Здесь мы создадим API для использования его в клиентском приложении, это продолжение проекта, начатого в предыдущем уроке. У вас уже должны быть обязательно выполнены следующие действия:

- Настроен .htaccess

- Импортирована БД и прописаны данные для подключения в .htaccess

- Созданы модели Question, Answer, Variant, User.

Статья о REST API:

https://habr.com/ru/post/483202/

Готовый проект по этой методичке:

https://github.com/afanasyevadina/tousurveysbackend

Для тестирования запросов понадобится скачать Postman (программу или расширение для браузера).

Какие бывают HTTP методы – GET, POST, PUT, PATCH, DELETE.

Разница между GET и POST запросами в том, что при отправке POST запроса можно передать body – некоторые данные. Аналогично работают и PUT с PATCH, но можно обойтись и POST.

Для нашего API спецификация следующая:

<table>
<tr><td colspan="2">Авторизация
AuthController@login</td></tr>
<tr><td>POST /login
{
"email": "admin@surveys.touschool",
"password": "admin",
}</td><td>200 Success
{
"data": {
"email": "admin@surveys.touschool",
"name": "Admin",
"api_token": "5oN7HPGJlhGcLL7J1lovuATBIY5IAvVKPJ8WGESUlB6MB"
}
}
Or
403
{
"errors": {
"error": "Unauthorized"
}
}</td></tr>
<tr><td colspan="2">Регистрация
AuthController@register</td></tr>
<tr><td>POST /register
{
"name": "Simple user",</td><td>200 Success
{
"data": {
"email": "user@surveys.touschool",
"name": "Simple user",</td></tr>
</table>

<table>
<tr><td>

```
    "email": "
user@surveys.touschool"
,
    "password": "
user",
}
```

</td><td>

```
        "api_token": "TGJ5kZVsCCxuQLZwuWkcIFjkclcE20vtP5B96IpeLQCPA"
    }
}
```

</td></tr>
<tr><td colspan="2">Получить список опросов
QuestionController@index</td></tr>
<tr><td>GET /questions</td><td>200 Success

```
{
    "data": [
        {
            "id": 1,
            "text": "Когда мы начнем саморазвиваться?",
            "answers_count": 1,
            "has_answer ": true,
            "user": {
                "id": 1,
                "name": "Admin"
            }
        }
    ]
}
```

</td></tr>
<tr><td colspan="2">Получить опрос по ИД
QuestionController@show</td></tr>
<tr><td>GET /questions /1</td><td>200 Success

```
{
    "data": {
        "id": 1,
        "text": "Когда мы начнем саморазвиваться?",
        "answers_count": 1,
        "has_answer ": true,
        "user": {
            "id": 1,
            "name": "Admin"
        },
        "variants": [
            {
                "id": 1,
                "text": "С понедельника"
            },
            {
                "id": 2,
                "text": "С 1 апреля"
            },
            {
                "id": 3,
                "text": "Я уже закончил"
            },
            {
                "id": 4,
                "text": "Да"
```

</td></tr>
</table>

<table>
<tr><td></td><td><pre>
            }
        ]
    }
}
</pre></td></tr>
<tr><td colspan="2">Создать опрос
QuestionController@store</td></tr>
<tr><td>POST /questions
<pre>
{
    "text": "Вы что-то поняли?",
    "variants": [
        {
            "text": "Да"
        },
        {
            "text": "Нет"
        },
        {
            "text": "Кто я?"
        }
    ]
}
</pre></td><td>201 Created</td></tr>
<tr><td colspan="2">Ответ на опрос
AnswerController@store</td></tr>
<tr><td>POST /questions/1/answer
<pre>
{
    "variant_id": "2"
}
</pre></td><td>201 Created</td></tr>
<tr><td colspan="2">Результаты опроса
AnswerController@show</td></tr>
<tr><td>GET /questions/1/results</td><td>200 Success
<pre>
{
    "data": {
        "id": 2,
        "text": "Вы что-то поняли?",
        "user": {
            "id": 1,
            "name": "Admin"
        },
        "variants": [
            {
                "id": 5,
                "text": "Да",
                "users": [
                    {
                        "id": 2,
                        "name": "Simple user"
</pre></td></tr>
</table>

<table>
<tr>
<td></td>
<td>
<pre>
                    }
                ]
            },
            {
                "id": 6,
                "text": "Нет",
                "users": [
                    {
                        "id": 1,
                        "name": "Admin"
                    }
                ]
            },
            {
                "id": 7,
                "text": "Кто я?",
                "users": []
            }
        ]
    }
}
</pre>
</td>
</tr>
</table>

Сообщения об ошибках валидации выглядят так:

Статус: 422

```
{
    "errors": {
        "password": "The password field is required."
    }
}
```

Ответ для неавторизованного пользователя такой:

Статус: 403

```
{
    "errors": {
        "error": "Unauthorized"
    }
}
```

Тело запроса (body) передается в формате JSON. Авторизация через GET параметр api_token или заголовок Bearer token. У нас будет 3 контроллера для API. Все маршруты, кроме логина и регистрации доступны только через авторизацию

