

# CREATING ENDPOINT FOR SHOWING QUESTION DETAILS

If you remember, in the previous lesson we've created the show question endpoint to retrieve a question when we're going to edit our question and use it to populate the form.

Now in this lesson we're going to create another endpoint which going to use to show the details of a certain question. Unlike the previous endpoint which is protected with `auth:api` middleware, this one is going to be public so that we'll define this route outside the `auth:api` middleware.

Also for this endpoint we're going to put the logic in a brand new controller instead of in `Api/QuestionsController.php`.

## CREATE THE ENDPOINT

Alright, Let's switch over to our terminal. And then as always let's create a new branch for our work today.

```
git checkout -b lesson-50
```

### 1. Create a brand new Controller

OK, now let's generate a brand new controller for showing the question details. Let me call this controller as `QuestionDetailsController`. Let's also specify `-i` flag or `invokable` because the purpose of this controller as I said earlier is just to show the details of the question.

```
php artisan make:controller Api/QuestionDetailsController -i
```

```
zsh
~/Workspace/Laravel/laravel-qa lesson-49 git checkout -b lesson-50 ✓ 3357 21:31:26
Switched to a new branch 'lesson-50'
~/Workspace/Laravel/laravel-qa lesson-50 php artisan make:controller Api/QuestionDetailsController -i ✓ 3358 21:31:33
Controller created successfully.
~/Workspace/Laravel/laravel-qa lesson-50 ? ✓ 3359 21:32:21
```

Now let's open up the `Api/QuestionDetailsController`. At the top let's import the `Question` model namespace as well as the `QuestionResource`.

```
<?php

namespace App\Http\Controllers\Api;

use Illuminate\Http\Request;
use App\Http\Controllers\Controller;
use App\Http\Resources\QuestionResource;
use App\Question;

class QuestionDetailsController extends Controller
{
    /**
     * Handle the incoming request.
     *
     * @param  \Illuminate\Http\Request  $request
     * @return \Illuminate\Http\Response
     */
    public function __invoke(Request $request)
    {
        //
    }
}
```

In the `__invoke` method let's replace the type hint to the `Question` model. Inside the method we need to do two things. *First*, we need to increment the `views` counter. *Second*, we need to transform the question instance into `json` format by making use the `QuestionResource` class.

```
public function __invoke(Question $question)
{
    $question->increment('views');

    return new QuestionResource($question);
}
```

So again when we need to return collection of data we can making use the `ResourceClass::collection` and past the collection in. While for returning the single object we can *instantiate* the `ResourceClass` and put the instance model into it.

## 2. Define the api route

Alright, next let's open up our `api.php` file and define a new route for the question details. We're going to define it outside the `auth:api` middleware since we don't need anyone to have authenticated to access this route.

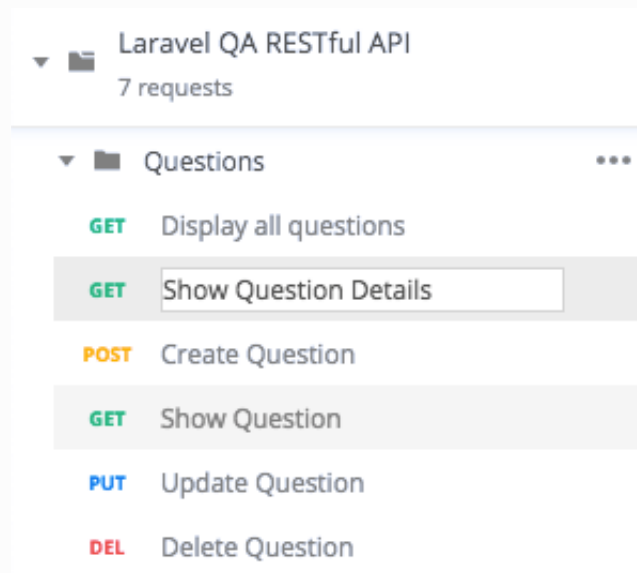
For this route we're going to use the `get` method. In the first argument we're going to make the path as `/questions/{slug}`. While in the second argument we map the path to refer to `Api/QuestionDetailsController.php`.

```
// api.php
Route::get('/questions/{slug}', 'Api\QuestionDetailsController');
Route::middleware(['auth:api'])->group(function() {
    // ...
});
```

Alright, now let's test our brand new endpoint in the Postman.

