**1. Создание проекта**

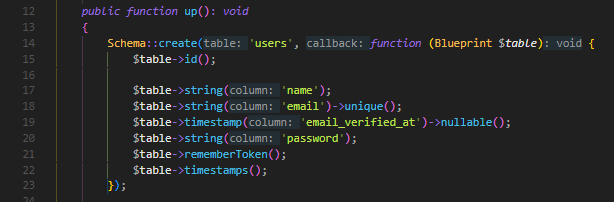
Создаём пустую папку, заходим в неё через VS Code и в терминал выполняем следующую команду:

composer create-project laravel/laravel .

**2. Изменение миграции пользователя**

Создадим для таблицы пользователей данные по типу «логин», «роль», «заблокирован акк?», «количество неправильных попыток», «последнее время логина», «должен изменить пароль», чтобы было куда хранить эту инфу.

Переходим по пути (‘database/mirgations’) и открываем миграцию с названием users. Найдите следующий кусок кода:



Вот в этот кусок кода вставляем следующие строки:

$table->string('login')->unique(); *// логин*

$table->string('role')->default('user'); *// роль*

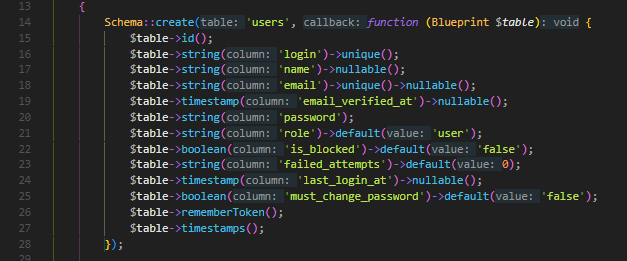
$table->boolean('is\_blocked')->default(false); *// заблокирован*

$table->integer('failed\_attempts')->default(0); *// неудачные попытки*

$table->timestamp('last\_login\_at')->nullable(); *// последний вход*

$table->boolean('must\_change\_password')->default(false); *// смена пароля*

В конечном итоге всё должно выглядеть вот так:



ДЛЯ ПОЛЕЙ NAME И EMAIL УКАЗЫВАЕМ ОБЯЗАТЕЛЬНО АТРИБУТЫ NULLABLE (17, 18 СТРОКИ)

Для конечного итога в терминал вводим две команды для обновления наших миграций:

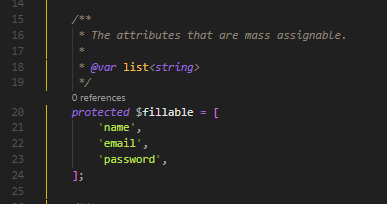
php artisan migrate:fresh

php artisan migrate

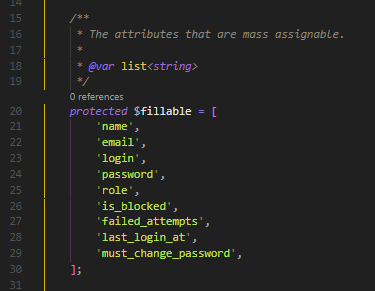
**3. Создание модели и контроллеров**

Модель нужна, чтобы миграции, которые мы сохранили и наша форма взаимодействовали. Переходим в файл «User.php» по пути (‘app/Models’).

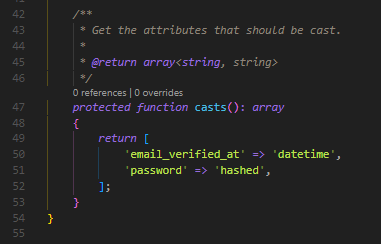
Находим следующие строки:



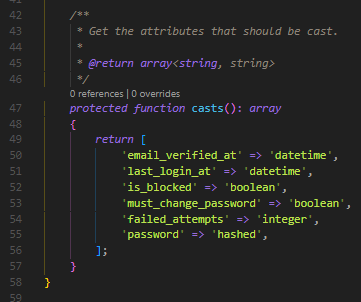
После этого там добавляем то же самое, что мы добавляли в миграции, но тока названия:



Спускаемся ниже, видим следующие строки:



Делаем то же самое, но уже потом через равно пишем тот тип данных, которые мы писали в миграциях и наша модель готова, после этого сохраняем файл:



Теперь создадим контроллер аутентификации пользователей. Контроллер – это вещь, в которой мы можем создать «правила» для отображения определённых данных из нашей БД или, в нашем случае, для отображения ошибок при вводе данных.