Начнем с авторизации. Добавим маршруты в routes/api.php:

```
Route::post('/login', [App\Http\Controllers\Api\AuthController::class,
'login']);
Route::post('/register', [App\Http\Controllers\Api\AuthController::class,
'register']);
```

Перейдем в консоли в папку с нашим проектом и выполним:

```
php artisan make:controller Api/AuthController
```

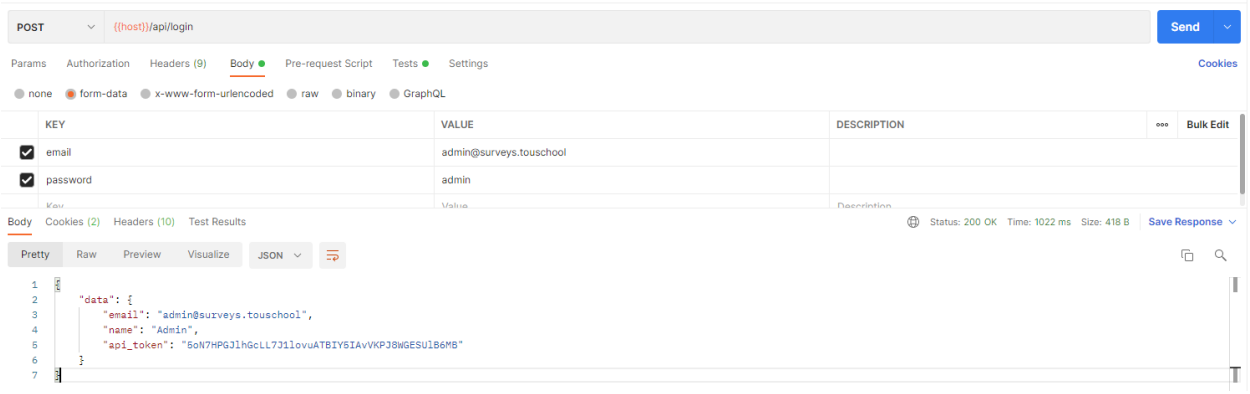
напишем метод login:

```
namespace App\Http\Controllers\Api;

use App\Http\Controllers\Controller;
use App\Models\User;
use Illuminate\Http\Request;
use Illuminate\Support\Facades\Hash;
use Illuminate\Support\Facades\Validator;
use Illuminate\Support\Str;


class AuthController extends Controller
{
    public function login(Request $request)
    {
        $validator = Validator::make($request->all(), [
            'email' => ['required', 'string'],
            'password' => ['required', 'string'],
        ]);
        if($validator->fails()) {
            return response()->json([
                'errors' => collect($validator->errors())->map(function
($item) {
                    return $item[0];
                }),
            ])->setStatusCode(422);
        }
        $user = User::where('email', $request->email)->first();
        if($user && Hash::check($request->password, $user->password)) {
            auth()->login($user);
            return response()->json([
                'data' => auth()->user()->only(['email', 'name',
'api_token']),
            ]);
        }
        return response()->json([
            'errors' => [
                'error' => 'Unauthorized',
            ]
        ])->setStatusCode(403);
    }
}
```

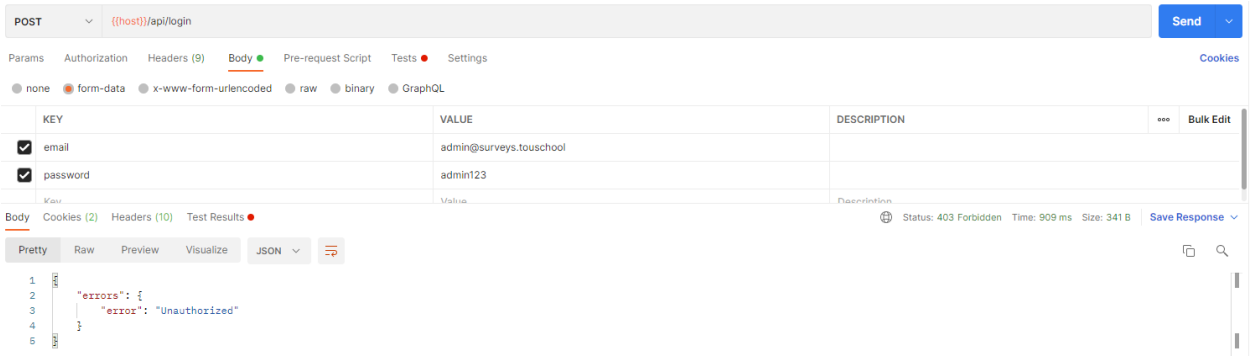
Во-первых, происходит валидация с помощью Validator. Параметры email и password обязательные. Ошибки валидации возвращаются в форматированном виде как по спецификации.

Далее ищем пользователя с таким email и если он есть, сверяем хэши паролей. Если все совпало, логиним и возвращаем данные. Иначе кидаем ошибку – неавторизован.

