Предварительно убедиться, что в OpenServer выставлены такие настройки:

См. настройку Path

Изображение выглядит как текст

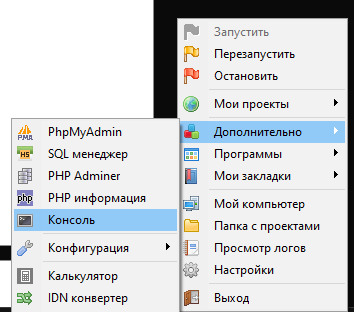
Автоматически созданное описание

См. MySQL/MariaDB

Изображение выглядит как текст

Автоматически созданное описание

Установим Laravel.Откроем консоль:



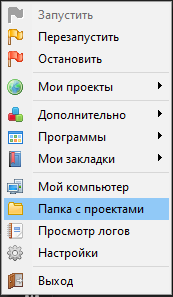
Выполним следующее:

cd domains

composer create-project --prefer-dist laravel/laravel surveys

cd surveys

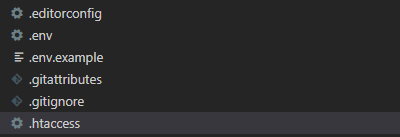
Консоль не закрываем. Откройте проект в редакторе. Папка нашего проекта лежать в папке *domains* OpenServer.



Если мы откроем проект, то нужно получать к нему доступ через /public.

Изображение выглядит как текст, снимок экрана, монитор, экран

Автоматически созданное описаниеИсправим это, создав в корне проекта файл .htaccess с таким содержимым:

 Изображение выглядит как текст

Автоматически созданное описание

Теперь можем получить доступ к проекту по адресу <http://surveys>:

Изображение выглядит как текст

Автоматически созданное описание

Далее нужно настроить базу данных. Отредактируйте файл .env, изменив имя БД и при необходимости данные для подключения:

DB\_CONNECTION=mysql

DB\_HOST=127.0.0.1

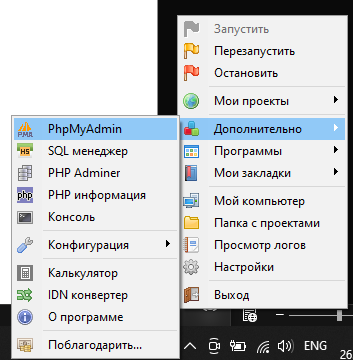
DB\_PORT=3306

DB\_DATABASE=surveys

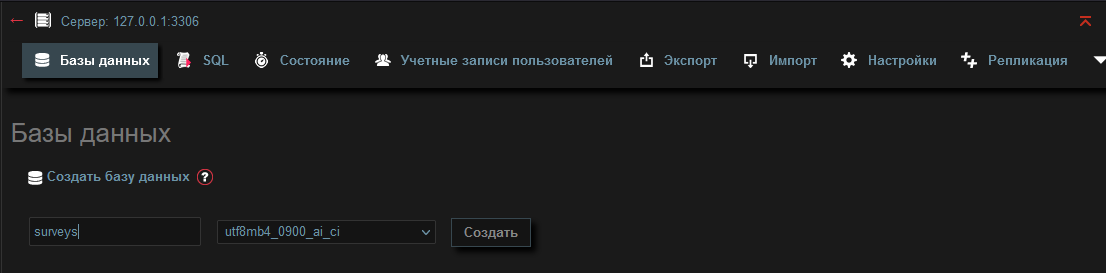
DB\_USERNAME=root

DB\_PASSWORD=

Зайдите в phpMyAdmin. Имя пользователя root, пароль root или без пароля.



Создаем новую базу, называем surveys:

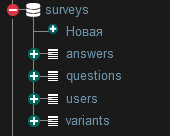


Переходим на вкладку Импорт созданной базы и импортируем предоставленный .sql файл:

Изображение выглядит как текст

Автоматически созданное описание

В дереве она должна отобразиться так:



У нас есть 4 таблицы.

|  |  |
| --- | --- |
| users  пользователи | id – ключ  name – имя пользователя  email – ну тут понятно  password – хэшированный пароль  remember\_token – для хранения сессии  api\_token – токен для авторизации через API  created\_at – дата создания  updated\_at – дата обновления |
| questions  вопросы | id – ключ  text – текст вопроса  user\_id – ИД пользователя создателя  created\_at – дата создания  updated\_at – дата обновления |
| variants  варианты ответов | id – ключ  text – текст варианта ответа  question\_id – ИД вопроса  created\_at – дата создания  updated\_at – дата обновления |
| answers  ответы | id – ключ  user\_id – ИД пользователя  question\_id – ИД вопроса  variant\_id – ИД варианта ответа  created\_at – дата создания  updated\_at – дата обновления |