В терминале выполним команду для его создания:

php artisan make:controller AuthController

Переходим по пути (‘app/Http/Controllers’) и заходим в созданный файл «AuthController.php».

Вообще сейчас будет много всяких разных строк отборного кала, но они нужны, поэтому нужно всё сделать.

После пятой строки пишем ещё несколько новых:

*use* Illuminate\Support\Facades\Auth;

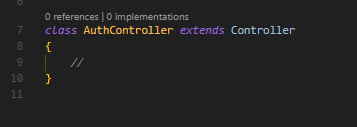
*use* Illuminate\Support\Facades\Hash;

*use* App\Models\User;

*use* Carbon\Carbon;

Первая отвечает за метод авторизации, вторая – за хэширование пароля, третья – за импортирование нашей модели, а четвёртая – не ебу чё такое, но она нужна.

Потом, вместо двух «//» нам нужно будет написать пять функций:



Поехали, первая функция – отображение формы авторизации:

public function loginForm()

{

return view('auth.login');

}

Вторая функция – логин (самая большая):

В ней происходит следующее:

1. Мы задаём логину и паролю свойство обязательного ввода
2. Если пользователь не найден с таким логином, то выводит ошибку
3. Если пользователь заблокирован, то выводит ошибку
4. Если пользователь был неактивен месяц, то его аккаунт блокируется
5. Если пароль неправильно введён, то увеличивается кол-во неудачных попыток. Если кол-во достигает 3 или больше, то статус аккаунта становится заблокированным
6. Если все данные верны, то кол-во неудачных попыток обнуляется, дата последнего логина ставится на текущее время и пользователя сохраняет.
7. После этого пользователя авторизуют и идёт обязательное условие: если он должен сменить пароль сразу после авторизации – его перекидывает на соответстующую страницу, а если не должен, то его перекидывает на дашборд с уведомлением, что он авторизовался

    public function login(Request $request) {

        $request->validate([

            'login' => 'required',

            'password' => 'required',

        ], [

            'login.required' => 'Поле "Логин" обязательно',

            'password.required' => 'Поле "Пароль" обязательно',

        ]);

        $user = User::where('login', $request->login)->first();

        if (

            !$user ||

            $user->is\_blocked ||

            ($user->last\_login\_at && Carbon::parse($user->last\_login\_at)->diffInDays(now()) > 30 && ($user->is\_blocked = true) && $user->save())

        ) {

            return back()->withErrors(['login' => $user && $user->is\_blocked ? 'Вы заблокированы. Обратитесь к администратору' : 'Вы ввели неверный логин или пароль. Пожалуйста, проверьте ещё раз введенные данные'])->withInput();

        }

        if (!Hash::check($request->password, $user->password)) {

            $user->failed\_attempts++;

            if ($user->failed\_attempts >= 3) $user->is\_blocked = true;

            $user->save();

            return back()->withErrors(['login' => 'Вы ввели неверный логин или пароль. Пожалуйста, проверьте ещё раз введенные данные'])->withInput();

        }

        $user->failed\_attempts = 0;

        $user->last\_login\_at = now();

        $user->save();

        Auth::login($user);

        if ($user->role === 'admin') return redirect('/admin/users')->with('success', 'Вы успешно авторизовались как администратор');

        if ($user->must\_change\_password) return redirect()->route('password.change.form');

        if ($user->role === 'user') return redirect('/dashboard')->with('success', 'Вы успешно авторизовались как пользователь');

    }

В них всё написано в комментариях, что за что отвечает.

Третья функция – отображение после логина функции отображения пароля:

*public* *function* changePasswordForm()

{

*return* view('auth.change-password');

}

Четвёртая функция – отвечает за смену пароля:

Здесь происходят следующие действия:

1. Задаются условия для ввода данных (текущий пароль обязателен; новый пароль обязателен, минимум 6 символов и он должен быть подтверждён)
2. Если эти условия нарушаются, то выдаются соответствующие уведомления
3. Если всё хорошо, то пользователя авторизует и дальше идёт работа с его данными.
4. Его пароль через метод хэширования изменяется на тот, который был введён в поле «новый пароль», поле должен изменить пароль – меняется на false.
5. Данные пользователя сохраняются, его перекидывает на дашборд с успешным уведомлением.

public function changePassword(Request $request)