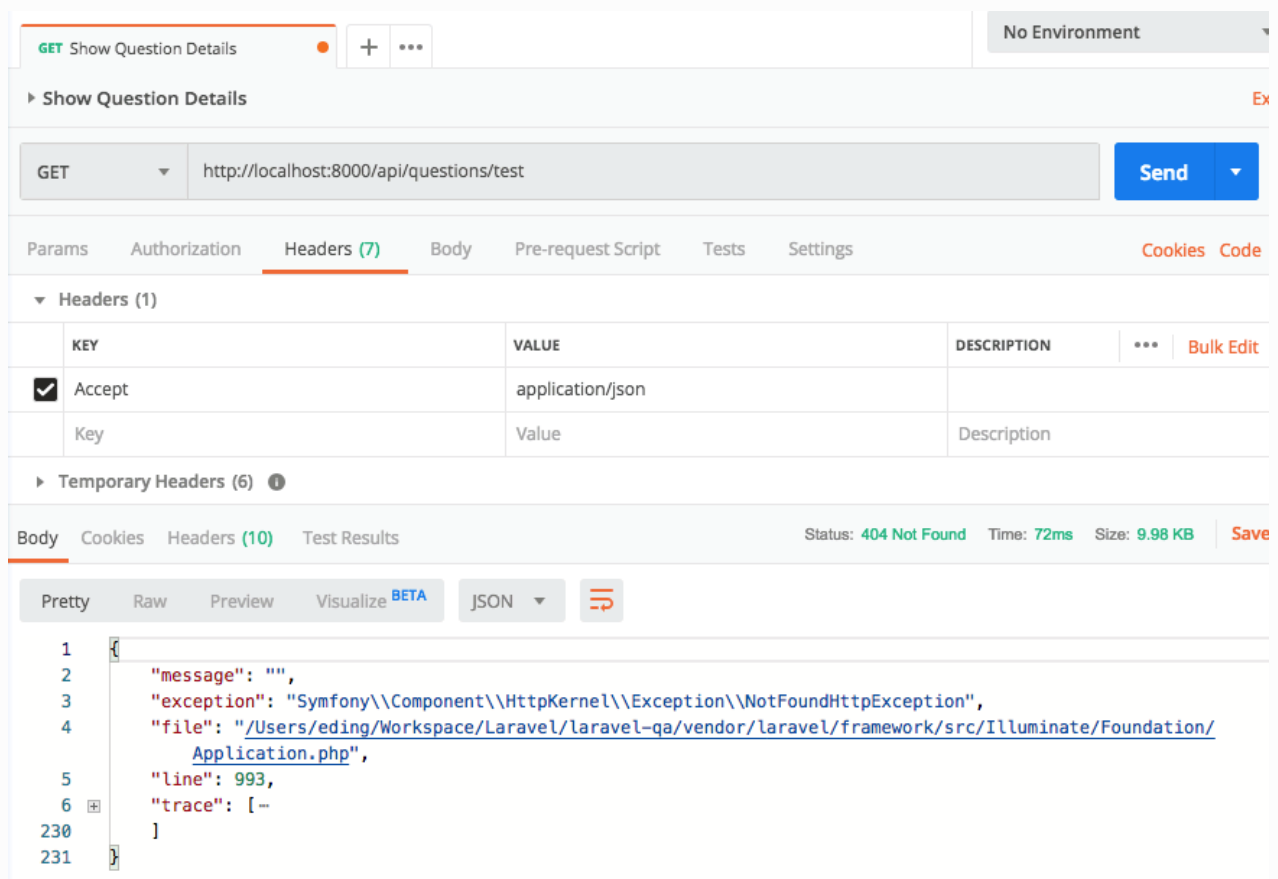
## TEST THE ENDPOINT

Let's head over to the Postman. Then duplicate the `Display all questions` test request. Rename it to `Question details`.