Создадим соответствующие модели (модель User уже есть, мы ее только модифицируем). Выполним команды:

php artisan make:model Question

php artisan make:model Variants

php artisan make:model Answer

Эти классы должны появиться в папке app/Models:

Изображение выглядит как текст

Автоматически созданное описание

Откроем User.php и добавим api\_token в fillable:

protected $fillable = [

    'name',

    'email',

    'password',

    'api\_token',

];

Сделаем авторизацию. Подключим базовый функционал:

composer require laravel/ui

php artisan ui vue --auth

Появились файлы в resources/views:

Изображение выглядит как текст

Автоматически созданное описание

Надо подключить css и js.

*Первый вариант* - скомпилировать ассеты, если установлен npm:

npm install

npm run prod

если все сломалось,

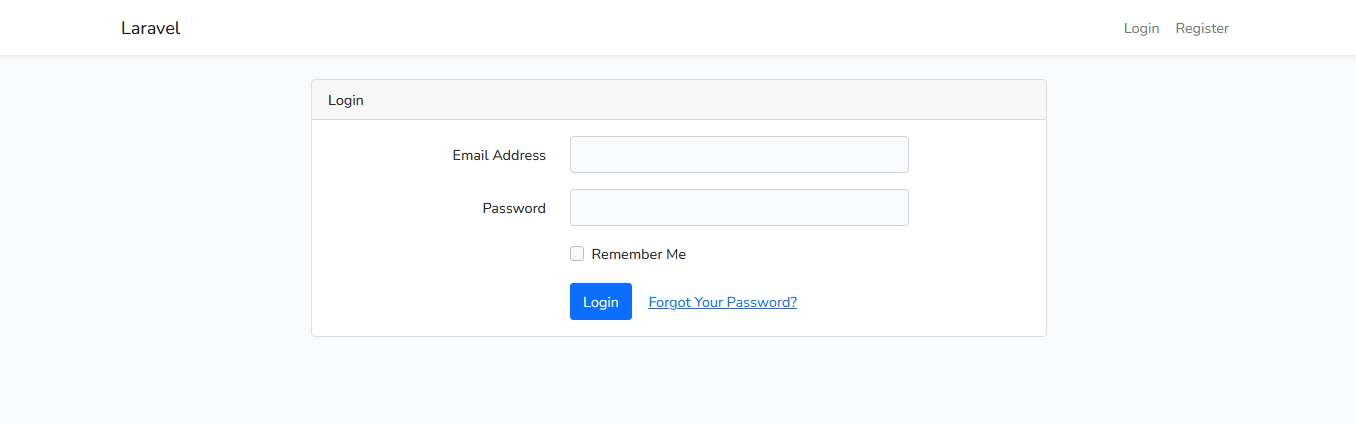
npm upgrade

*Второй вариант* – положить в папку public библиотеку bootstrap таким образом, чтобы папки css и js лежали в public. В resources/views/layouts/app.blade.php заменить подключение app.css на bootstrap.min.css и app.js на bootstrap.bundle.min.js:

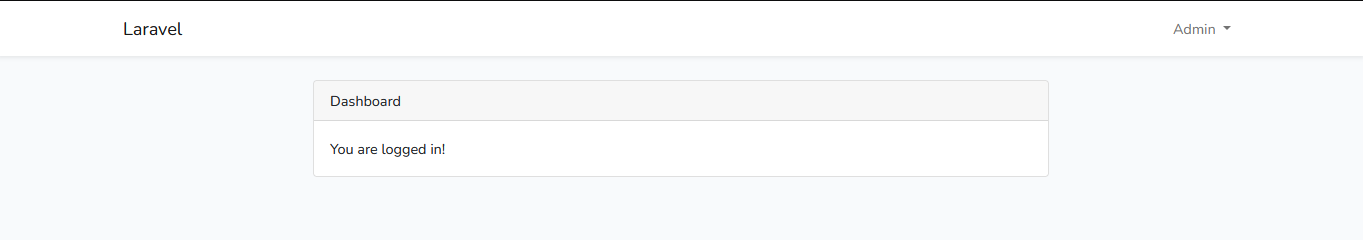
Изображение выглядит как текст

Автоматически созданное описание

Проверим, работает ли авторизация, перейдем на <http://surveys/login>.