{

$request->validate([

'current\_password' => 'required',

'new\_password' => 'required|min:6|confirmed',

], [

'current\_password.required' => 'Введите текущий пароль',

'new\_password.required' => 'Введите новый пароль',

'new\_password.min' => 'Минимальная длина пароля 6 символов',

'new\_password.confirmed' => 'Пароли не совпадают',

]);

$user = Auth::user();

if (!Hash::check($request->current\_password, $user->password)) {

return back()->withErrors(['current\_password' => 'Текущий пароль неверен'])->withInput();

}

$user->password = Hash::make($request->new\_password);

$user->must\_change\_password = false;

$user->save();

return redirect('/dashboard')->with('success', 'Пароль успешно изменён');

}

Пятая функция – логаут.

public function logout()

{

    Auth::logout();

    return redirect('/')->with('success', 'Вы успешно вышли из системы');

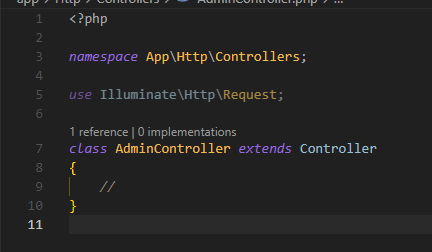
}

Сохраняем всю эту поебень с божьей помощью.

Вводим в терминал:

php artisan make:controller AdminController

Заходим в наш файл (как сделать - страница 8):



Далее, после 5-ой строки добавляем следующие две строки, как в предыдущем контроллере:

*use* Illuminate\Support\Facades\Hash;

*use* App\Models\User;

В нём у нас также есть 5 функций, но они не такие большие.

Первая – отображение страницы списка пользователей со всеми данными пользователя:

public function index()

{

    $users = User::all();

    return view('admin.users', compact('users'));

}

Вторая – это отображение страницы создания пользователей:

public function create()

{

    return view('admin.create-user');

}

Третья – это валидация для создания пользователей:

public function store(Request $request)

{

$request->validate([

'login' => 'required|unique:users,login',

'password' => 'required|min:6',

'role' => 'required|in:admin,user',

], [

'login.required' => 'Логин обязателен',

'login.unique' => 'Пользователь с таким логином уже существует',

'password.required' => 'Пароль обязателен',

'password.min' => 'Минимальная длина пароля 6 символов',

'role.required' => 'Роль обязательна',

]);

User::create([

'login' => $request->login,

'password' => Hash::make($request->password),

'role' => $request->role,

'must\_change\_password' => true,

]);

return redirect()->route('admin.users')->with('success', 'Пользователь успешно добавлен');

}

Четвёртая – это страница с отображением редактирования определённого пользователя:

public function edit($id)

{

    $user = User::findOrFail($id);

    return view('admin.edit-user', compact('user'));

}

Пятая – это процесс обновления данных пользователя:

*public* *function* update(Request $request, $id)

{

    $user = User::findOrFail($id);

    $user->login = $request->input('login');

    $user->role = $request->input('role');

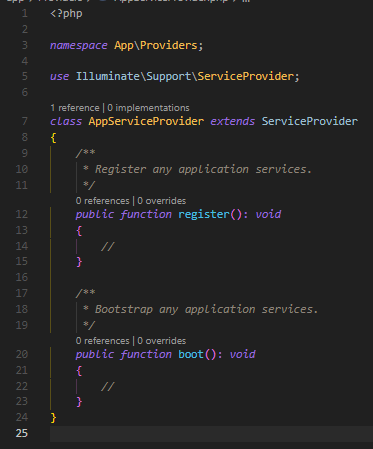
    $user->is\_blocked = $request->has('is\_blocked') ? 1 : 0; *// <-- обработка чекбокса*

    $user->save();

*return* redirect()->route('admin.users')->with('success', 'Пользователь обновлён');

}

Ещё нужно добавить сам утвердитель роли, заходим в файл «AppServiceProvider» по пути (‘app/Http/Providers’). Там будем следующий вид:



На 6-ую строку пишем следующую строчку:

*use* Illuminate\Support\Facades\Gate;