The HTTP verb is still gonna be `GET`. While the url it's gonna be `http://localhost:8000/api/questions/test` for now. Let's got to **Headers** tab and add `Accept` in key column and `application/json` in *value* column.

Now if I hit the **Send** button. We'll find 404 Not Found response error.



This error occurred because I don't have a question with slug "test". So let me grab a question with valid slug in tinker.

```
~/Workspace/Laravel/laravel-qa lesson-49 git checkout -b lesson-50 ✓ 3357 21:31:26
Switched to a new branch 'lesson-50'
~/Workspace/Laravel/laravel-qa lesson-50 php artisan make:controller Api/QuestionDetailsController -i ✓ 3358 21:31:33
Controller created successfully.
~/Workspace/Laravel/laravel-qa lesson-50 ? php artisan tinker ✓ 3359 21:32:21
Psy Shell v0.9.9 (PHP 7.3.4 - cli) by Justin Hileman
>>> App\Question::first()->slug
=> "updated-my-question"
>>>
```

Back to Postman. And change the `test` part in the request url with the valid slug that we grabbed from tinker. Hit the **Send** button again and now we'll get a success response which contains a single question.

The screenshot shows the Postman interface with a GET request to `http://localhost:8000/api/questions/updated-my-question`. The response status is 200 OK, and the body is a JSON object:

```
{
  "data": {
    "id": 1,
    "title": "Updated my question",
    "slug": "updated-my-question",
    "votes_count": 0,
    "answers_count": 5,
    "views": 17,
    "status": "answered-accepted",
    "excerpt": "I update my question body",
    "created_date": "3 months ago",
    "user": {
      "id": 1,
      "url": "#",
      "name": "Paige McCullough",
      "avatar": "https://www.gravatar.com/avatar/cc770b522cebaca4210aa50ce3c3b902?s=32"
    }
  }
}
```

But if you take a look at this response, we actually need a slightly different structure. If you remember, when showing question details we need to have `favorites_count`, `is_favorited` and so on.

# CREATE NEW API RESOURCE CLASS

So to solve this issue, at least you have two options:

- *First*, you can add additional values in you `QuestionResource`.
- *Second*, you can create brand new Resource API for showing question details and define which column to be exposed.

I think both option is valid, and it's up to you to choose which one you preferred. In my case I'll choose the second option.

So let's switch back over to our terminal. Then create a new resource api like so:

```
php artisan make:resource QuestionDetailsResource
```

Let's open up our `QuestionResource`. Then copy everything inside the `toArray` method. We can then open the `QuestionDetailsResource`. Then replace everything inside the `toArray` with the code that we got from `QuestionResource` class.

And let's replace unnecessary values with other values that we need. If you unsure which values to be expose, just go to question details page then identify which one that you need to expose.

```
// QuestionDetailsResource.php
class QuestionDetailsResource extends JsonResource
{
    public function toArray($request)
    {
        return [
            'id'                => $this->id,
            'title'             => $this->title,
            'votes_count'       => $this->votes_count,
            'answers_count'     => $this->answers_count,
            'is_favirited'      => $this->is_favirited,
            'favorites_count'   => $this->favorites_count,
            'body'              => $this->body,
            'body_html'         => $this->body_html,
            'user'              => new UserResource($this->user),
            'created_date'      => $this->created_date,
        ];
    }
}
```

