

CREATE QUESTION ENDPOINT

In this lesson we are going to see how to create API endpoint for creating question and how to properly test it in postman. Alright let's go ahead and open up our terminal. And create a new branch to isolate our work today:

```
git checkout -b lesson-48
```

CREATING API ENDPOINT

1. Define API Resource Route

Now let's go ahead and open up our `api.php` in `routes` directory. Then inside this file instead of define a `post` route for creating our question, let's define a resource route API by saying `Route::apiResource`. So by doing this way we no longer need to define new routes when we working with other endpoints such as `update` or `delete` question.

Also note that the `apiResource` does almost the same thing with what `Route::resource` does, but it exclude the `edit` and `create` actions.

In the first argument let's specify `/question`, and in the second argument it will refer to `Api\QuestionsController`.

Since we've already defined the `index` route in the previous lesson, so we need to exclude it in our route by chain in our route definition with `except` method and put the `index` on it.

So here our API resource route look like:

```
// routes/api.php
Route::apiResource('/questions', 'Api\QuestionsController')-
>except('index');
```

You can save the change and then verify your routes in your terminal.

Domain	Method	URI	Name	Action	Middleware
	GET HEAD	api/questions		App\Http\Controllers\Api\QuestionsController@index	api
	POST	api/questions	questions.store	App\Http\Controllers\Api\QuestionsController@store	api,auth:api
	GET HEAD	api/questions/{question}	questions.show	App\Http\Controllers\Api\QuestionsController@show	api,auth:api
	PUT PATCH	api/questions/{question}	questions.update	App\Http\Controllers\Api\QuestionsController@update	api,auth:api
	DELETE	api/questions/{question}	questions.destroy	App\Http\Controllers\Api\QuestionsController@destroy	api,auth:api
	POST	api/token		App\Http\Controllers\Auth\LoginController@getToken	api,guest
	GET HEAD	api/user		Closure	api,auth:api

2. Protect our api resource in middleware `auth:api`

Now we need to protect our question api resource route to be accessible only by authenticated user. In order to do that we can wrap it in middleware `auth:api` like this:

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

3. The `store` method

Alright, now let's open up our old `QuestionsController`. Inside that file let's go ahead and copy the `store` method.

Let's open up our `QuestionsController` inside `Api` folder. Inside this file let's replace `store` method with the code that we grabbed from our old `QuestionsController`.

Instead of returning a view we are going to return a `json` response contains a message that we can grab from the view.

```
// Api/QuestionsController.php
public function store(AskQuestionRequest $request)
{
    $request->user()->questions()->create($request->only('title',
'body'));

    return response()->json([
        'message' => "Your question has been submitted",
    ]);
}
```

Let's also return the newly created question in our json response. Since eloquent `create` method will return new object, we can assign it into a variable.

```
$question = $request->user()->questions()->create($request->only('title', 'body'));
```

Then inside the array of our json response we can add `question` as array key. For the value we can instantiate the `QuestionResource` and pass the `question` object to it.

```
// Api/QuestionsController.php
public function store(AskQuestionRequest $request)
{
    $question = $request->user()->questions()->create($request->only('title', 'body'));

    return response()->json([
        'message' => "Your question has been submitted",
        'question' => new QuestionResource($question)
    ]);
}
```

Don't forget to import the `AskQuestionRequest` namespace at the top of the file.

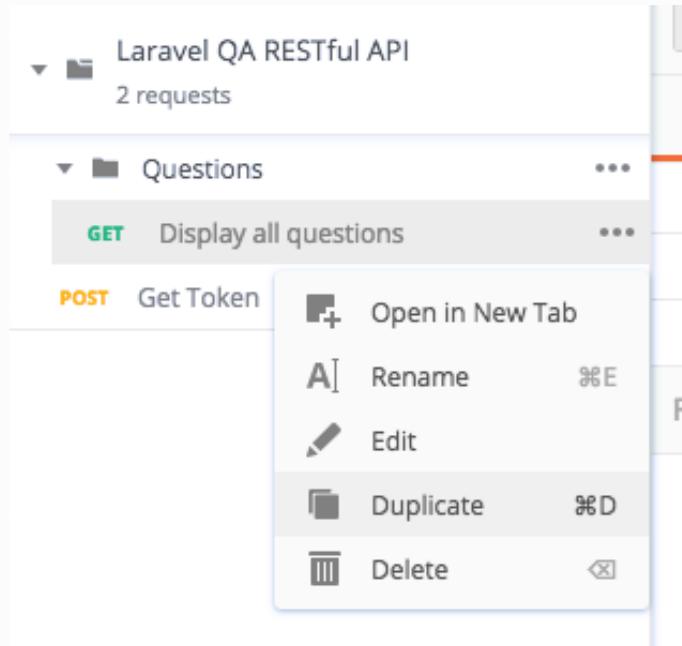
```
use App\Http\Requests\AskQuestionRequest;
```

