installations –таблица установок первоначальных

сколько вопросов - quantity

год выполнения - date

время выполнения - term

процент выполнения – percent

учреждение institutions таблица

questions-таблица

* quest – вопрос
* drawing –рисунок вопрос
* number – номер задания или номер теста
* view – вид вопроса
* answer – ответ если ответ письменный

answers – таблица ответов

* questions\_id - связь с вопросами
* answer – ответ
* right – верный ответ
* ball

users – таблица пользователей

* role - роль
* login
* password
* password\_str
* surname - фамил
* name - имя
* patronymic – отчество
* institution\_id– учреждение
* post –должность

results таблица результатов

* user\_id
* question\_id
* question\_string
* variant номер выполнения
* answer – ответ ученика
* question\_number номер вопроса
* result
* date

result\_answers таблица результатов ответов

* result\_id
* answer\_id
* right –верный
* answer\_user
* string\_answer

php artisan make:model Institution –m

<https://github.com/pam-70/oxtr.git>

git commit -m "relis v2"

git push -u origin main

git init

git add README.md

git commit -m "first commit"

git branch -M main

git remote add origin https://github.com/pam-70/oxtr.git

git push -u origin main

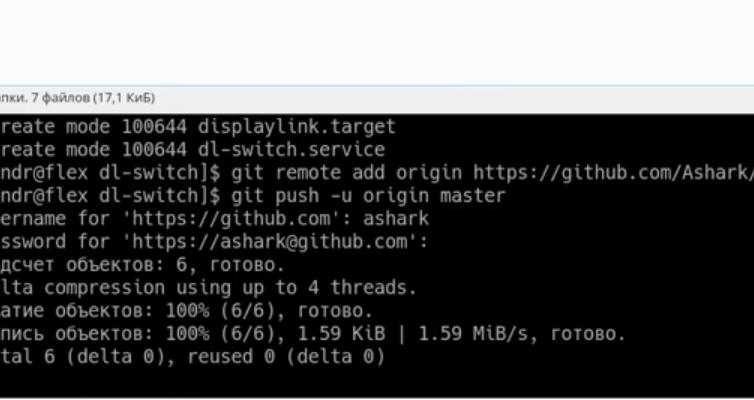
**…or push an existing repository from the command line**

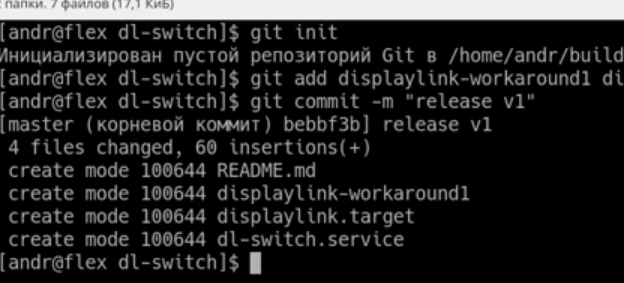
git remote add origin https://github.com/pam-70/oxtr.git

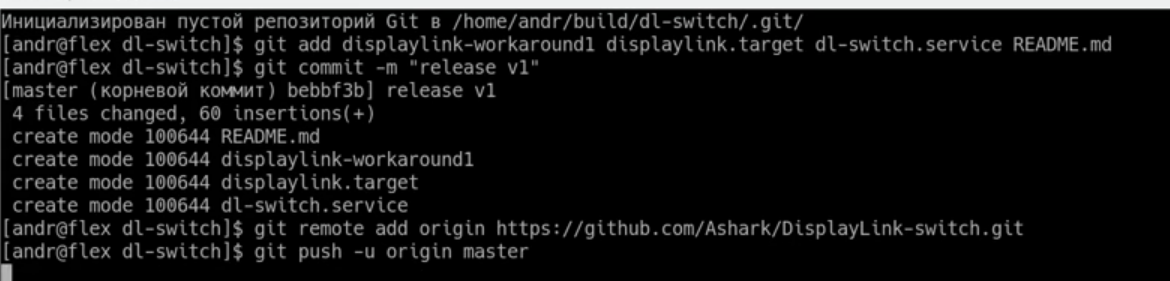
git branch -M main

git push -u origin main

Schema::table('posts', function (Blueprint $table) {  
    $table->integer('user\_id')->unsigned();  
  
    $table->foreign('user\_id')->references('id')->on('users');  
});







Связи помогают нам контролировать таблицы во время работы с базой данных - не сможете вставить значение, отсутствующее в соответствующем столбце связанной родительской таблицы. Так же, не сможете, случайно, удалить строки в родительской таблице если они используются для связи с дочерней таблицей.  
Например, если клиент сделал заказ, то уже не получится удалить его из базы данных просто так.  
  
При связывании таблиц отношением один к одному или один ко многим, нужно сначала определить какая из них является родительской - у нее связь делается по существующему первичному ключу (обычно это поле «id») и какая дочерней - в ней нужно будет создать дополнительное поле для связи с родительской таблицей, например «user\_id» и внешний ключ для этого поля.  
  
Так, например, пользователь может сделать несколько заказов, но заказы не могут быть без пользователя (покупателя). Тут:

* users – родительская таблица;
* customers – дочерняя таблица.

В данном случае, связывающее поле user\_id и внешний ключ для него необходимо создавать в дочерней таблице (customers).  
Рассмотрим особенности работы со связанными таблицами в php-фреймворке Laravel.

#### Пример создания файла миграции в консоли.

Создать пустой файл миграции с названием CreateRoleTables:

php artisan make:migration CreateRoleTables

Создать файл миграции с названием CreateRoleTables и определить в нем создание таблицы roles:

php artisan make:migration CreateRoleTables --create=roles

#### Примеры миграций для создания связывающего поля/таблицы и внешних ключей.

**Миграция – создание связывающего поля с внешним ключом** (связь один к одному и один ко многим).  
Миграция для создания таблицы 'countries' со связывающим полем и внешним ключом:

public function up()

{

Schema::create('countries', function (Blueprint $table) {

$table->increments('id');

$table->string('name');

*//создание поля для связывания с таблицей user*

table->integer('user\_id')->unsigned()->default(1);

*//создание внешнего ключа для поля 'user\_id', который связан с полем id таблицы 'users'*

$table->foreign('user\_id')->references('id')->on('users');

$table->timestamps();

});

}

public function down()

{

Schema::dropIfExists('countries');

}

Имя внешнего ключа должно быть - название владеющей модели (родительской таблицы в единственном числе) плюс \_id (тут 'user\_id') или указывать явно в связывающем методе модели.  
  
  
**Миграция - создания дополнительного, связывающего поля в существующую таблицу с внешним ключо**м.

public function up()

{

Schema::table('posts', function (Blueprint $table) {

$table->integer('user\_id')->unsigned()->default(1);

$table->foreign('user\_id')->references('id')->on('users');

});

}

public function down()

{

Schema::table('posts', function ($table) {

$table->dropForeign('posts\_user\_id\_foreign');

$table->dropColumn('user\_id');

});

}

**Миграция – создание связывающей таблицы (для связи многие ко многим).**  
  
Пример связывания таблиц **roles**и **users**.  
Создаем таблицу **role\_user**. Название ее не случайно - указываются две связываемые таблицы roles и users через нижнее подчеркивание в единичном числе. Или же произвольное название и тогда указывается явно в модели, в связывающем методе **belongsToMany()**в качестве второго аргумента.

public function up()

{

Schema::create('role\_user', function (Blueprint $table) {

$table->increments('id');

$table->integer('user\_id')->unsigned();

$table->foreign('user\_id')->references('id')->on('users');

$table->integer('role\_id')->unsigned();

$table->foreign('role\_id')->references('id')->on('roles');

$table->timestamps();

});

}

public function down()

{

Schema::dropIfExists('role\_user');

}

### Выборка данных, создание связывающих методов моделей.

##### Связь "один к одному".

Например, в таблице **countries**создан внешний ключ для связи с таблицей **users**. В таком случае, таблица users является родительской и нужно создать метод **hasOne()** в классе ее модели **User**, а для модели **Country**создать метод**belongsTo()**.  
  