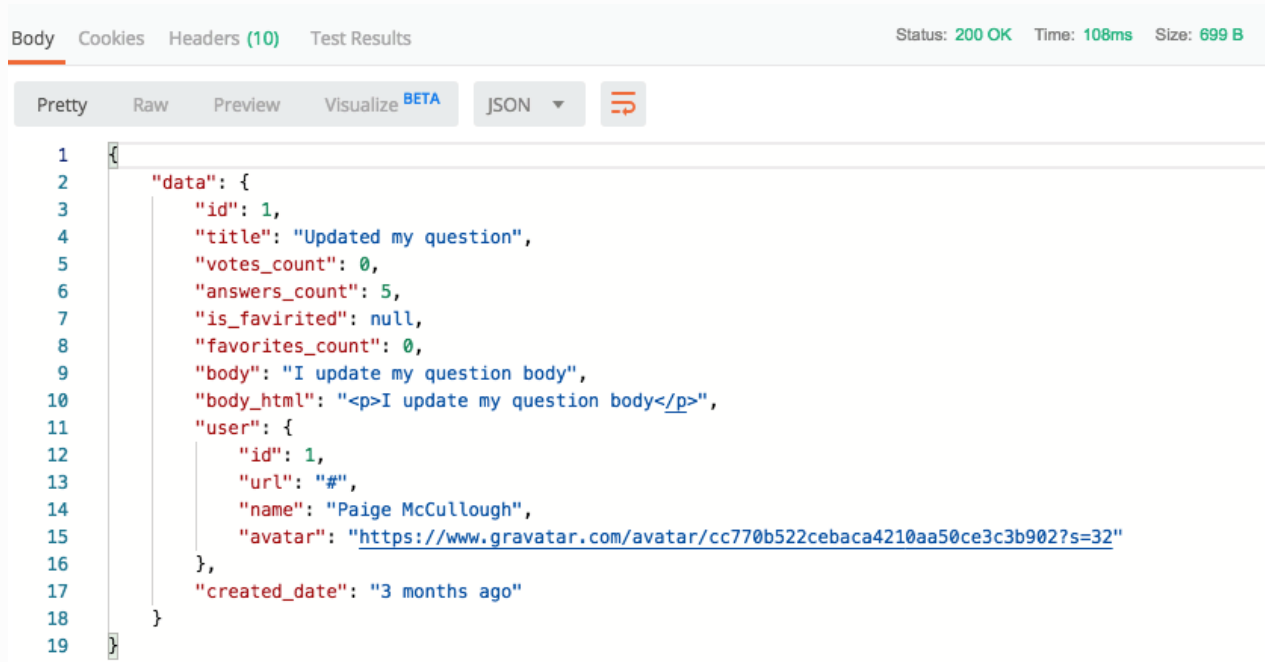
Now let's switch back over to `QuestionDetailsController`. Let's replace all `QuestionResource` in this file with `QuestionDetailsResource`.

```
# Api/QuestionDetailsController.php
// ...
use App\Http\Resources\QuestionDetailsResource;
use App\Question;

class QuestionDetailsController extends Controller
{
    public function __invoke(Question $question)
    {
        $question->increment('views');

        return new QuestionDetailsResource($question);
    }
}
```

Let's switch back over to postman. Then hit the **Send** button one more time. Now we'll get a response back with proper structure.



The screenshot shows the Postman interface with the 'Body' tab selected. The status bar at the top indicates 'Status: 200 OK', 'Time: 108ms', and 'Size: 699 B'. The response is displayed in JSON format, showing a question object with various attributes.

```
{
  "data": {
    "id": 1,
    "title": "Updated my question",
    "votes_count": 0,
    "answers_count": 5,
    "is_favorited": null,
    "favorites_count": 0,
    "body": "I update my question body",
    "body_html": "<p>I update my question body</p>",
    "user": {
      "id": 1,
      "url": "#",
      "name": "Paige McCullough",
      "avatar": "https://www.gravatar.com/avatar/cc770b522cebaca4210aa50ce3c3b902?s=32"
    },
    "created_date": "3 months ago"
  }
}
```

## A NEW ROUTE PARAMETER'S FORMAT

Before we end this lesson, let me show you one issue. If we open the `Show Question` request and run that. It now no longer work. Instead we'll find 404 Not Found error response.