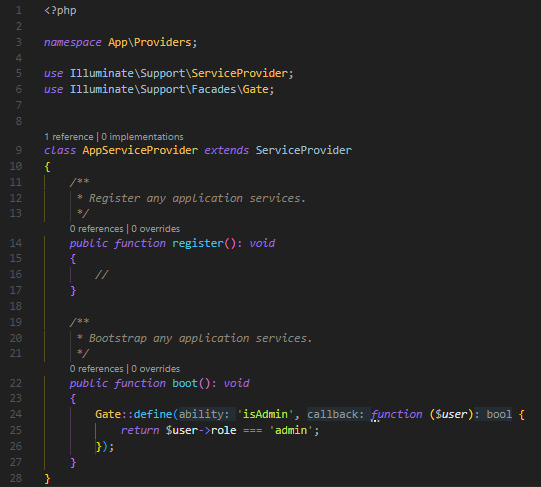
Вместо // на 22-ой строке пишем следующую функцию:

Gate::define('isAdmin', *function* ($*user*) {

*return* $user->role === 'admin';

});

В конечном итоге всё должно выглядеть так:



Чтобы нам добавить админа, мы делаем следующие шаги:

В терминале пишем

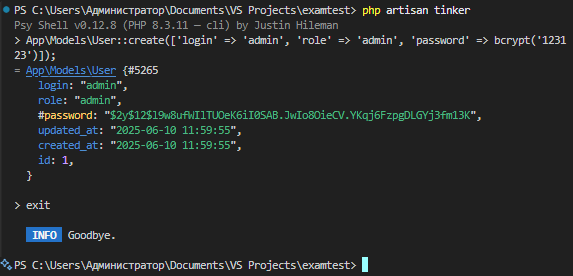
php artisan tinker

Далее как раз команда для создания пользователя:

App\Models\User::create(['login' => 'admin', 'role' => 'admin', 'password' => bcrypt('123123')]);

И выходим из тинкера при помощи команды exit

При удачном раскладе будет следующий результат:



**4. Создание форм**

Перед этим нам нужно создать файл для того, чтобы у нас был общий вид вкладок с единым названием. Потом этот же файл мы будем привязывать к каждой нашей страничке. Заходим по пути (‘resources/views’). Создаём там папку «layouts», а в ней файл «app.blade.php». Заполняем следующим образом:

<!DOCTYPE html>

<html lang="en">

<head>

<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>Модуль 3</title>

</head>

<body>

@yield('content')

*@if* ($errors->any())

        <div>

            <ul>

*@foreach* ($errors->all() as $error)

                    <li>{{ $error }}</li>

*@endforeach*

            </ul>

        </div>

*@endif*

*@if*(session('success'))

        <script>

            alert("{{ session('success') }}");

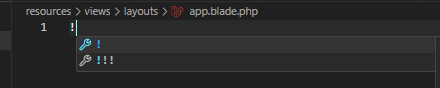
        </script>

*@endif*

</body>

</html>

P.S. Просто в пустом документе вводим восклицательный знак



После такой картины нажимаем на Enter и у нас само всё заполнится.

Единственное, что нужно изменить – это строка с тэгом title (внутри напишите типа «Модуль 3»). И в тэге body единственная строчка — это функция, которая будет вызывать весь контент из других страничек, которые мы дальше будем создавать.

После этого идёт два условия:

1. Если есть ошибки, то они выводятся через лист
2. Если усепшное уведомление, то оно выводится через alert.

Сделано это в целях удобства, чтобы не писать по миллион раз одну и ту же хуйню.

Для того, чтобы у пользователя была возможность залогиниться – нам нужно зайти по пути (‘resources/views’). Там создадим папку «auth» и в ней создадим файл «login.blade.php». Должно всё выглядеть следующим образом:

*@extends*('layouts.app')

*@section*('content')

<h2>Вход</h2>

<form *method*="POST" *action*="{{ route('login') }}">

*@csrf*

    <div>

        <label *for*="login">Логин</label><br>

        <input *type*="text" *id*="login" *name*="login" *required*>

    </div>

    <div>

        <label *for*="password">Пароль</label><br>

        <input *type*="password" *id*="password" *name*="password" *required*>

    </div>

    <div>

        <button *type*="submit">Войти</button>

    </div>

</form>

*@endsection*

P.S. Первая, третья и последняя строка – импорт той странички, которой мы создали перед этим, эти строки будут повторяться в каждых новых страничках.

Далее у нас идёт форма с методом POST (отправка) и действие – путь к login, который мы рассмотрим далее, чтобы наши данные отправлялись по пути.

Внутри метод @csrf - уникальная мера безопасности, разработанная для защиты веб-приложений от несанкционированных или вредоносных запросов. Оставляем эту строчку обязательно.