Создание метода для модели User:

public function country()

{

return $this->hasOne('App\Country');

}

**Eloquent**(реализация шаблона ActiveRecord в Larave) считает, что внешний ключ отношения называется по имени модели. В данном случае предполагается, что это **user\_id**. Если вы хотите перекрыть стандартное имя, передайте второй параметр методу **hasOne()**:

return $this->hasOne('App\Country', 'foreign\_key');

Получение связи (в таблице '**countries**') для пользователя с id=1:

$user = User::find(1);

$country = $user->country;

$name = $country->name; *//Ukraine*

вызываем не метод, а одноименное динамическое свойство.  
  
  
Создание обратной связи (для таблицы с которой связывали).  
Метод модели:

public function user()

{

return $this->belongsTo('App\User');

}

получение данных аналогичное:

$country = Country::find(1);

$user = $country->user;

$name = $user->name; *//Vova*

##### Связь "Один ко многим".

На примере связи «один автор – несколько постов». АВТОР (таблица users) – ПОСТЫ (таблица posts).  
Для этого должен быть создан внешний ключ для таблицы постов. После этого в модели **User**можно использовать метод **hasMany()**, а в модели **Post**метод **belongsTo()**.  
  
Создание метода для модели **User**:

public function posts()

{

return $this->hasMany('App\Post');

}

Получение данных:

$user = User::find(1);

$posts = $user->posts;

foreach ($posts as $post) {

*//*

}

вызываем не метод, а одноименное динамическое свойство.  
  
Создание обратной связи (для таблицы с которой связывали).  
Метод модели **Post**:

public function User()

{

return $this->belongsTo('App\User');

}

получение данных:

$post = Post::find(3);

$user = $post->user;

echo $user->name;

##### Связь "Многие ко многим".

Связываем две таблицы: **roles**и **users**создав таблицу с названием **role\_user**. При использовании произвольного имени связывающей таблицы- название указывается явно вторым аргументом в методе **belongsToMany()**:

return $this->belongsToMany('App\Role', 'user\_roles');

так же можно перекрыть имена ключей по умолчанию:

return $this->belongsToMany('App\Role', 'user\_roles', 'user\_id', 'foo\_id');

Создание метода.  
Для модели **User**:

public function roles() {

return $this->belongsToMany('App\Role');

}

Для модели **Role**:

public function users(){

return $this->belongsToMany('App\User');

}

Получение данных:

$user = User::find(1);

$roles = $user->roles;

foreach ($roles as $role) {

*//*

}

как и для других связей, вызываем не метод, а одноименное динамическое свойство "roles".  
Для обратной связи (другой модели), получение данных проводится аналогично.

#### Проверка связей при выборке.

Есть возможность отобрать данные из таблицы, только те, которые имеют связь с другой, указанной таблицей.  
  
Например нужно отобрать всех пользователей, которые написали какие-то посты (имеют связь с таблицей постов – в таблице posts поле user\_id соответствует id пользователя):

$users = User::has('posts')->get();

foreach ($users as $user) {

echo $user->name;

}

можно указать кол-во связей, которое должно быть:

$users = User::has('posts', '>=', '2')->get();

можно добавить произвольное условие выборки:

$users = User::whereHas('posts', function($q){

$q->where('description','like','Описание-%');

})->get();

#### Если использовать не динамическое свойство а метод связи.

В таком случае получаем HasMany Object, для получения данных из которого нужно использовать конструктор запросов:

$user = User::find(1);

$postsHasMany = $user->post();

$posts = $postsHasMany->where('description','like','Описание-%')->get();

foreach ($posts as $post) {

*//*

}

то есть можно создать дополнительные условия для выборки.  
  
При использовании могут возникнуть конфликты если в условии используются поля с одинаковыми названиями для разных таблиц. В таком случае нужно уточнять к какой таблице относится поле. Например:

->where('roles.id','<','10')

### Вставка данных.

Для того, чтобы автоматически заполнялись связывающие поля между таблицами, для вставки данных нужно использовать специальные методы (аналогичные для любых типов связей):  
  
Один из 2-х способов:

$user = User::find(1);

$user->posts()->create([

'description' => 'Описание',

'text' => 'text',

'user\_id' => $user->id

]);

или

$user = User::find(1);

$post = new Post([

'description' => 'Описание',

'text' => 'text',

]);

$user->posts()->save($post);

При использовании метода **save()**«массовой вставки» может появиться ошибка  
MassAssignmentException in Model.php line 225  
это значит, что нужно разрешить вставку указанных полей в модели – метод **$fillable()**.  
  
Сохранить несколько связанных моделей можно так:

$posts = [

new Post(['description' => 'Описание 1', 'text' => 'text']),

new Post(['description' => 'Описание 2', 'text' => 'text']),

new Post(['description' => 'Описание 3', 'text' => 'text']),

];

$user = User::find(1);

$user->posts()->saveMany($posts);

Аналогично и вставка для связимногие ко многим (через связывающую таблицу).  
Пример.  
Нужно создать пользователя с указанными данными и присвоить ему статус админа (в таблице roles соответствует id=1):

$user = new User(['name' => 'Masha', 'email' => 'email']);

$role = Role::find(1);

$role->users()->save($user);

или

$role = Role::find(1);

$role->users()->create([

'name' => 'Pasha',

'email' => 'email2'

]);

в данном случае, создастся указанный пользователь в таблице **users**, а так же в связывающей таблице “**role\_user**” создастся связывающая запись.

### Обновление данных.

Например нужно для пользователя под id=1 (таблица users) обновить поле 'text' из связанной таблицы постов (posts)

$user = User::find(1);

$user->posts()->where('id',2)->update(['text'=>'new text']);

1. сначала получаем модель нужного пользователя;
2. для данной модели вызываем связывающий метод;
3. т.к. один пользователь может иметь несколько записей, оператором where уточняем какой именно пост (id=2) нужно изменить;
4. в методе update() указываем поля и их новые значения.

Если не указать условие (where), то изменены будут все строки связанные с данным пользователем.  
  