При верных данных:



При неверном пароле:



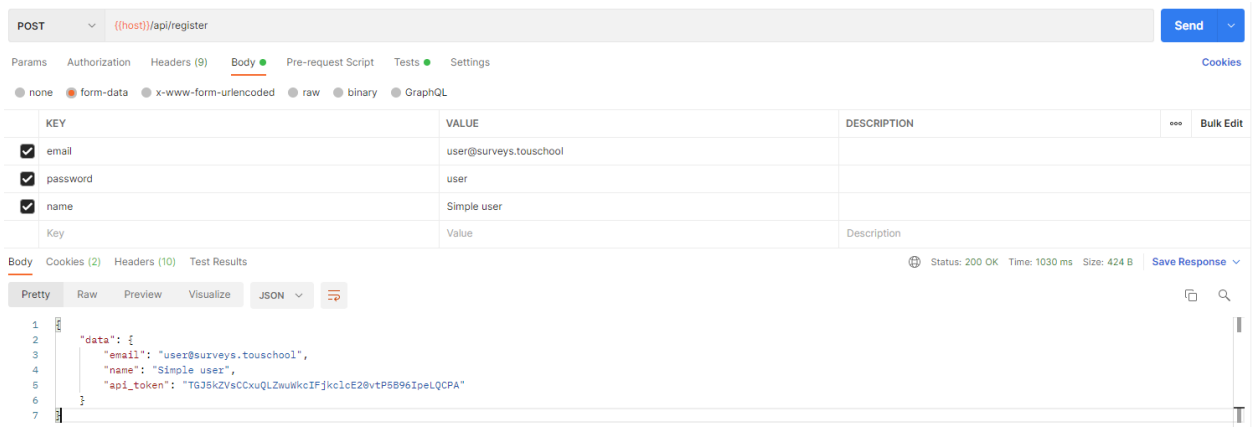
Создадим метод регистрации:

Api/AuthController.php:

```php
public function register(Request $request)
{
    $validator = Validator::make($request->all(), [
        'name' => ['required', 'string'],
        'email' => ['required', 'string', 'unique:users'],
        'password' => ['required', 'string'],
    ]);
    if($validator->fails()) {
        return response()->json([
            'errors' => collect($validator->errors())->map(function ($item) {
                return $item[0];
            }),
        ])->setStatusCode(422);
    }
    $user = User::create([
        'name' => $request->name,
        'email' => $request->email,
        'password' => Hash::make($request->password),
        'api_token' => Str::random(45),
    ]);
    auth()->login($user);
    return response()->json([
        'data' => auth()->user()->only(['email', 'name', 'api_token']),
    ]);
}
```

Здесь email должен быть уникальным. При сохранении пользователя пароль хэшируется, а api_token генерируется как рандомная строка.

Проверим:



Из ответа мы извлекаем *api_token*. Через него и будет происходить авторизация. Авторизация в API происходит путем передачи токена в заголовке *Authorization: Bearer *token**, или преедачей GET-параметра *api_token*, например */questions?api_token=123*.

Также нужно добавить настройку в config/auth.php, изменить поле guards таким образом:

```
'guards' => [
    'web' => [
        'driver' => 'session',
        'provider' => 'users',
    ],
    'api' => [
        'driver' => 'token',
        'provider' => 'users',
    ],
],
```

Определим все планируемые маршруты в routes/api.php. Эти маршруты нужно обернуть в middleware *auth:api*, чтобы доступны были только через авторизацию.

```
Route::group(['middleware' => 'auth:api'], function () {
    Route::get('/questions',
[App\Http\Controllers\Api\QuestionController::class, 'index']);
    Route::get('/questions/{id}',
[App\Http\Controllers\Api\QuestionController::class, 'show']);
    Route::post('/questions',
[App\Http\Controllers\Api\QuestionController::class, 'store']);
    Route::post('/questions/{id}/answer',
[App\Http\Controllers\Api\AnswerController::class, 'store']);
    Route::get('/questions/{id}/results',
[App\Http\Controllers\Api\AnswerController::class, 'show']);
});
```

Напишем обработчик для неавторизованных запросов в App/Http/Middleware/Authenticate.php. Модифицируем его так, чтобы для запросов API возвращался ответ в формате JSON:

```
namespace App\Http\Middleware;

use Illuminate\Auth\Middleware\Authenticate as Middleware;
use Illuminate\Http\Exceptions\HttpResponseException;

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->is('api/*')) {
            return route('login');
        }
        throw new HttpResponseException(response()->json([
            'errors' => [
                'error' => 'Unauthorized',
            ]
        ])->setStatusCode(403));
    }
}
```

