверить генерацию базового шаблона `resources/views/layouts/app.blade.php`.

В браузере на стартовой странице проекта перейти по ссылке Login и проверить внешний вид окна авторизации с подключенными стилями.



На основе базового шаблона `resources/views/layouts/app.blade.php` создать шаблон для вывода списка новостей `resources/views/news/index.blade.php` и шаблон для вывода страницы отдельной новости `resources/views/news/detail.blade.php`. Для оформления могут быть использованы как встроенные классы библиотеки bootstrap, так и подключаться дополнительный стилевые css файлы (обычно в папке `public/assets`).

В шаблоне списка новостей `resources/views/news/index.blade.php` необходимо указать ссылку для перехода на отдельную новость. Пример реализации ссылки

```
<a href="{ { route('blog.admin.posts.edit', $post->id) } }">{{ $post->title }}</a>
```

Скорректировать методы контроллера `NewsController` для вывода переменных с использованием созданных шаблонов.

Тема 12. Использование библиотеки `Laravel-admin`.

Для реализации админки приложения с возможностью ввода и редактирования данных может использоваться библиотека [Laravel-admin](#).

После установки библиотеки сгенерировать контроллер для управления сущностью Новости

php artisan admin:make NewsController --model=App\Models\News

или для Windows

php artisan admin:make NewsController --model=App\Models\News

Добавить маршрут в app/Admin/routes.php

\$router->resource('news', NewsController::class);

Добавить пункт в меню <https://laravel-admin.org/docs/en/quick-start>

Отредактировать созданный контроллер app/Admin/Controllers/NewsController:

- задать вывод полей в списке новостей в админке в методе Grid(), указать только те поля, которые необходимы в списке: id, title, is_active, created_at, updated_at, deleted_at. <https://laravel-admin.org/docs/en/model-grid>
- задать вывод полей для детального просмотра новости в админке в методе detail() <https://laravel-admin.org/docs/en/model-show>
- задать вывод полей новости для редактирования в методе form() <https://laravel-admin.org/docs/en/model-form>

Проверить возможность ввода, просмотра и редактирования Новостей в админке.

Проверить вывод новостей по ссылке /news относительно корня. Реализовать вывод списка из трех последних новостей на Главной странице.