Читайте так же про методы, позволяющие изменять значения связывающих полей/таблиц в моей [заметке](http://klisl.com/laravel_foreign.html).

https://tokmakov.msk.ru/blog/item/548

Это пароль кудато beK-w9K-37x-dKL

sudo docker-compose up -d запускаем докер

sudo docker-compose stop останавливаем

------------------------------------------------

sudo apt update && sudo apt-get upgrade

получилось

-------------------------------------------

sudo docker-compose up -d

php artisan ui vue --auth

sudo docker-compose exec myapp npm install

sudo docker-compose exec myapp npm fund

sudo docker-compose exec myapp npm run dev

------------------------------------------

//////////////////////

устанавливаем LARAVEL

$ mkdir ~/myapp && cd ~/myapp

$ curl -LO https://raw.githubusercontent.com/bitnami/bitnami-docker-laravel/master/docker-compose.yml

меняем порты для базы и работы

$ docker-compose up

sudo apt install composer

composer global require laravel/installer

(sudo docker-compose exec myapp composer require laravel/ui)

composer require laravel/ui:^2.4

sudo docker-compose exec myapp composer require laravel/ui "^1.2" --dev

sudo docker-compose exec myapp composer require laravel/ui --dev

php artisan ui vue --auth

node -v

npm -v

sudo apt install nodejs

sudo apt install npm

npm install

npm run dev запускаем для пересборки

npm run watch or npm run watch-poll Для атоматической персборки

npm install vuelidate --save установка валидатора

или npm run development

sudo docker-compose up -d

npm run watch

--------------------------------------------

сама работа

php artisan make:migration create\_users\_table

sudo docker-compose exec myapp php artisan migrate выполнитьмиграцию

sudo docker-compose exec myapp php artisan migrate:rollback последние

sudo docker-compose exec myapp php artisan migrate:reset

sudo docker-compose exec myapp php artisan migrate:refresh

sudo docker-compose exec myapp php artisan make:model Degree создание модели

php artisan make:controller PhotoController --resource создание ресурсного контроллера

php artisan make:controller PhotoController

sudo docker-compose exec myapp nmp install

sudo docker-compose exec myapp php artisan ui vue --auth

-------------------------------------------

sudo docker container ls -a

sudo docker ps

sudo docker run -it

0720941e029d

2e6a9b3eb6ec

-------------------------------------------

npm install pdfmake --save-dev

ниже нет русского

npm install --save jspdf

-------------------------------------------

форматировщик vetur для vue

Split HTML Attributes (Vue, React, Angular)

Prettier - Code formatter

Prettier - Code formatter

prettier.prettier-vscode

-------------------------------------------

В модальном окне для карате разметка

Ubuntu — Ctrl + Shift + I

Свернуть внутреннюю область возле курсора вам помогут следующие сочетания клавиш:

Windows /Ubuntu — Ctrl + Shift + [

Ubuntu — Ctrl+ d выделить слово

Ubuntu — Ctrl + g переход к строке

Ubuntu — Ctrl + Shift + d дублирование строки

Ubuntu — Ctrl + x удаление строки

Ubuntu — Ctrl + Alt + стрелка вверх / вниз добавление курсора

Ubuntu — Shift + Alt выделение столбца

Windows — Shift + Alt + стрелка вверх / вниз Копирование строки

===================================================

видео

https://www.youtube.com/watch?v=LO\_p\_3vukLE

загрузка файлов

https://youtu.be/J-J-EJ9CXNw

передача данных в вью

https://youtu.be/NNdTljRzPqE

https://youtu.be/UAxNcJprLA0

===================================

docker-machine start default

docker-machine ls

---------------------------------------------------

sudo apt update

sudo apt install mysql-server

sudo mysql\_secure\_installation

1

12345678

sudo mysqlSELECT user,authentication\_string,plugin,host FROM mysql.user;

ALTER USER 'root'@'localhost' IDENTIFIED WITH mysql\_native\_password BY 'password';

ALTER USER 'root'@'localhost' IDENTIFIED WITH mysql\_native\_password BY '12345678';

ALTER USER 'root'@'localhost' IDENTIFIED WITH mysql\_native\_password BY '17My#pam20';(mysql -u root -p)

FLUSH PRIVILEGES;

SELECT user,authentication\_string,plugin,host FROM mysql.user;

CREATE USER 'sammy'@'localhost' IDENTIFIED BY 'password';

CREATE USER 'lar'@'localhost' IDENTIFIED BY '#pamServ20';

systemctl status mysql.service

-------------------------------------

sudo mysql

mysql -u lar -p

SHOW DATABASES;

CREATE DATABASE testdb;

> composer require laravel/ui

> npm install && npm run dev

php artisan make:controller Auth/RegisterController

[Главная](https://tokmakov.msk.ru/) > [Блог](https://tokmakov.msk.ru/blog) > [Web-разработка](https://tokmakov.msk.ru/blog/category/1) > [Yii2 и Laravel](https://tokmakov.msk.ru/blog/category/18) >

Блог на Laravel 7, часть 2. Регистрация и аутентификация, восстановление пароля

05.12.2020

Теги: [Laravel](https://tokmakov.msk.ru/blog/tags/314) • [MySQL](https://tokmakov.msk.ru/blog/tags/8) • [Web-разработка](https://tokmakov.msk.ru/blog/tags/9) • [Аутентификация](https://tokmakov.msk.ru/blog/tags/316) • [БазаДанных](https://tokmakov.msk.ru/blog/tags/87) • [Блог](https://tokmakov.msk.ru/blog/tags/315) • [Пользователь](https://tokmakov.msk.ru/blog/tags/259) • [Практика](https://tokmakov.msk.ru/blog/tags/285) • [Форма](https://tokmakov.msk.ru/blog/tags/138) • [Фреймворк](https://tokmakov.msk.ru/blog/tags/275) • [ШаблонСайта](https://tokmakov.msk.ru/blog/tags/214)

Вообще, готовых пакетов для аутентификации существует великое множество, но мы все сделаем сами, чтобы хорошенько разобраться, как это работает. Нам нужно будет создать контроллеры: RegisterController — для регистрации, LoginController — для аутентификации, ForgotPasswordController — для восстановления пароля.

Layout-шаблон

Создаем шаблон resources/views/layout/site.blade.php:

<!doctype html>

<html lang="ru">

<head>

<meta charset="UTF-8">

<meta name="viewport"

content="width=device-width, user-scalable=no, initial-scale=1.0, maximum-scale=1.0, minimum-scale=1.0">

<meta http-equiv="X-UA-Compatible" content="ie=edge">

<title>{{ $title ?? env('APP\_NAME') }}</title>

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

<script src="{{ asset('js/app.js') }}"></script>

</head>

<body>

<div class="container">

<nav class="navbar navbar-expand-lg navbar-dark bg-dark mb-4">

<!-- Логотип и кнопка «Гамбургер» -->

<a class="navbar-brand" href="/">Блог</a>

<button class="navbar-toggler" type="button" data-toggle="collapse"

data-target="#navbar-blog" aria-controls="navbar-blog"

aria-expanded="false" aria-label="Toggle navigation">

<span class="navbar-toggler-icon"></span>

</button>

<!-- Основная часть меню (может содержать ссылки, формы и прочее) -->

<div class="collapse navbar-collapse" id="navbar-blog">

<!-- Этот блок расположен слева -->

<ul class="navbar-nav mr-auto">

<li class="nav-item">

<a class="nav-link" href="#">Блог</a>

</li>

<li class="nav-item">

<a class="nav-link" href="#">Теги</a>

</li>

<li class="nav-item">

<a class="nav-link" href="#">Контакты</a>

</li>

</ul>

<!-- Этот блок расположен посередине -->

<form class="form-inline my-2 my-lg-0">

<input class="form-control mr-sm-2" type="search"

placeholder="Поиск по блогу" aria-label="Search">

<button class="btn btn-outline-info my-2 my-sm-0"

type="submit">Искать</button>

</form>

<!-- Этот блок расположен справа -->

<ul class="navbar-nav ml-auto">

@guest

<li class="nav-item">

<a class="nav-link" href="{{ route('auth.login') }}">Войти</a>

</li>

<li class="nav-item">

<a class="nav-link" href="{{ route('auth.register') }}">Регистрация</a>

</li>

@else

<li class="nav-item">

<a class="nav-link" href="{{ route('user.index') }}">Личный кабинет</a>

</li>

<li class="nav-item">

<a class="nav-link" href="{{ route('auth.logout') }}">Выйти</a>

</li>

@endif

</ul>

</div>

</nav>

<div class="row">

<div class="col-md-3">

<h4>Категории блога</h4>

<p>Здесь будут категории блога</p>

<h4>Популярные теги</h4>

<p>Здесь будут популярные теги</p>

</div>

<div class="col-md-9">

@if ($message = Session::get('success'))