Войдем с email [admin@surveys.touschool](mailto:admin@surveys.touschool) и паролем admin:



Только в админке регистрация не будет нужна, также уберем главную страницу, чтобы все было закрыто от неавторизованных. Отредактируем routes/web.php:

/\*

|--------------------------------------------------------------------------

| 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!

|

\*/

Auth::routes(['register' => false]);

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

И изменим app/Providers/RouteServiceProvider.php:

class RouteServiceProvider extends ServiceProvider

{

    /\*\*

     \* The path to the "home" route for your application.

     \*

     \* This is used by Laravel authentication to redirect users after login.

     \*

     \* @var string

     \*/

    public const HOME = '/';

Далее создадим контроллеры админки:

php artisan make:controller UserController

php artisan make:controller QuestionController

И отредактируем модели, добавим поля для массового присваивания и связи.

<https://laravel.com/docs/9.x/eloquent-relationships>

Массовое присвоение – определяет, какие поля можно инициализировать, передав массив.

<https://laravel.com/docs/9.x/eloquent#mass-assignment>

Это будет понятно, когда будем писать контроллеры. Можно либо перечислить доступные поля в fillable, либо перечислить запрещенные в guarded. У нас все будут массово присваиваемые.

Question.php

class Question extends Model

{

    use HasFactory;

    protected $guarded = [];

    public function variants()

    {

        return $this->hasMany(Variant::class);

    }

    public function answers()

    {

        return $this->hasMany(Answer::class);

    }

    public function user()

    {

        return $this->belongsTo(User::class)->withDefault();

    }

}

Определили связи – варианты ответа, ответы респондентов, автор вопроса.

Variant.php

class Variant extends Model

{

    use HasFactory;

    protected $guarded = [];

    public function question()

    {

        return $this->belongsTo(Question::class)->withDefault();

    }

}

Определили связь – вопрос.

Answer.php

class Answer extends Model

{

    use HasFactory;

    protected $guarded = [];

    public function variant()

    {

        return $this->belongsTo(Variant::class)->withDefault();

    }

    public function question()

    {

        return $this->belongsTo(Question::class)->withDefault();

    }

    public function user()

    {

        return $this->belongsTo(User::class)->withDefault();

    }

}

Определили связи – вариант ответа, респондент, вопрос.

Также добавим связи в User.php – вопросы, автором которых является и ответы. $casts, в принципе, не нужен, можно убрать. Класс примет такой вид:

class User extends Authenticatable

{

    use HasApiTokens, HasFactory, Notifiable;

    /\*\*

     \* The attributes that are mass assignable.

     \*

     \* @var array<int, string>

     \*/

    protected $fillable = [

        'name',

        'email',

        'password',

        'api\_token',

    ];

    /\*\*

     \* The attributes that should be hidden for serialization.

     \*

     \* @var array<int, string>

     \*/

    protected $hidden = [

        'password',

        'remember\_token',

    ];

    public function answers()

    {

        return $this->hasMany(Answer::class);

    }

    public function questions()

    {

        return $this->hasMany(Question::class);

    }

}

Сделаем простой контроллер, который выведет список пользователей. Перейдем в app/Http/Controllers/UserController.php. Добавим метод index:

namespace App\Http\Controllers;

use Illuminate\Http\Request;

use App\Models\User;

class UserController extends Controller

{

    public function index()

    {

        $users = User::all();

        return view('users', [

            'users' => $users,

        ]);

    }

}

Добавим маршрут в routes/web.php. Здесь и далее все маршруты помещаем внутрь обертки middleware auth, чтобы они были защищены от неавторизованного доступа.

Route::group(['middleware' => 'auth'], function() {

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

});

Добавим ссылку в resources/views/layouts/app.blade.php в правой части меню:

Изображение выглядит как текст

Автоматически созданное описание

Теперь создадим файл resources/views/users.blade.php со следующим содержанием:

@extends('layouts.app')

@section('content')

<div class="container">

    <h1 class="mb-4">Users</h1>

    <div class="table-responsive">

        <table class="table table-hover">

            <tr>

                <th>ID</th>

                <th>Name</th>

                <th>Email</th>

            </tr>

            @foreach($users as $user)

            <tr>

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

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

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

            </tr>

            @endforeach

        </table>

    </div>

</div>

@endsection

Изображение выглядит как текст

Автоматически созданное описание

Теперь выведем количество созданных вопросов и ответов пользователя. Изменим запрос в UserController.php:

$users = User::withCount('questions')->withCount('answers')->get();