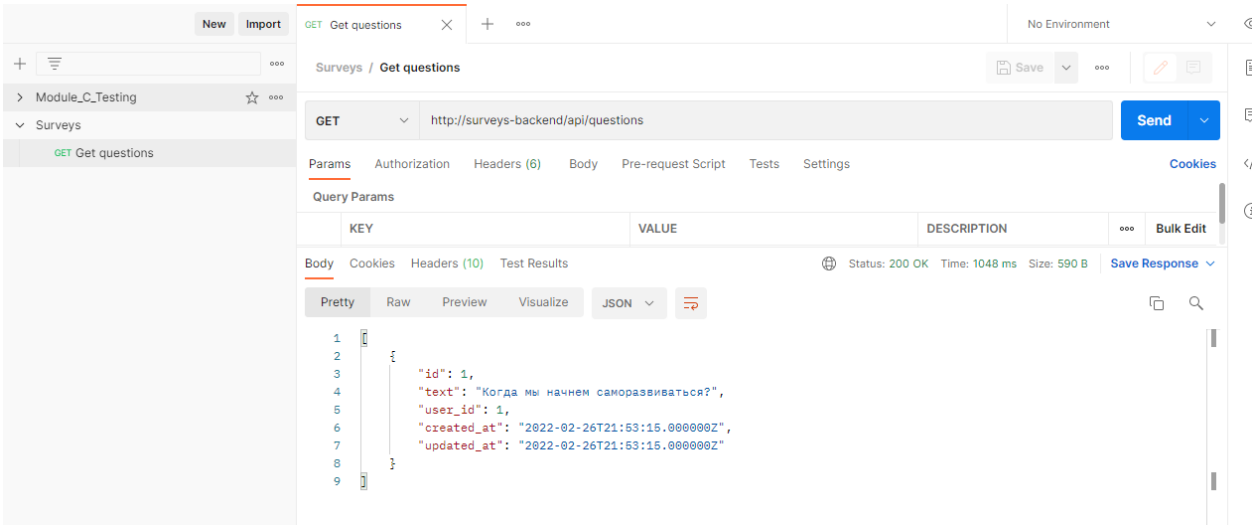
Итак, контроллер для работы с опросами будет Api\QuestionController. Создадим метод, возвращающий все опросы.

```
php artisan make:controller Api/QuestionController
```

Создадим метод index, возвращающий все опросы в формате JSON. Вернем отсортированными таким образом, чтобы сначала шли последние.

```
namespace App\Http\Controllers\Api;

use App\Http\Controllers\Controller;
use Illuminate\Http\Request;
use App\Models\Question;
use Illuminate\Support\Facades\Validator;

class QuestionController extends Controller
{
    public function index()
    {
        return response()->json(Question::latest()->get());
    }
}
```

По этому запросу получаем:



Работает. Но нам необходимо форматировать вывод. Мы можем сделать это прямо в контроллере, но грамотным будет вынести в отдельный класс ресурса. Так и будем делать для последующих роутов. Выполним:

```
php artisan make:resource QuestionResource
```

и в созданном файле App/Http/Resources/QuestionResource.php модифицируем код следующим образом:

```php
class QuestionResource extends JsonResource
{
    /**
     * Transform the resource into an array.
     *
     * @param  \Illuminate\Http\Request  $request
     * @return
array|\Illuminate\Contracts\Support\Arrayable|\JsonSerializable
     */
    public function toArray($request)
    {
        return [
            'id' => $this->id,
            'text' => $this->text,
            'has_answer' => $this->has_answer,
            'answers_count' => $this->answers()->count(),
            'user' => [
                'id' => $this->user->id,
                'name' => $this->user->name,
            ],
        ];
    }
}
```

Этот класс отвечает за форматирование ответа, добавляя только нужные поля. Также здесь подтягивается информация об авторе.

Также нужно добавить вычисляемое поле has_answer, которое означает, ответил ли уже пользователь на этот вопрос. Добавим метод в модель Question.php:

```php
public function getHasAnswerAttribute()
{
    return $this->answers()->where('user_id', auth()->id())->exists();
}
```

Дословно можно прочитать как «существует ответ, где user_id соответствует id текущего пользователя».

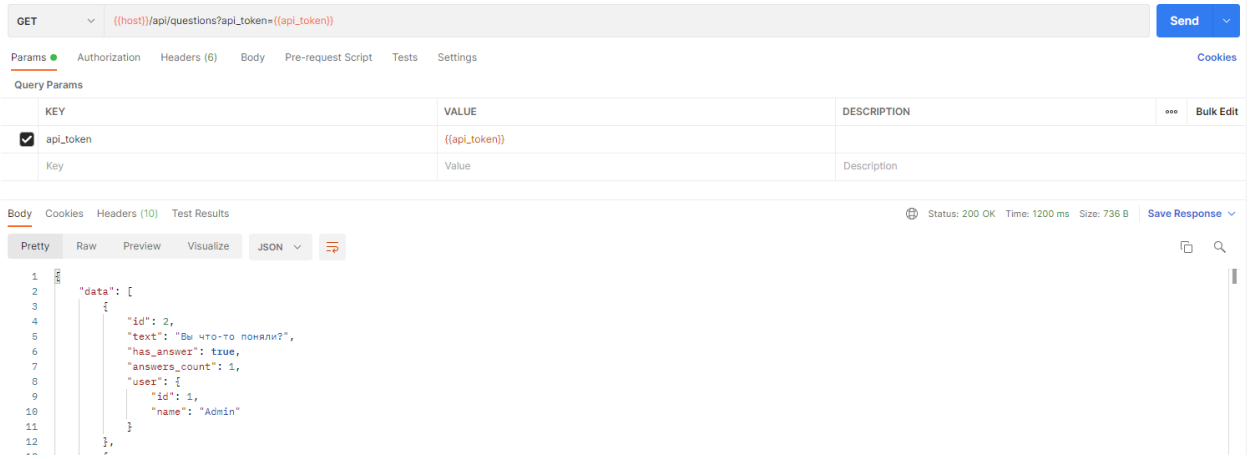
Модифицируем Api/QuestionController, изменив метод index:

```
public function index()
{
    return QuestionResource::collection(Question::latest()->get());
}
```