<div class="alert alert-success alert-dismissible mt-0" role="alert">

<button type="button" class="close" data-dismiss="alert" aria-label="Закрыть">

<span aria-hidden="true">&times;</span>

</button>

{{ $message }}

</div>

@endif

@if ($errors->any())

<div class="alert alert-danger alert-dismissible mt-4" role="alert">

<button type="button" class="close" data-dismiss="alert" aria-label="Закрыть">

<span aria-hidden="true">&times;</span>

</button>

<ul class="mb-0">

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

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

@endforeach

</ul>

</div>

@endif

@yield('content')

</div>

</div>

</div>

</body>

</html>Копировать

Чтобы использовать фреймворк bootstrap, выполяняем в консоли две команды:

> composer require laravel/ui

> npm install && npm run devКопировать

Посредники auth и guest

Сразу после установки Laravel доступны посредники auth (для аутентифицированных пользователй) и guest (для не аутентифицированных пользователей) — они определены в файле app/Http/Kernel.php. Мы будем использовать эти посредники, чтобы защитить маршруты контроллеров.

class Kernel extends HttpKernel {

/\*\*

\* The application's route middleware.

\*

\* These middleware may be assigned to groups or used individually.

\*

\* @var array

\*/

protected $routeMiddleware = [

'auth' => \App\Http\Middleware\Authenticate::class,

/\* ... \*/

'guest' => \App\Http\Middleware\RedirectIfAuthenticated::class,

/\* ... \*/

];

}Копировать

Если не аутентифицированный пользователь попробует перейти по маршруту, защищенному посредником auth, он будет перенаправлен на маршрут login. У нас такого маршрута нет, вместо него у нас будет auth.login, так что вносим изменения в класс посредника Authenticate.

namespace App\Http\Middleware;

use Illuminate\Auth\Middleware\Authenticate as Middleware;

class Authenticate extends Middleware {

/\*\*

\* Get the path the user should be redirected to when they are not authenticated.

\*

\* @param \Illuminate\Http\Request $request

\* @return string|null

\*/

protected function redirectTo($request) {

if (! $request->expectsJson()) {

// не аутентифицированных пользователей отправляем

// на страницу формы входа в личный кабинет

return route('auth.login');

}

}

}Копировать

Если аутентифицированный пользователь попробует перейти по маршруту, защищенному посредником guest, он будет перенаправлен на RouteServiceProvider::HOME. Но у нас такой страницы нет, так что вносим изменения в класс посредника RedirectIfAuthenticated.

namespace App\Http\Middleware;

use Closure;

use Illuminate\Support\Facades\Auth;

class RedirectIfAuthenticated {

/\*\*

\* Handle an incoming request.

\*

\* @param \Illuminate\Http\Request $request

\* @param \Closure $next

\* @param string|null $guard

\* @return mixed

\*/

public function handle($request, Closure $next, $guard = null) {

if (Auth::guard($guard)->check()) {

// аутентифицированных пользователей отправляем

// на главную страницу личного кабинета

return redirect()->route('user.index');

}

return $next($request);

}

}Копировать

Регистрация

1. Добавляем маршруты

Добавляем маршруты в файл routes/web.php:

/\*

\* Регистрация, вход в ЛК, восстановление пароля

\*/

Route::group([

'as' => 'auth.', // имя маршрута, например auth.index

'prefix' => 'auth', // префикс маршрута, например auth/index

], function () {

// форма регистрации

Route::get('register', 'Auth\RegisterController@register')

->name('register');

// создание пользователя

Route::post('register', 'Auth\RegisterController@create')

->name('create');

});Копировать

2. Контроллер RegisterController

Создаем контроллер RegisterController:

> php artisan make:controller Auth/RegisterControllerКопировать

namespace App\Http\Controllers\Auth;

use App\Http\Controllers\Controller;

use Illuminate\Http\Request;

use App\User;

use Hash;

class RegisterController extends Controller {

public function \_\_construct() {

$this->middleware('guest');

}

/\*\*

\* Форма регистрации

\*/

public function register() {

return view('auth.register');

}

/\*\*

\* Добавление пользователя

\*/

public function create(Request $request) {

$request->validate([

'name' => 'required|string|max:255',

'email' => 'required|string|email|max:255|unique:users',

'password' => 'required|string|min:8|confirmed',

]);

User::create([

'name' => $request->name,

'email' => $request->email,

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

]);

return redirect()

->route('auth.login')

->with('success', 'Вы успешно зарегистрировались');

}

}Копировать

Все маршруты контроллера защищены посредником guest, так что аутентифицированный пользователь не сможет перейти по ним.

3. Форма регистрации

Создаем шаблон resources/views/auth/register.blade.php:

@extends('layout.site', ['title' => 'Регистрация'])

@section('content')

<h1>Регистрация</h1>

<form method="post" action="{{ route('auth.register') }}">

@csrf

<div class="form-group">

<input type="text" class="form-control" name="name" placeholder="Имя, Фамилия"

required maxlength="255" value="{{ old('name') ?? '' }}">

</div>

<div class="form-group">

<input type="email" class="form-control" name="email" placeholder="Адрес почты"

required maxlength="255" value="{{ old('email') ?? '' }}">

</div>

<div class="form-group">

<input type="text" class="form-control" name="password" placeholder="Придумайте пароль"

required maxlength="255" value="">

</div>

<div class="form-group">

<input type="text" class="form-control" name="password\_confirmation"