И добавим в шаблон еще 2 колонки:

@extends('layouts.app')

@section('content')

<div class="container">

    <h1 class="mb-4">Users</h1>

    <div class="table-responsive">

        <table class="table table-hover">

            <tr>

                <th>ID</th>

                <th>Name</th>

                <th>Email</th>

                <th>Questions created</th>

                <th>Answers count</th>

            </tr>

            @foreach($users as $user)

            <tr>

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

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

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

                <td>{{ $user->questions\_count }}</td>

                <td>{{ $user->answers\_count }}</td>

            </tr>

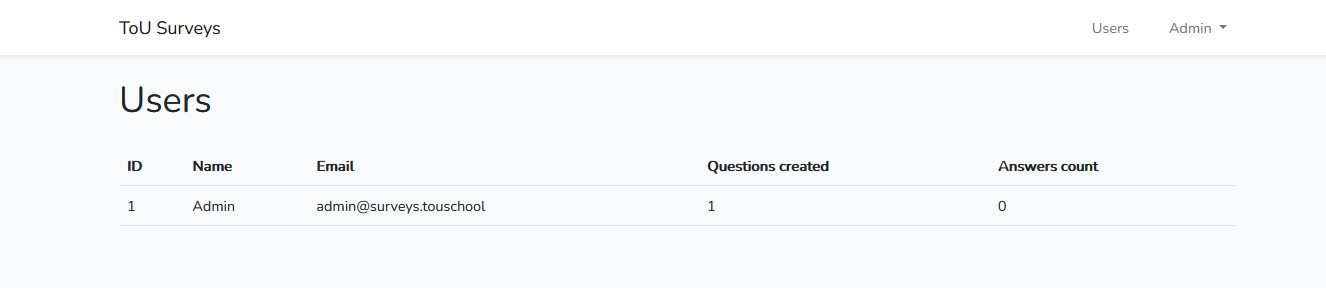
            @endforeach

        </table>

    </div>

</div>

@endsection



Добавим еще один маршрут – список вопросов:

routes/web.php:

Route::group(['middleware' => 'auth'], function() {

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

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

});

app/Http/Controllers/QuestionController.php:

namespace App\Http\Controllers;

use Illuminate\Http\Request;

use App\Models\Question;

class QuestionController extends Controller

{

    public function index()

    {

        $questions = Question::withCount('variants')->withCount('answers')->get();

        return view('questions', [

            'questions' => $questions,

        ]);

    }

}

resources/views/layouts/app.blade.php (добавить еще ссылку):

Изображение выглядит как текст, экран, снимок экрана

Автоматически созданное описание

resources/views/questions.blade.php:

@extends('layouts.app')

@section('content')

<div class="container">

    <h1 class="mb-4">Questions</h1>

    <div class="table-responsive">

        <table class="table table-hover">

            <tr>

                <th>ID</th>

                <th>Text</th>

                <th>Author</th>

                <th>Variants count</th>

                <th>Answers count</th>

            </tr>

            @foreach($questions as $question)

            <tr>

                <td>{{ $question->id }}</td>

                <td>{{ $question->text }}</td>

                <td>{{ $question->user->name }}</td>

                <td>{{ $question->variants\_count }}</td>

                <td>{{ $question->answers\_count }}</td>

            </tr>

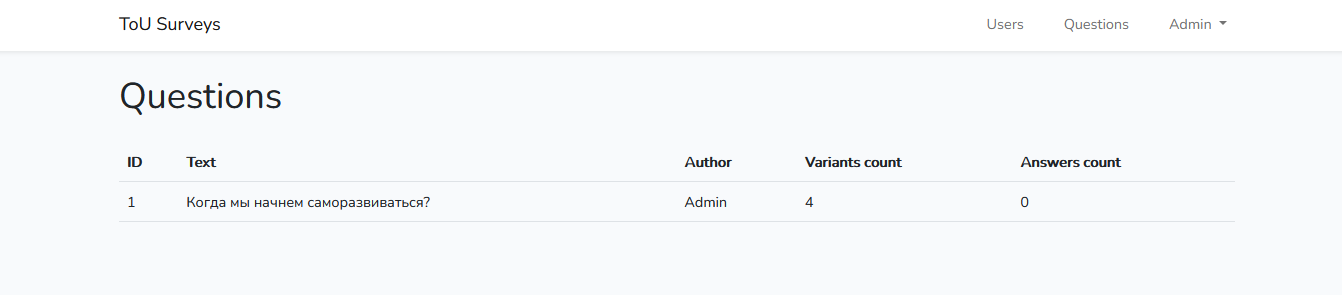
            @endforeach

        </table>

    </div>

</div>

@endsection



Страница вопроса. Создаем марштрут в web.php:

Route::get('/questions/{id}', [App\Http\Controllers\QuestionController::class, 'view'])->name('question');

Создаем новый метод view в QuestionController.php. С помощью findOrFail уже выбираем не коллекцию, а одну запись:

class QuestionController extends Controller

{

    public function index()

    {

        $questions = Question::withCount('variants')->withCount('answers')->get();

        return view('questions', [

            'questions' => $questions,

        ]);

    }

    public function view($id)

    {

        $question = Question::findOrFail($id);

        return view('question', [

            'question' => $question,

        ]);

    }

}

Создаем представление question.blade.php:

@extends('layouts.app')

@section('content')

<div class="container">

    <h1 class="mb-4">

    <a href="{{ route('questions') }}">Questions</a> /

    Question #{{ $question->id }}

    </h1>

    <hr>

    <p class="mb-4">Author: {{ $question->user->name }}</p>

    <h5 class="mb-4">{{ $question->text }}</h5>

    <div class="table-responsive">

        <table class="table table-hover">

            <tr>

                <th>#</th>

                <th>Variant</th>

                <th>Answers</th>

            </tr>

            @foreach($question->variants as $key => $variant)

            <tr>

                <td>{{ $key + 1 }}</td>

                <td>{{ $variant->text }}</td>

                <td>

                    @forelse($variant->answers as $answer)

                    <div>{{ $answer->user->name }}</div>

                    @empty

                    <small class="text-muted">Пока никто</small>

                    @endforelse

                </td>

            </tr>

            @endforeach

        </table>

    </div>

</div>

@endsection

Forelse аналогичен foreach, но можно использовать @empty, чтобы указать, если список пуст.

И добавим ссылку в questions.blade.php. Изменим код второй ячейки таким образом:

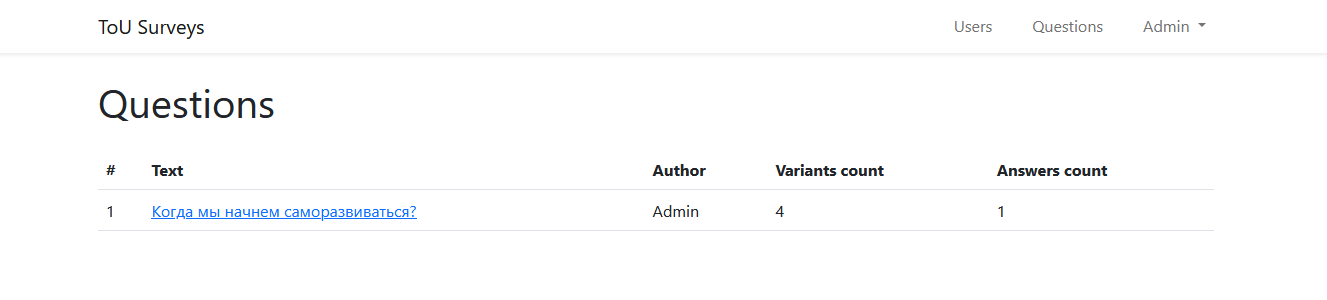
<td>

     <a href="{{ route('question', $question->id) }}">

         {{ $question->text }}

     </a>

</td>



Страница вопроса будет выглядеть так:

Изображение выглядит как текст

Автоматически созданное описание

Теперь аналогично добавим страницу пользователя. Создадим маршрут:

Route::get('/users/{id}', [App\Http\Controllers\UserController::class, 'view'])->name('user');

Метод view в UserController.php:

class UserController extends Controller

{

    public function index()

    {

        $users = User::withCount('questions')->withCount('answers')->get();

        return view('users', [

            'users' => $users,

        ]);

    }

    public function view($id)

    {

        $user = User::findOrFail($id);

        return view('user', [

            'user' => $user,

        ]);

    }

}

Теперь нужно создать представление user.blade.php. В одной части выведем его вопросы, в другой ответы:

@extends('layouts.app')

@section('content')

<div class="container">

    <h1 class="mb-4">

    <a href="{{ route('users') }}">Users</a> / {{ $user->name }}</h1>

    <hr>

    <h5 class="mb-4">Email: {{ $user->email }}</h5>

    <h5 class="mb-4">Registration date: {{ date('d.m.Y', strtotime($user->created\_at)) }}</h5>