Также не забудьте добавить перед классом.

```
use App\Http\Resources\QuestionResource;
```

Теперь ответ выглядит как в задании:



Создадим класс ресурса для метода show:

```
php artisan make:resource QuestionDetailResource
```

```
class QuestionDetailResource extends JsonResource
{
    /**
     * Transform the resource into an array.
     *
     * @param  \Illuminate\Http\Request  $request
     * @return
array|\Illuminate\Contracts\Support\Arrayable|\JsonSerializable
     */
    public function toArray($request)
    {
        return [
            'id' => $this->id,
            'text' => $this->text,
            'has_answer' => $this->has_answer,
            'answers_count' => $this->answers()->count(),
            'user' => [
                'id' => $this->user->id,
                'name' => $this->user->name,
            ],
            'variants' => $this->variants->map(function($variant) {
```

```
                return [
                    'id' => $variant->id,
                    'text' => $variant->text,
                ];
            }),
        ];
    }
}
```

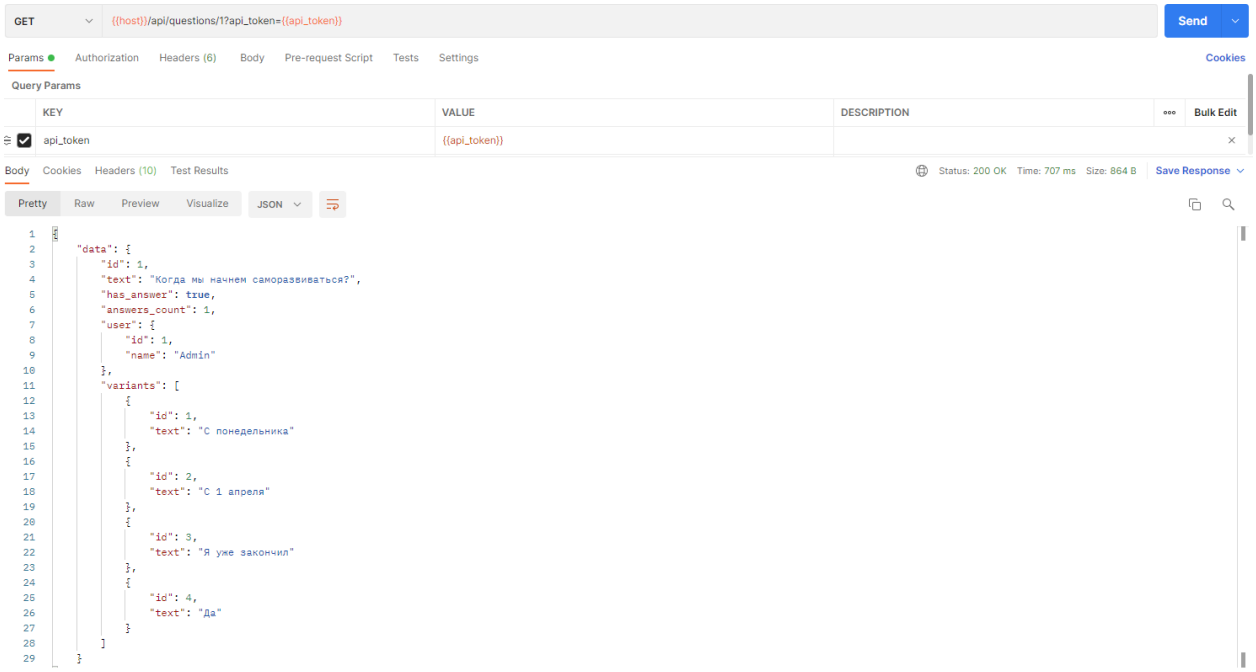
И метод show в нашем контроллере:

```
use App\Http\Resources\QuestionDetailResource;
```

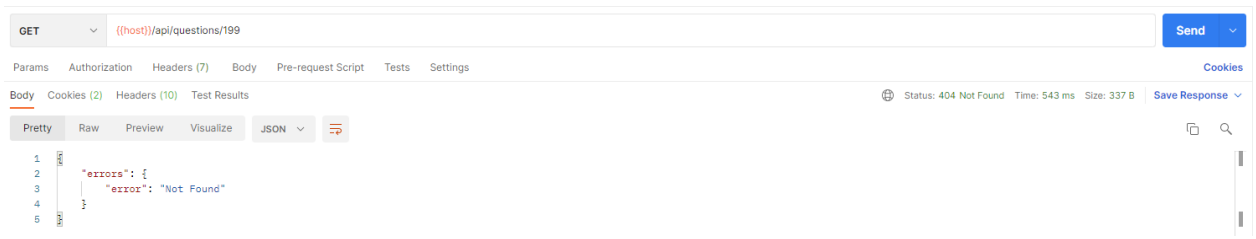
...

```
public function show($id)
{
    $question = Question::find($id);
    if(!$question)
        return response()->json([
            'errors' => [
                'error' => 'Not Found',
            ],
        ])->setStatusCode(404);
    return QuestionDetailResource::make($question);
}
```