Далее у нас поля Логин и Пароль с соответстующими тэгами label, input, в которых мы обязательно указываем id, name, type.

Завершает всё кнопка с обязательным атрибутом type со значением submit, иначе нихуя не отправится.

После того, как мы залогинимся – нас должно перекинуть на форму смены пароля.

В этой же папке «auth» создаём файл «change-password.blade.php»:

Здесь всё точно так же идентично, как и в форме авторизации.

*@extends*('layouts.app')

*@section*('content')

<h2>Смена пароля</h2>

<form *method*="POST" *action*="{{ route('password.change') }}">

*@csrf*

    <div>

        <label *for*="current\_password">Текущий пароль</label><br>

        <input *type*="password" *id*="current\_password" *name*="current\_password" *required*>

    </div>

    <div>

        <label *for*="new\_password">Новый пароль</label><br>

        <input *type*="password" *id*="new\_password" *name*="new\_password" *required*>

    </div>

    <div>

        <label *for*="new\_password\_confirmation">Подтвердите новый пароль</label><br>

        <input *type*="password" *id*="new\_password\_confirmation" *name*="new\_password\_confirmation" *required*>

    </div>

    <div>

        <button *type*="submit">Изменить пароль</button>

    </div>

</form>

*@endsection*

Для того, чтобы наш пользователь мог видеть результат авторизации, его должно перебросить на страницу. Этой страницей будет файл «welcome.blade.php» по пути (‘resources/views’).

После того, как мы туда зашли – очищаем файл полностью. В нём будет только 6 строк:

*@extends*('layouts.app')

*@section*('content')

<h1>Добро пожаловать!</h1>

<form *action*="{{ route('logout') }}" *method*="GET">

    <button *type*="submit">Выйти</button>

</form>

Тут только заголовок и форма, чтобы выйти из аккаунта, что очень удобно.

Заходим в папку (‘resources/views’) и там создаём папку «admin», заходим туда и делаем грязь.

Первой страницей у нас будет список пользователей («users.blade.php»):

*@extends*('layouts.app')

*@section*('content')

<h2>Пользователи</h2>

<a *href*="{{ route('admin.users.create') }}">Добавить пользователя</a>

<table border="1">

    <thead>

        <tr>

            <th>ID</th>

            <th>Логин</th>

            <th>Роль</th>

            <th>Статус</th>

            <th>Действия</th>

        </tr>

    </thead>

    <tbody>

*@foreach*($users as $user)

        <tr>

            <td>{{ $user->id }}</td>

            <td>{{ $user->login }}</td>

            <td>{{ $user->role }}</td>

            <td>

*@if*($user->is\_blocked)

                    Заблокирован

*@else*

                    Активен

*@endif*

            </td>

            <td>

                <a *href*="{{ route('admin.users.edit', $user->id) }}">Редактировать</a>

            </td>

        </tr>

*@endforeach*

    </tbody>

</table>

<form *action*="{{ route('logout') }}" *method*="GET">

    <button *type*="submit">Выйти</button>

</form>

*@endsection*

Сделан список в виде таблицы, а в нём показан id, логин, роль (заблокирован или активен), и кнопка «Редактировать». Обёрнута вся таблица после тэга tbody в функцию @foreach($users as $user), которая отвечает за перечисление всех пользователей.

Второй страницей является создание пользователей («create-user.blade.php»):

*@extends*('layouts.app')

*@section*('content')

<h2>Добавить пользователя</h2>

<form *method*="POST" *action*="{{ route('admin.users.store') }}">

*@csrf*

    <div>

        <label *for*="login">Логин</label><br>

        <input *type*="text" *id*="login" *name*="login" *required*>

    </div>

    <div>

        <label *for*="role">Роль</label><br>

        <select *id*="role" *name*="role" *required*>

            <option *value*="user">Пользователь</option>

            <option *value*="admin">Администратор</option>

        </select>

    </div>

    <div>

        <label *for*="password">Пароль</label><br>

        <input *type*="password" *id*="password" *name*="password" *required*>

    </div>

    <div>

        <button *type*="submit">Добавить</button>

    </div>

</form>

*@endsection*

Тут абсолютно такая же песня, как и с авторизацией, ничё нового

И третьей страницей будет являться страница редактирования пользователей («edit-user.blade.php»):

*@extends*('layouts.app')

*@section*('content')