    <div class="row">

        <div class="col-md-6 mb-4">

            <div class="card">

                <div class="card-body">

                    <h4 class="mb-4">Вопросы</h4>

                    @forelse($user->questions as $question)

                    <a href="{{ route('question', $question->id) }}">{{ $question->text }}</a>

                    @empty

                    <div class="text-muted">Пока нет вопросов</div>

                    @endforelse

                </div>

            </div>

        </div>

        <div class="col-md-6 mb-4">

            <div class="card">

                <div class="card-body">

                    <h4 class="mb-4">Ответы</h4>

                    @forelse($user->answers as $answer)

                    <a href="{{ route('question', $answer->question->id) }}">{{ $answer->question->text }}</a> - {{ $answer->variant->text }}

                    @empty

                    <div class="text-muted">Пока нет ответов</div>

                    @endforelse

                </div>

            </div>

        </div>

    </div>

</div>

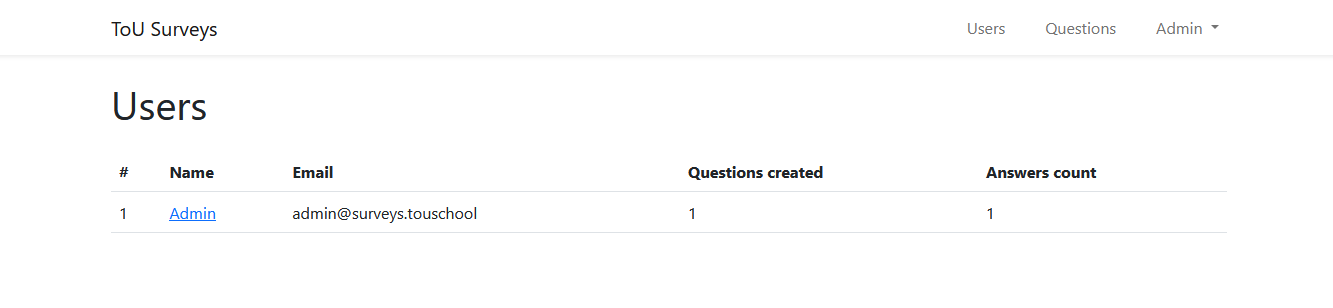
@endsection

И в users.blade.php (список пользователей) также добавить ссылку на каждого пользователя:

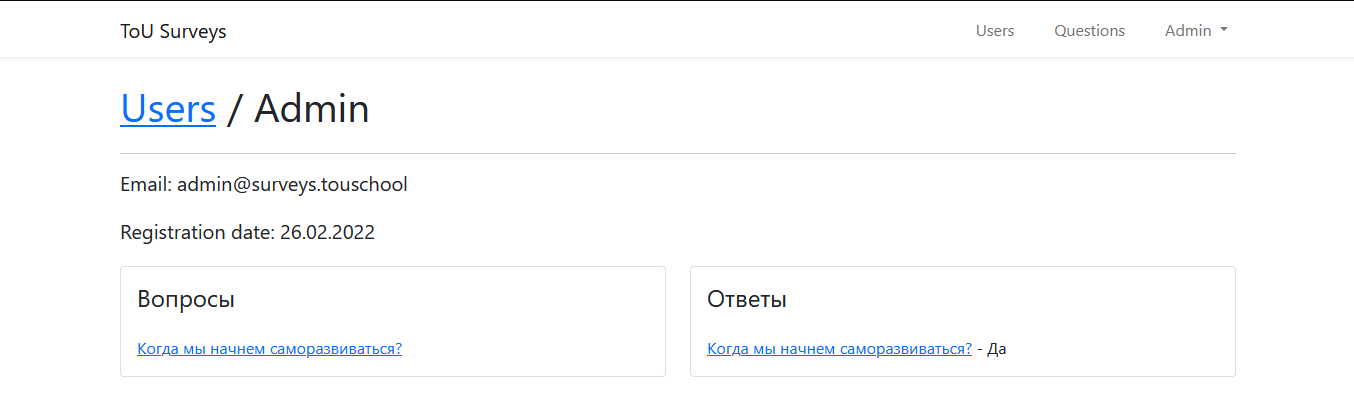
<td>

   <a href="{{ route('user', $user->id) }}">{{ $user->name }}</a>

</td>



Страница пользователя выглядит так:



Ссылки на вопросы кликабельны.

При желании доработать, добавив вывод даты ответов или даты создания вопроса.