Здесь выполняется проверка, найден ли искомый ресурс. Проверим в Postman:



Теперь попробуем передать несуществующий id, например 199:

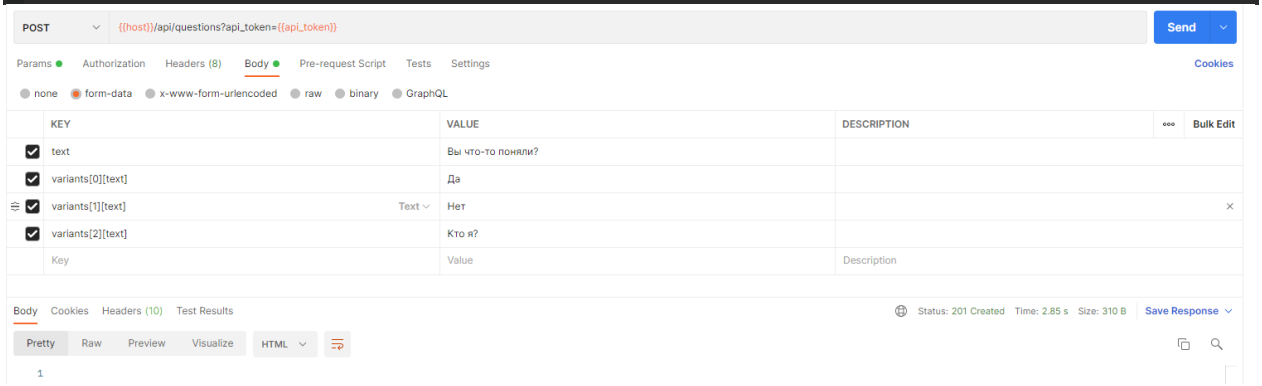


Все работает.

Перейдем к созданию опроса.

Вернемся в QuestionController и добавим третий метод store:

```
public function store(Request $request)
{
    $validator = Validator::make($request->all(), [
        'text' => ['required', 'string'],
        'variants' => ['required', 'array'],
        'variants.*.text' => ['required', 'string'],
    ]);
    if($validator->fails()) {
        return response()->json([
            'errors' => collect($validator->errors())->map(function ($item) {
                return $item[0];
            }),
        ])->setStatusCode(422);
    }
    $question = auth()->user()->questions()->create([
        'text' => $request->text,
    ]);
    foreach($request->variants as $variant) {
        $question->variants()->create($variant);
    }
    return response()->noContent()->setStatusCode(201);
}
```



Здесь уже валидация массива. Каждый объект массива variants должен иметь свойство text. В контроллере создаем вопрос, потом по циклу создаем варианты и все это сохраняем. С помощью запроса /questions и /questions/2 можно проверить, что данные действительно появились.

Далее будем писать функционал по добавлению ответа. Он тоже защищен от неавторизованного доступа. Создадим контроллер:

```
php artisan make:controller Api/AnswerController
```

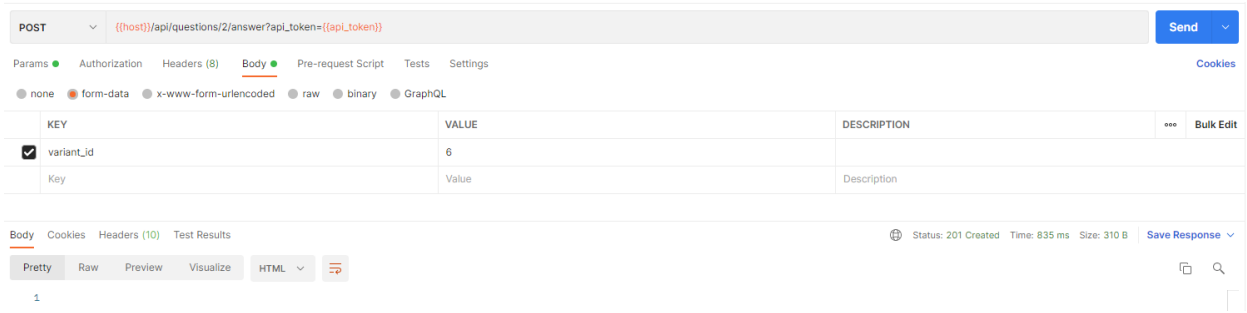
Через request body передается только variant_id выбранного ответа. Нужно проверить, что поле не пусто и что данный опрос имеет вариант с таким id. Реализуем валидатор так:

```
class AnswerController extends Controller
{
    public function store(Request $request, $id)
    {
        $question = Question::find($id);
        if(!$question)
```