The screenshot shows a web browser interface for a REST client. The URL bar shows `http://localhost:8000/api/questions/1`. The response status is `404 Not Found`. The response body is a JSON object:

```
{
  "message": "",
  "exception": "Symfony\\Component\\HttpKernel\\Exception\\NotFoundHttpException",
  "file": "/Users/eding/Workspace/Laravel/laravel-qa/vendor/laravel/framework/src/Illuminate/Foundation/Application.php",
  "line": 993,
  "trace": [
    ...
  ]
}
```

This is because when we access any url, Laravel routing system will match it with our routes definition sequentially. That's why when we try to access `/questions/1` it matches with this route: `/questions/{slug}` instead of: `/questions/{question}`. This is because the `/questions/{slug}` route comes first.

```
Route::get('/questions/{slug}', 'Api\\QuestionDetailsController');

Route::middleware(['auth:api'])->group(function() {
    Route::apiResource('/questions', 'Api\\QuestionsController')-
    >except('index');
});
```

So to fix this issue we have at least two options:

- *First*, change the route path to something else. For example instead of `/questions/{slug}` (in plurals) you can change it to `/question/{slug}` (in singular).
- *Second*, change the route parameter to other format. For example instead of `/questions/{slug}` we can change it to `/questions/{question}-{slug}`.

From these two options I'm going to use the second option because I want to make the url for the same resource (in this case Question) in uniform.

So in `api.php` let's make a change to our show question details route from `/questions/{slug}` to `/questions/{question}-{slug}` like so:

```
// api.php
Route::get('/questions/{question}-{slug}',
    'Api\\QuestionDetailsController');
```

By using this way we now no longer need to find a certain question with `slug`. Instead we use the `id`. The slug part in the url now just to beautify our url or to make it SEO friendly.

So you can optionally open the `RouteServiceProvider` and get rid of these lines:

```
Route::bind('slug', function($slug) {  
    return Question::with('user')->where('slug', $slug)->first() ??  
    abort(404);  
});
```

Also let's make a bit change in `slug` part in our `QuestionResource.php` by prepending the `id` and concatenate with the `slug`.

```
return [  
    // ...  
    'slug' => $this->id . '-' . $this->slug,  
    // ...  
];
```

Now if switch back to Postman then re-run the `Show Question` request. We'll find it's work again as before.

The screenshot shows the Postman interface for a GET request to `http://localhost:8000/api/questions/1`. The request is successful with a status of 200 OK, a time of 71ms, and a size of 442 B. The response body is displayed in JSON format:

```
{  
  "title": "Updated my question",  
  "body": "I update my question body",  
  "body_html": "<p>I update my question body</p>"  
}
```

And now we need to make a small change in `Show Question Detail` by prepending question id before the slug. If we missed that we'll get 404 Not Found error response. Otherwise we'll have an expected response.

The screenshot shows a REST client interface with a GET request to `http://localhost:8000/api/questions/1-updated-my-question`. The response status is 200 OK, with a time of 126ms and a size of 699 B. The response body is a JSON object:

```
1 {
2   "data": {
3     "id": 1,
4     "title": "Updated my question",
5     "votes_count": 0,
6     "answers_count": 5,
7     "is_favirited": null,
8     "favorites_count": 0,
9     "body": "I update my question body",
10    "body_html": "<p>I update my question body</p>",
11    "user": {
12      "id": 1,
13      "url": "#",
14      "name": "Paige McCullough",
15      "avatar": "https://www.gravatar.com/avatar/cc770b522cebaca4210aa50ce3c3b902?s=32"
16    },
17    "created_date": "3 months ago"
18  }
19 }
```

## SUMMARY

In this lesson, we looked at how we can create api endpoint for showing a question details. We've created another API Resource class for question details data transformation. And we've also learned how to fix the route collision.

Let's move on to the next lesson and look at how to create endpoints for answer resource. Let's commit all changes that we made today into our git repo.

```
git add .
git commit -m "Create api endpoint to show question details"
git push origin lesson-50
```