<h2>Редактировать пользователя</h2>

<form *method*="POST" *action*="{{ route('admin.users.update', $user->id) }}">

*@csrf*

    <div>

        <label *for*="login">Логин</label><br>

        <input *type*="text" *id*="login" *name*="login" *value*="{{ old('login', $user->login) }}" *required*>

    </div>

    <div>

        <label *for*="role">Роль</label><br>

        <select *id*="role" *name*="role" *required*>

            <option *value*="user" {{ old('role', $user->role) == 'user' ? 'selected' : '' }}>Пользователь</option>

            <option *value*="admin" {{ old('role', $user->role) == 'admin' ? 'selected' : '' }}>Администратор</option>

        </select>

    </div>

    <div>

        <label>

            <input *type*="checkbox" *name*="is\_blocked" *value*="1" {{ old('is\_blocked', $user->is\_blocked) ? 'checked' : '' }}>

            Заблокировать

        </label>

    </div>

    <div>

        <button *type*="submit">Сохранить</button>

    </div>

</form>

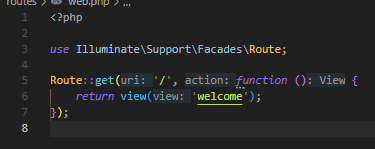
*@endsection*

Тут мы делаем ту же самую форму, НО в конце ставим запятую и дописываем $user->id, потому что обращаемся именно к этому пользователю ёпт.

В этот раз мы можем выдать ему роль через тэги option, и имеем возможность блокировать или разблокировывать пользователя.

**5. Создание маршрутизации**

Перед тем как создавать страницы, нам нужно задать всем страницам пути. Заходим в папку routes в корне проекта, там в файл «web.php»:



После строки 3 вставляем две строки, которые отвечают за подключение наших путей к контроллерам.

use App\Http\Controllers\AuthController;

use App\Http\Controllers\AdminController;

5-7 строки мы стираем (которые на рисунке), вместо них пишем следующие строки:

Route::get('/', function () {

return redirect('/login');

});

Route::get('/login', [AuthController::class, 'loginForm'])->name('login.form');

Route::post('/login', [AuthController::class, 'login'])->name('login');

Route::get('/password/change', [AuthController::class, 'changePasswordForm'])->name('password.change.form');

Route::post('/password/change', [AuthController::class, 'changePassword'])->name('password.change');

Route::get('/logout', [AuthController::class, 'logout'])->name('logout');

Route::get('/dashboard', function() {

return view('welcome');

})->middleware('auth');

Первые три строки отвечают за перенаправление с «/» на login

2 отвечает за страницу login, где мы наблюдаем форму авторизации

3 отвечает за отправку данных со страницы login

4 и 5 соответственно за страницу смены пароля и отпавку данных с неё.

6 строка отвечает за логаут из текущего пользователя.

7-9 отвечают за показ страницы после авторизации и только для авторизованных пользователей.

Во всех строках, кроме последних есть ссылки на контроллеры, но в последних трёх их нет, потому что нет данных, которые нужно отображать из БД.

В тот же самый файл «web.php» заполняем дальше следующие строки:

Route::middleware(['auth'])->group(*function* () {

    Route::middleware(['can:isAdmin'])->group(*function* () {

        Route::get('/admin/users', [AdminController::*class*, 'index'])->name('admin.users');

        Route::get('/admin/users/create', [AdminController::*class*, 'create'])->name('admin.users.create');

        Route::post('/admin/users', [AdminController::*class*, 'store'])->name('admin.users.store');

        Route::get('/admin/users/{id}/edit', [AdminController::*class*, 'edit'])->name('admin.users.edit');

        Route::post('/admin/users/{id}', [AdminController::*class*, 'update'])->name('admin.users.update');

    });

});

1-ая строка отвечает за группировку следующих путей только для авторизованных пользователей (при попытки перейти на них без авторизации перекинет на /login).

2-ая отвечает за роль пользователя в группировке всех путей (если зайдёт обычный чел с ролью user, то его выкинет к хуям).

/admin/users (c методом GET) – это список пользователей.

/admin/users/create – страница создания пользователя

/admin/users/ (с методом POST) – пересылает на страницу создания пользователя

/admin/users/{id}/edit – это страница редактирования определённого пользователя

/admin/users/{id (с методом POST) – это процесс обновления данных определённого пользователя.

Теперь можете запустить проект через:

php artisan serve

в терминале VS Code и посмотреть готовый результат.