placeholder="Пароль еще раз" required maxlength="255" value="">

</div>

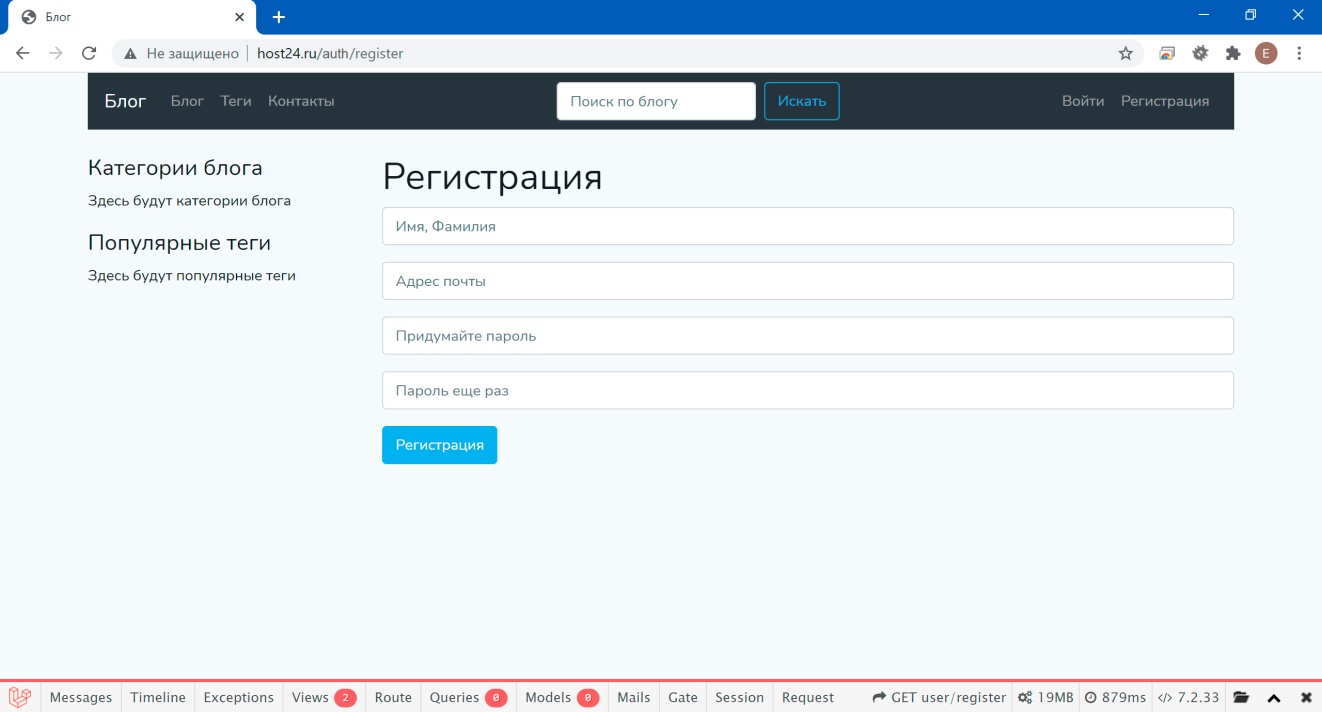
<div class="form-group">

<button type="submit" class="btn btn-info text-white">Регистрация</button>

</div>

</form>

@endsectionКопировать

[](https://tokmakov.msk.ru/files/blog/596/1.jpg)

Аутентификация

1. Добавляем маршруты

Добавляем маршруты в файл routes/web.php:

/\*

\* Регистрация, вход в ЛК, восстановление пароля

\*/

Route::group([

'as' => 'auth.', // имя маршрута, например auth.index

'prefix' => 'auth', // префикс маршрута, например auth/index

], function () {

// форма регистрации

Route::get('register', 'Auth\RegisterController@register')

->name('register');

// создание пользователя

Route::post('register', 'Auth\RegisterController@create')

->name('create');

// форма входа

Route::get('login', 'Auth\LoginController@login')

->name('login');

// аутентификация

Route::post('login', 'Auth\LoginController@authenticate')

->name('auth');

// выход

Route::get('logout', 'Auth\LoginController@logout')

->name('logout');

});

/\*

\* Личный кабинет пользователя

\*/

Route::group([

'as' => 'user.', // имя маршрута, например user.index

'prefix' => 'user', // префикс маршрута, например user/index

'namespace' => 'User', // пространство имен контроллеров

'middleware' => ['auth'] // один или несколько посредников

], function () {

// главная страница

Route::get('index', 'IndexController')->name('index');

});Копировать

2. Контроллер LoginController

Создаем контроллер LoginController:

> php artisan make:controller Auth/LoginControllerКопировать

namespace App\Http\Controllers\Auth;

use App\Http\Controllers\Controller;

use Illuminate\Http\Request;

use Auth;

class LoginController extends Controller {

public function \_\_construct() {

$this->middleware('guest', ['except' => 'logout']);

}

/\*\*

\* Форма входа в личный кабинет

\*/

public function login() {

return view('auth.login');

}

/\*\*

\* Аутентификация пользователя

\*/

public function authenticate(Request $request) {

$request->validate([

'email' => 'required|string|email',

'password' => 'required|string',

]);

$credentials = $request->only('email', 'password');

if (Auth::attempt($credentials)) {

return redirect()

->route('user.index')

->with('success', 'Вы вошли в личный кабинет');

}

return redirect()

->route('auth.login')

->withErrors('Неверный логин или пароль');

}

/\*\*

\* Выход из личного кабинета

\*/

public function logout() {

Auth::logout();

return redirect()

->route('auth.login')

->with('success', 'Вы вышли из личного кабинета');

}

}Копировать

Все маршруты контроллера (за исключением метода logout()) защищены посредником guest, так что аутентифицированный пользователь не сможет перейти по ним.

3. Форма входа

Создаем шаблон resources/views/auth/login.blade.php:

@extends('layout.site', ['title' => 'Вход в личный кабинет'])

@section('content')

<h1>Вход в личный кабинет</h1>

<form method="post" action="{{ route('auth.auth') }}">

@csrf

<div class="form-group">

<input type="email" class="form-control" name="email" placeholder="Адрес почты"

required maxlength="255" value="{{ old('email') ?? '' }}">

</div>

<div class="form-group">

<input type="text" class="form-control" name="password" placeholder="Ваш пароль"

required maxlength="255" value="">

</div>

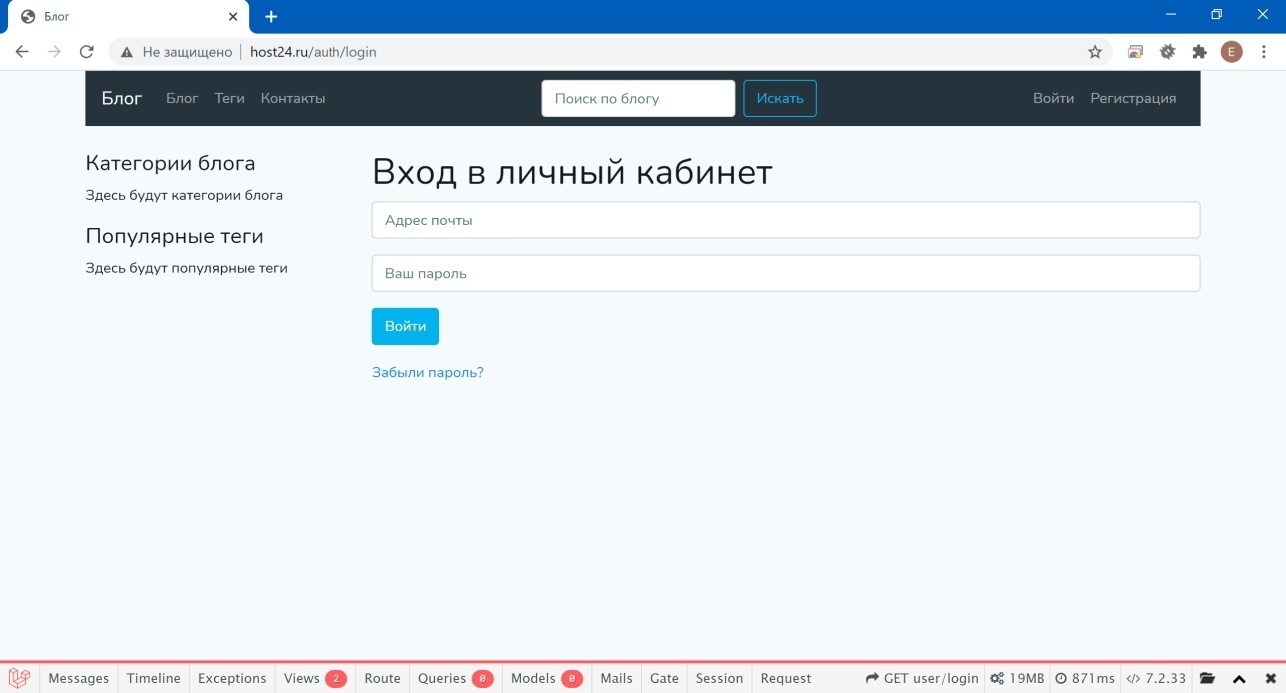
<div class="form-group">

<button type="submit" class="btn btn-info text-white">Войти</button>

</div>

</form>

@endsectionКопировать

[](https://tokmakov.msk.ru/files/blog/596/2.jpg)

4. Личный кабинет

Создаем контроллер IndexController:

> php artisan make:controller User/IndexController --invokableКопировать

namespace App\Http\Controllers\User;

use App\Http\Controllers\Controller;

use Illuminate\Http\Request;

class IndexController extends Controller {

/\*\*

\* Handle the incoming request.