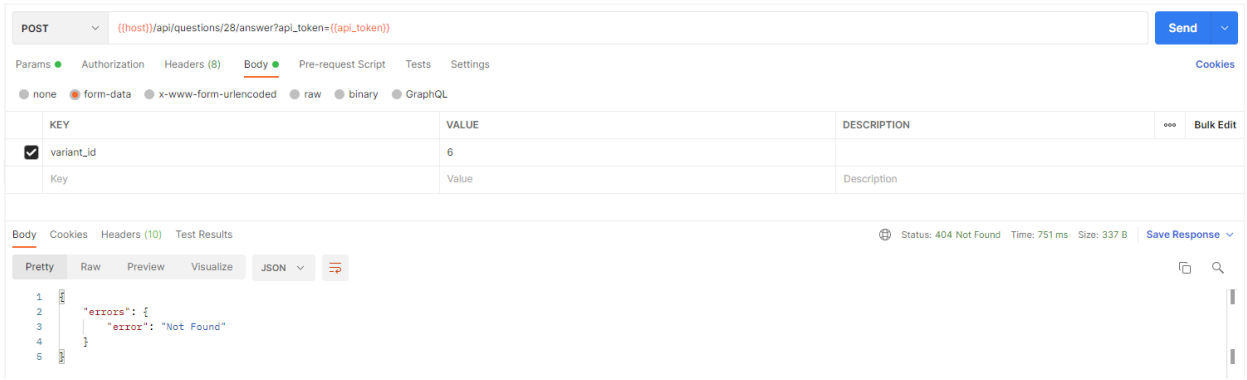
```
            return response()->json([
                'errors' => [
                    'error' => 'Not Found',
                ],
            ])->setStatusCode(404);
        $validator = Validator::make($request->all(), [
            'variant_id' => ['required', 'in:' . $question->variants->pluck('id')->implode(',')],
        ]);
        if($validator->fails()) {
            return response()->json([
                'errors' => collect($validator->errors())->map(function ($item) {
                    return $item[0];
                }),
            ])->setStatusCode(422);
        }
        auth()->user()->answers()->updateOrCreate([
            'question_id' => $question->id,
        ], [
            'variant_id' => $request->variant_id,
        ]);
        return response()->noContent()->setStatusCode(201);
    }
}
```

Функция updateOrCreate умная. Если у пользователя уже есть ответ с таким question_id, то он будет обновлен, иначе создан новый.

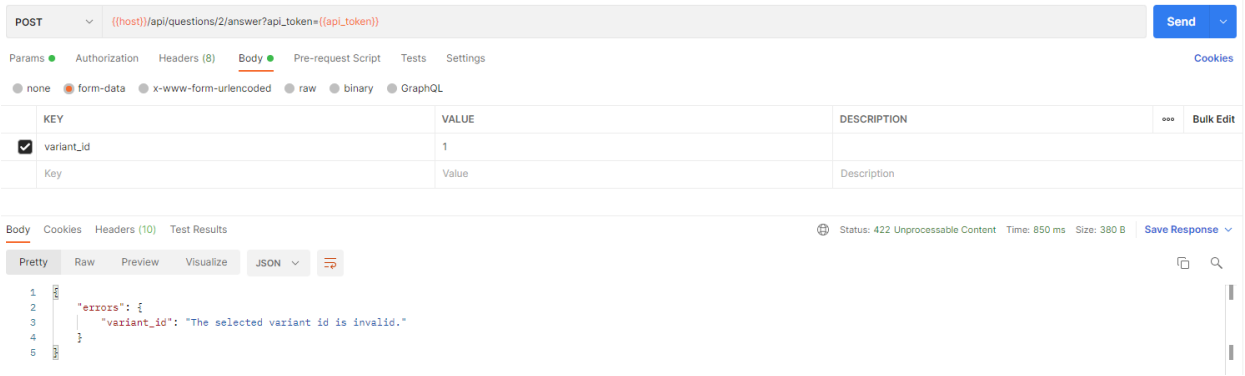
Проверим:



Попробуем передать несуществующий id вопроса:



Попробуем передать id варианта, не относящегося к вопросу:



Работает.

Следующий метод – получение результатов голосования. Сделаем так, чтобы узнать ответы можно было только после прохождения опроса.

Создадим новый ресурс для форматированной информации о результатах:

```
php artisan make:resource QuestionResultResource
```

```
class QuestionResultResource extends JsonResource
{
    /**
     * Transform the resource into an array.
     *
     * @param  \Illuminate\Http\Request  $request
     * @return
array|\Illuminate\Contracts\Support\Arrayable|\JsonSerializable
     */
    public function toArray($request)
    {
        return [
            'id' => $this->id,
            'text' => $this->text,
            'user' => [
                'id' => $this->user->id,
                'name' => $this->user->name,
            ],
            'variants' => $this->variants->map(function($variant) {
                return [
                    'id' => $variant->id,
                    'text' => $variant->text,
                    'users' => $variant->answers->map(function($answer) {
                        return [
                            'id' => $answer->user->id,
                            'name' => $answer->user->name,
                        ];
                    })
                ];
            }),
        ];
    }
}
```

Этот ресурс для каждого варианта возвращает список людей, выбравших его.

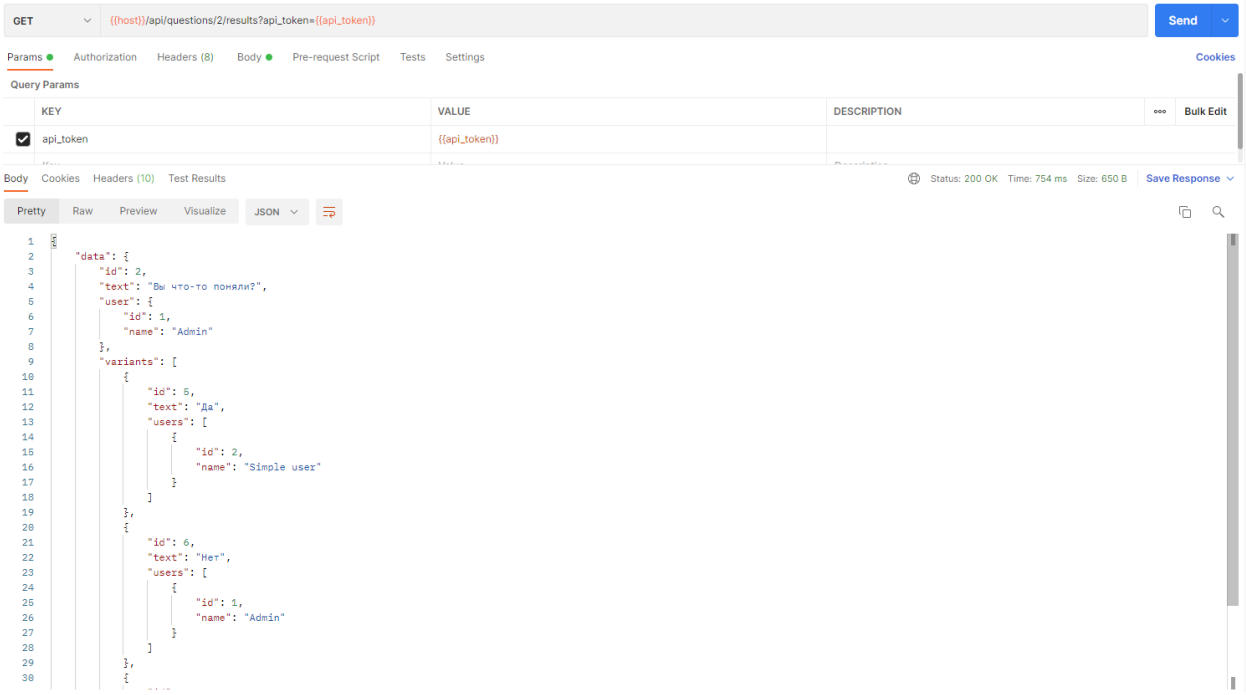
Создадим метод show в AnswerContoller:

```
use App\Http\Resources\QuestionResultResource;
```

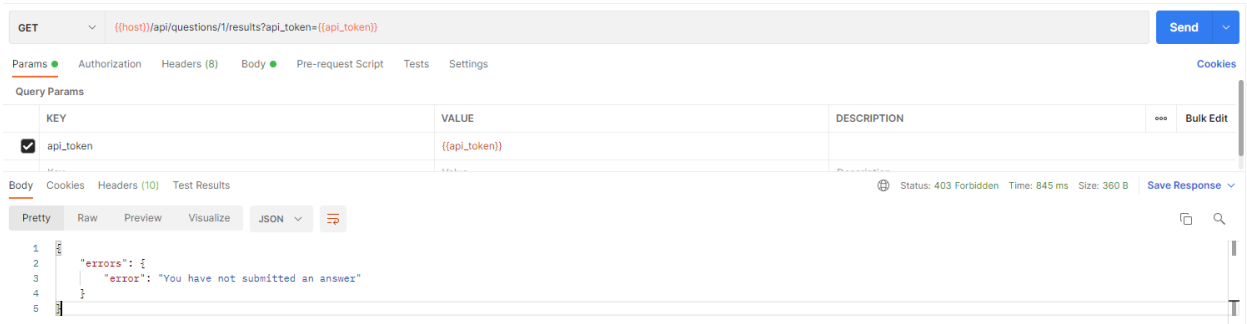
…

```php
public function show(Request $request, $id)
{
    $question = Question::find($id);
    if(!$question)
        return response()->json([
            'errors' => [
                'error' => 'Not Found',
            ],
        ])->setStatusCode(404);
    if(!$question->has_answer)
        return response()->json([
            'errors' => [
                'error' => 'You have not submitted an answer',
            ],
        ])->setStatusCode(403);
    return QuestionResultResource::make($question);
}
```

Проверим:



С другим опросом не выйдет:



Создание API завершено. Теперь можно просматривать вопросы, создавать новые и отвечать. Впоследствии этот функционал будем использовать для клиентского приложения.