\*

\* @param \Illuminate\Http\Request $request

\* @return \Illuminate\Http\Response

\*/

public function \_\_invoke(Request $request) {

return view('user.index');

}

}Копировать

Создаем шаблон resources/views/user/index.blade.php

@extends('layout.site', ['title' => 'Личный кабинет'])

@section('content')

<h1>Личный кабинет</h1>

<p>Добрый день {{ auth()->user()->name }}!</p>

<p>Это личный кабинет пользователя сайта.</p>

@endsectionКопировать

Все маршруты контроллера защищены посредником auth, так что попасть в личный кабинет может только аутентифицированный пользователь.

Восстановление пароля

1. Добавляем маршруты

Добавляем маршруты в файл routes/web.php:

/\*

\* Регистрация, вход в ЛК, восстановление пароля

\*/

Route::group([

'as' => 'auth.', // имя маршрута, например auth.index

'prefix' => 'auth', // префикс маршрута, например auth/index

], function () {

// форма регистрации

Route::get('register', 'Auth\RegisterController@register')

->name('register');

// создание пользователя

Route::post('register', 'Auth\RegisterController@create')

->name('create');

// форма входа

Route::get('login', 'Auth\LoginController@login')

->name('login');

// аутентификация

Route::post('login', 'Auth\LoginController@authenticate')

->name('auth');

// выход

Route::get('logout', 'User\LoginController@logout')

->name('logout');

// форма ввода адреса почты

Route::get('forgot-password', 'Auth\ForgotPasswordController@form')

->name('forgot-form');

// письмо на почту

Route::post('forgot-password', 'Auth\ForgotPasswordController@mail')

->name('forgot-mail');

});Копировать

2. Контроллер ForgotPasswordController

Создаем контроллер ForgotPasswordController:

> php artisan make:controller Auth/ForgotPasswordControllerКопировать

namespace App\Http\Controllers\Auth;

use App\Http\Controllers\Controller;

use Carbon\Carbon;

use Illuminate\Http\Request;

use Illuminate\Support\Facades\DB;

use Illuminate\Support\Facades\Mail;

use Illuminate\Support\Str;

class ForgotPasswordController extends Controller {

public function \_\_construct() {

$this->middleware('guest');

}

/\*\*

\* Форма ввода адреса почты

\*/

public function form() {

return view('auth.forgot');

}

/\*\*

\* Письмо на почту для восстановления

\*/

public function mail(Request $request) {

$request->validate([

'email' => 'required|email|exists:users',

]);

$token = Str::random(60);

DB::table('password\_resets')->insert(

['email' => $request->email, 'token' => $token, 'created\_at' => Carbon::now()]

);

// ссылка для сброса пароля

$link = route('auth.reset-form', ['token' => $token, 'email' => $request->email]);

Mail::send(

'email.reset-password',

['link' => $link],

function($message) use ($request) {

$message->to($request->email);

$message->subject('Восстановление пароля');

}

);

return back()->with('success', 'Ссылка для восстановления пароля отправлена на почту');

}

}Копировать

Все маршруты контроллера защищены посредником guest, так что аутентифицированный пользователь не сможет перейти по ним.

3. Форма ввода адреса

Создаем шаблон resources/views/auth/forgot.blade.php:

@extends('layout.site', ['title' => 'Восстановление пароля'])

@section('content')

<h1>Восстановление пароля</h1>

<form method="post" action="{{ route('auth.forgot-mail') }}">

@csrf

<div class="form-group">

<input type="email" class="form-control" name="email" placeholder="Адрес почты"

required maxlength="255" value="{{ old('email') ?? '' }}">

</div>

<div class="form-group">

<button type="submit" class="btn btn-info text-white">Восстановить</button>

</div>

</form>

@endsectionКопировать

4. Шаблон для письма

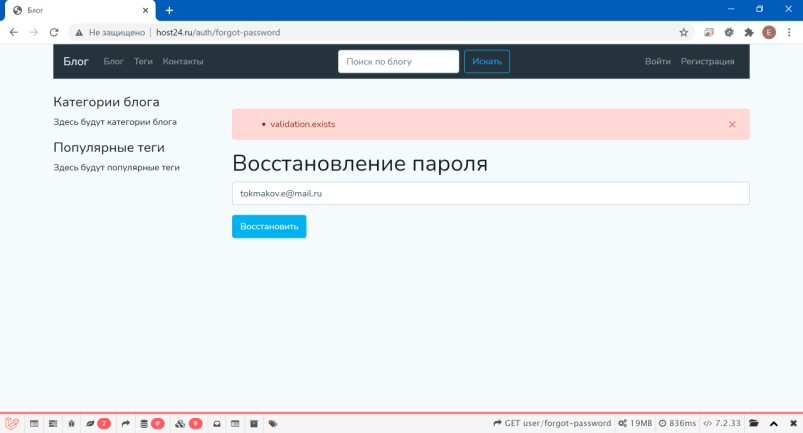
Создаем файл resources/views/email/reset-password.blade.php:

<h3>Восстановление пароля</h3>

<p>Для восстановления пароля перейдите по этой <a href="{{ $link }}">ссылке</a>.</p>Копировать

4. Перевод на русский

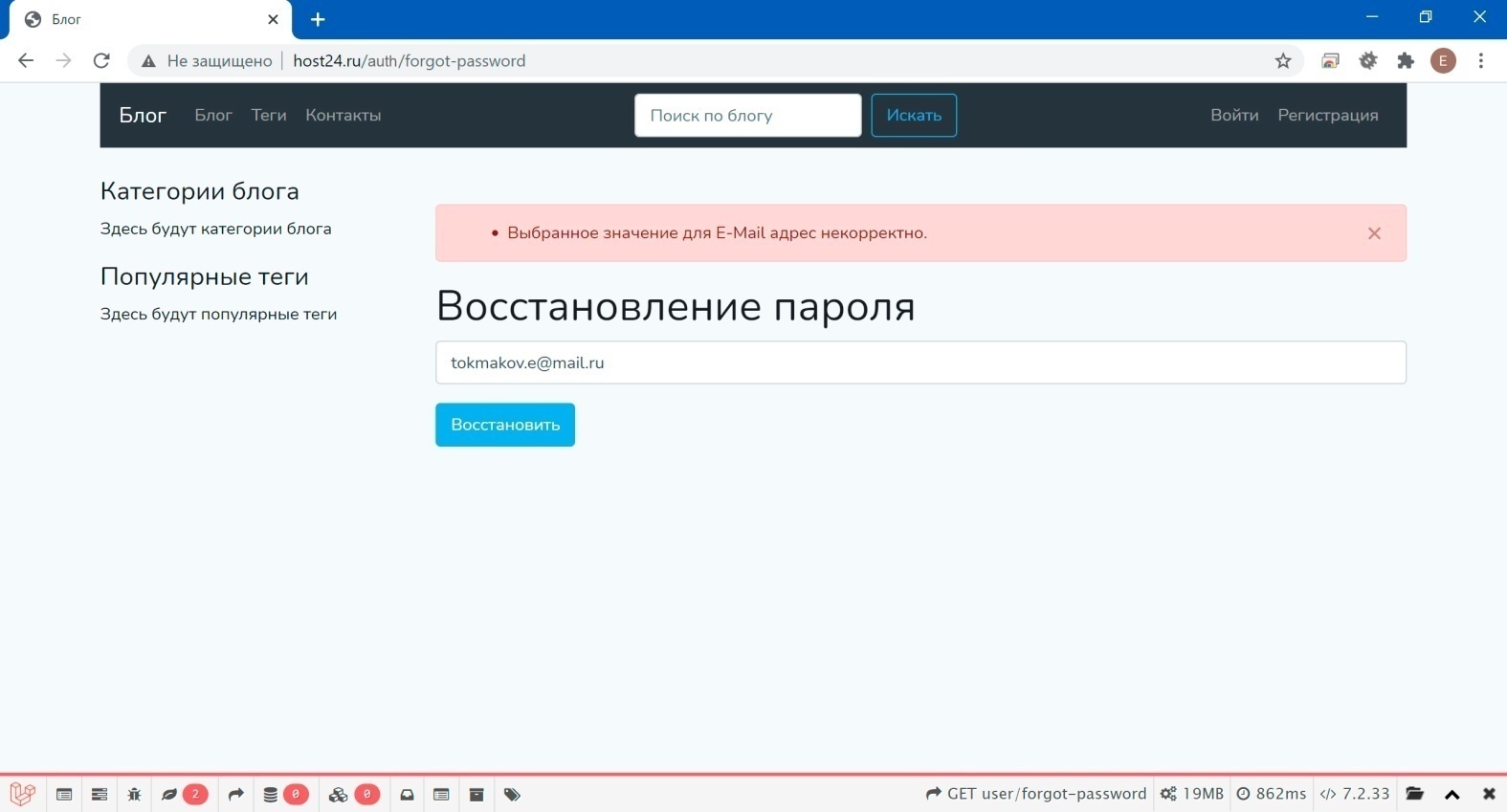
При вводе адреса почты не зарегистрированного пользователя получаем такое сообщение об ошибке:

[](https://tokmakov.msk.ru/files/blog/596/3.jpg)

В настройках config/app.php у меня указан русский язык, но файлов перевода нет. Строка validation.exists означает, что должно быть показано сообщение exists из файла resources/lang/ru/validation.php. Установим языковый пакет

> composer require laravel-lang/lang:~7.0Копировать

После чего скопируем vendor/laravel-lang/lang/src/ru в директорию resources/lang.

[](https://tokmakov.msk.ru/files/blog/596/4.jpg)

5. Отправка письма

На этапе разработки письма отправлять не будем, вместо этого будем их записывать в log-файл. Для этого редактируем файл конфигурации .env:

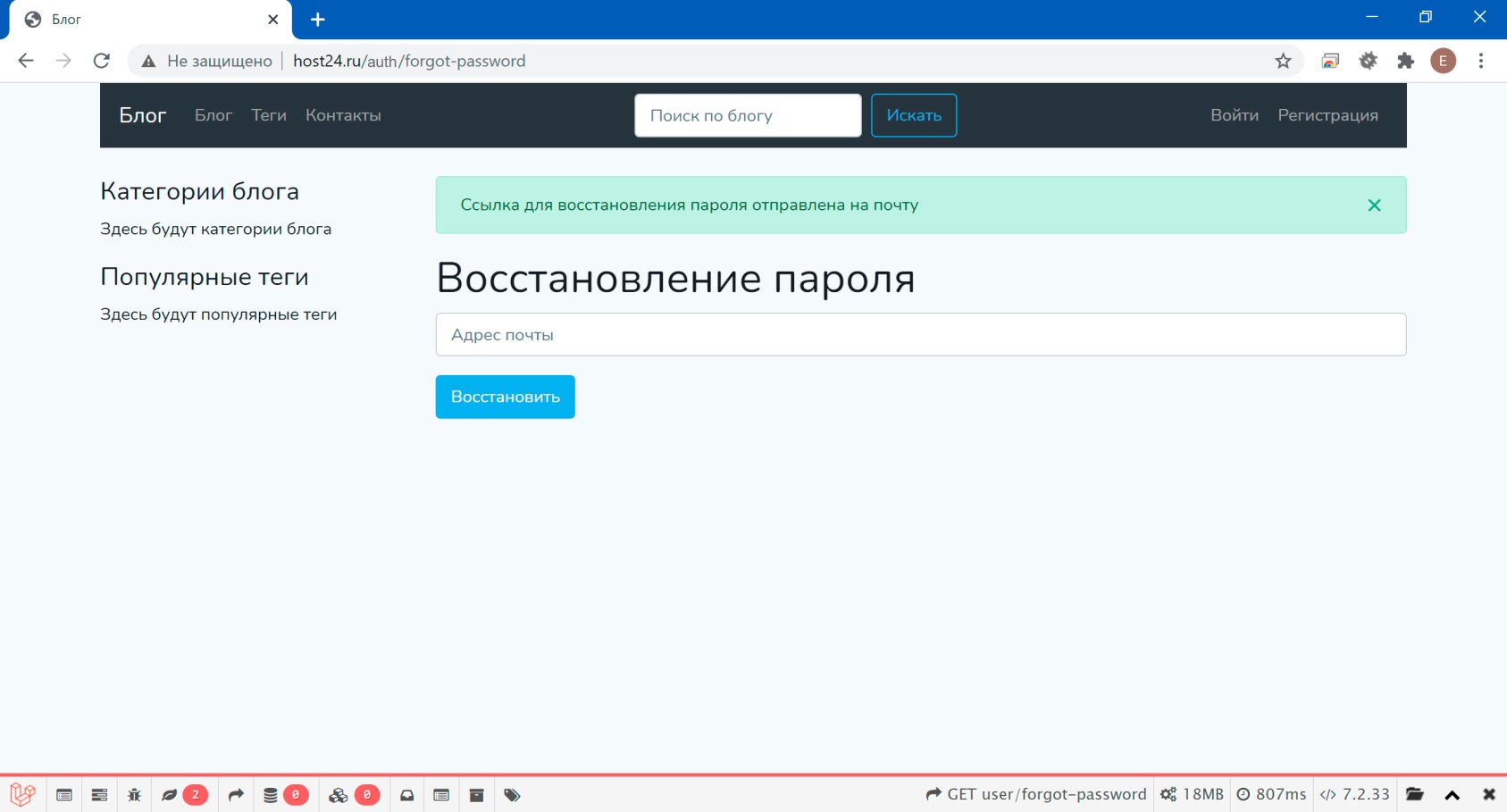
MAIL\_MAILER=log

MAIL\_FROM\_ADDRESS=noreply@example.com

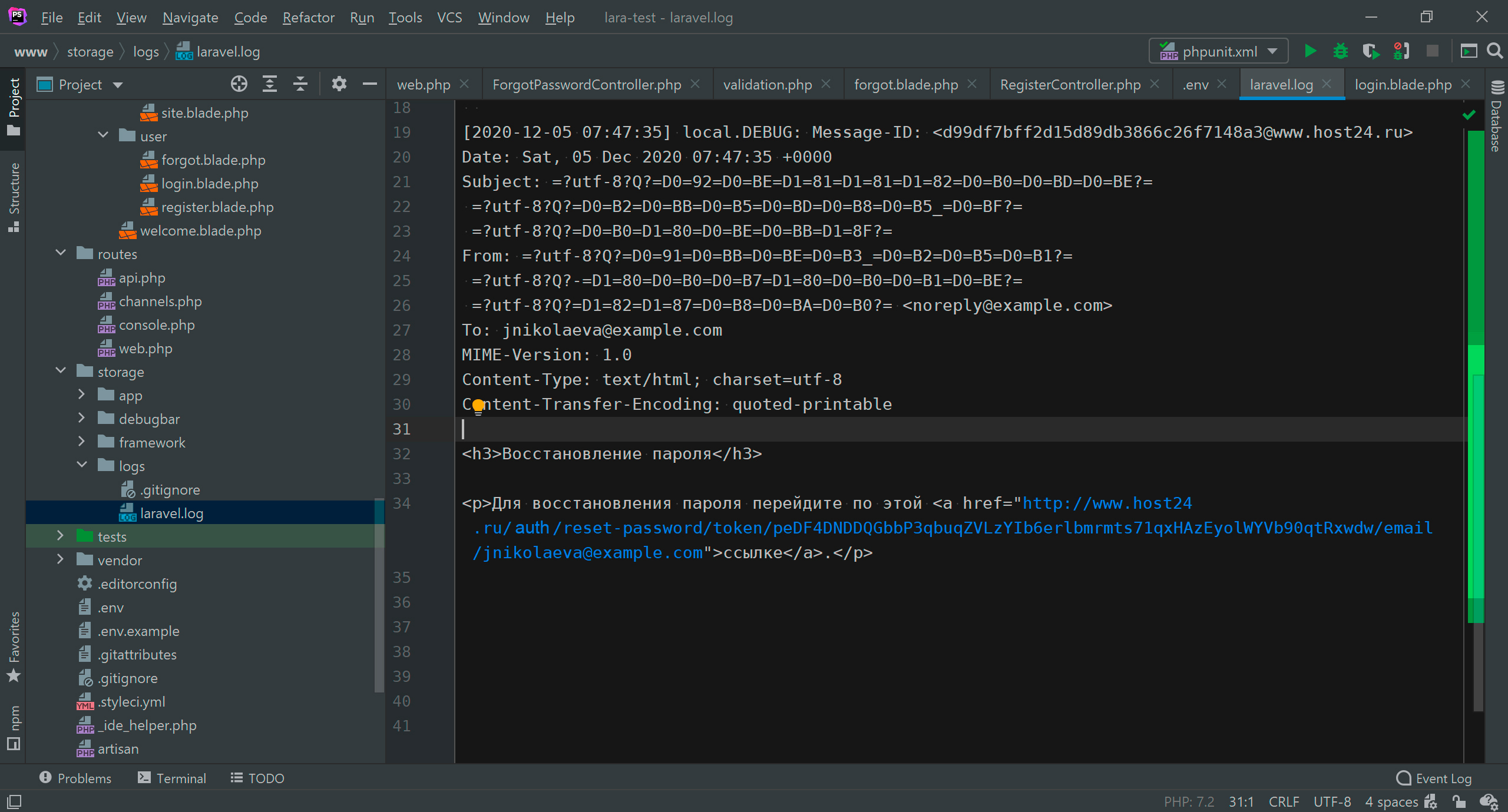
MAIL\_FROM\_NAME="${APP\_NAME}"Копировать

Эти настройки используются в файле config/mail.php, поэтому при отправке письма мы не используем метод from() для установки адреса почты и имени отправителя письма.

Возьмем из базы данных адрес почты любого пользователя и попробуем восстановить пароль:

[](https://tokmakov.msk.ru/files/blog/596/5.jpg)

Проверяем log-файл — письмо, содержащее ссылку для восстановления пароля, было отправлено:

[](https://tokmakov.msk.ru/files/blog/596/6.jpg)

6. Добавляем маршруты

Добавляем маршруты в файл routes/web.php:

/\*

\* Регистрация, вход в ЛК, восстановление пароля

\*/

Route::group([

'as' => 'auth.', // имя маршрута, например auth.index

'prefix' => 'auth', // префикс маршрута, например auth/index

], function () {

// форма регистрации

Route::get('register', 'Auth\RegisterController@register')

->name('register');

// создание пользователя

Route::post('register', 'Auth\RegisterController@create')

->name('create');

// форма входа

Route::get('login', 'Auth\LoginController@login')

->name('login');

// аутентификация

Route::post('login', 'Auth\LoginController@authenticate')

->name('auth');

// выход

Route::get('logout', 'Auth\LoginController@logout')

->name('logout');

// форма ввода адреса почты

Route::get('forgot-password', 'Auth\ForgotPasswordController@form')

->name('forgot-form');

// письмо на почту

Route::post('forgot-password', 'Auth\ForgotPasswordController@mail')

->name('forgot-mail');

// форма восстановления пароля

Route::get(

'reset-password/token/{token}/email/{email}',

'Auth\ResetPasswordController@form'

)->name('reset-form');

// восстановление пароля

Route::post('reset-password', 'Auth\ResetPasswordController@reset')

->name('reset-password');

});Копировать

7. Контроллер ResetPasswordController

Создаем контроллер ResetPasswordController:

> php artisan make:controller Auth/ResetPasswordControllerКопировать

namespace App\Http\Controllers\Auth;

use App\Http\Controllers\Controller;

use App\User;

use Illuminate\Http\Request;

use Illuminate\Support\Carbon;

use Illuminate\Support\Facades\DB;

use Hash;

class ResetPasswordController extends Controller {

public function \_\_construct() {

$this->middleware('guest');

}

/\*\*

\* Форма ввода нового пароля

\*/

public function form($token, $email) {

return view('auth.reset-password', compact('token', 'email'));

}

/\*\*

\* Установка нового пароля

\*/

public function reset(Request $request) {

$request->validate([

'email' => 'required|email|exists:users',

'password' => 'required|string|min:8|confirmed',

]);

// удаляем старые записи из таблицы сброса паролей

$expire = Carbon::now()->subMinute(60);

DB::table('password\_resets')

->where('created\_at', '<', $expire)

->delete();

// если ссылка на восстановления была отправлена

$row = DB::table('password\_resets')

->where([

'email' => $request->email,

'token' => $request->token,

])

->first();

// если ссылка уже устарела, то ничего не делаем

if(!$row) {

return back()->withErrors('Ссылка восстановления пароля устарела');

}

// устанавливаем новый пароль для пользователя

User::where('email', $request->email)

->update(['password' => Hash::make($request->password)]);

// удаляем пользователя из таблицы сброса паролей

DB::table('password\_resets')->where(['email'=> $request->email])->delete();

return redirect()

->route('auth.login')

->with('success', 'Ваш пароль был успешно изменен');

}

}Копировать

Все маршруты контроллера защищены посредником guest, так что аутентифицированный пользователь не сможет перейти по ним.

8. Форма ввода пароля

Создаем файл resources/views/auth/reset-password.blade.php:

@extends('layout.site', ['title' => 'Сбросить пароль'])

@section('content')

<h1>Сбросить пароль</h1>

<form method="post" action="{{ route('auth.reset-password') }}">

@csrf

<input type="hidden" name="email" value="{{ $email }}">

<input type="hidden" name="token" value="{{ $token }}">

<div class="form-group">

<input type="text" class="form-control" name="password"

placeholder="Придумайте пароль" required maxlength="255" value="">

</div>

<div class="form-group">

<input type="text" class="form-control" name="password\_confirmation"

placeholder="Пароль еще раз" required maxlength="255" value="">

</div>

<div class="form-group">

<button type="submit" class="btn btn-info text-white">Сбросить</button>

</div>

</form>

@endsection