Alright, let's save all these changes and then test our brand new endpoint in our postman.

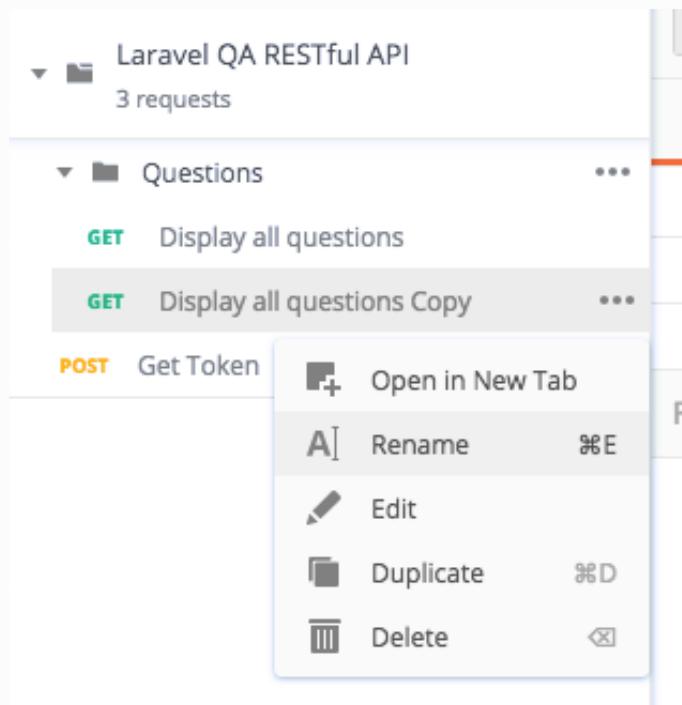
TEST OUR API ENDPOINT IN POSTMAN

1. Define a new request

In Postman let's duplicate our `Display all questions` test request.



We can then rename it as `create Question`.



In the right side we can change the HTTP verb to be `POST`, the request url is going to be the same, so we don't need to make any change on it. Let's go to *Headers* section and specify some headers. Firstly, let's add `Accept` in the *Key* column while `Application/json` in the *value*.

POST Get Token

GET Display all questions

POST Create Question

•

▶ Create Question

POST http://localhost:8000/api/questions

Params Authorization Headers (1) Body Pre-request Script Tests Settings

▼ Headers (1)

	KEY	VALUE
<input checked="" type="checkbox"/>	Accept	application/json
	Key	Value

Second, we let's specify the `Authorization` in the `key` column because remember we have put our route in `auth:api` middleware.

For the value let's grab that from **Get Token** request. Let's reopen the get token request. Hit the `Send` button, and copy the `access_token`'s value.

The screenshot shows the Postman interface with the following details:

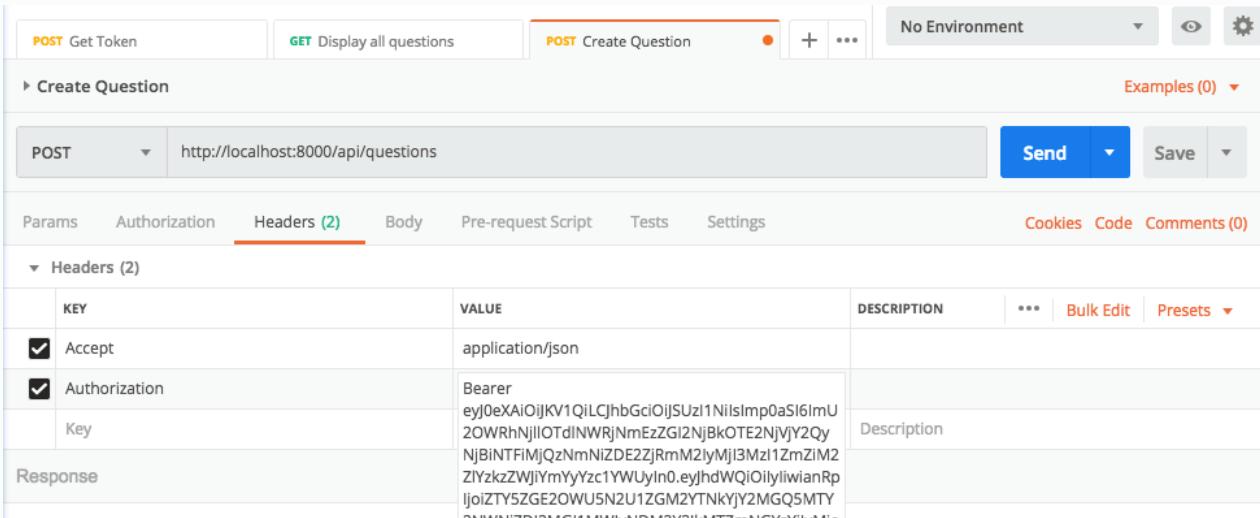
- Header Bar:** Contains tabs for "POST Get Token", "GET Display all questions", "POST Create Question", and "No Environment".
- Request Section:** A "POST" request to "http://localhost:8000/api/token".
- Params Tab:** Active tab, showing "Query Params" with a table:

KEY	VALUE	DESCRIPTION
Key	Value	Description

- Body Tab:** Active tab, showing the response body in JSON format:

```
1  {
2      "token_type": "Bearer",
3      "expires_in": 31622400,
4      "access_token":
5          "eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsImp0aSI6ImU20WRhNjll0TdlnWRjNmEzZGI2NjBk0TE2NjVjY20yNjBiNTFimJQzNmNiZDE2ZjRmM2IyMjI3MzI1ZmZmI2ZlyZkzZWjYmYyYzc1YWUyIn0.
6          eyJhdWQiOiIyIiwiZW1pbiI6Im5iZiI6MTU20Dg2MzI4MCwiZXhwIjoxNjAwNDg1NjgwLCJzdWIiOiIxiIwic2NvcGVzIjpbX0.
7          CM2y-sbwf5i0-3BW7XgreGgAscby8XjqreXfyMgad9cJ75ETPXQ9KMD9xPXEE0Rb4Whrgua1TCCurbokanVBiLzvEmX6kLvrE4YHcsIEngqIXJ1w
8          Kn3VwBp6nC4-yqGxKhYbUFLYg5x7UHD12JfgIvH-8lrr7rUQcYIbz9hydjzXSAXiu5gfa-u8w_B8T3hqbP0dp6xFmBozK_cjcJPsQwWvcgIEtT
9          0YS8cuxlFcqAVCTosGbUxjCJYCdKkWxjXPTZutjzQZW_GsguX3tjuui1WaiPzKZLk3S6d8Kc151QYLURsXq8qsXquExakz-ekqjjaXyKbGnPan
10         ZNw_bW4VUyAuyI6cinNRDratK9ok3R26xWz4Y1tRvxh3FyJ77t1c0UXBZfb0Kof1r3PHJcxGZ6NYJc2EsUewmYyVRNa3T1PUBVeks3n5yUb9
11         QZtqkV1wjf_YUrrWYhqqj1b97CZkbXB16i5RtV6vj0EAV3HpoEj9pT6DBnSXCCfkgQcm8aeJk0tlaGg8EY0tYNLzbSsb6WuBkwIdc031lBvA
12         ZrbJbqKH2jGImoE5Xn67V4tmyG3VUIGy04UqPh8RbW12MqPHL5zJ8KpE5jVxgBlBgEP0BVHtVy6pu0n9CUGUwsfg_dRgAI2DvD6sA55zlmxkGo
13         a9P_uGf6myJ",
```

Now, we can go back to `Create question` request. And for the `Authorization`'s value we can specify `Bearer`. Add a space and paste our access token.



POST Get Token GET Display all questions POST Create Question No Environment Examples (0)

POST http://localhost:8000/api/questions

Params Authorization Headers (2) Body Pre-request Script Tests Settings Cookies Code Comments (0)

Headers (2)

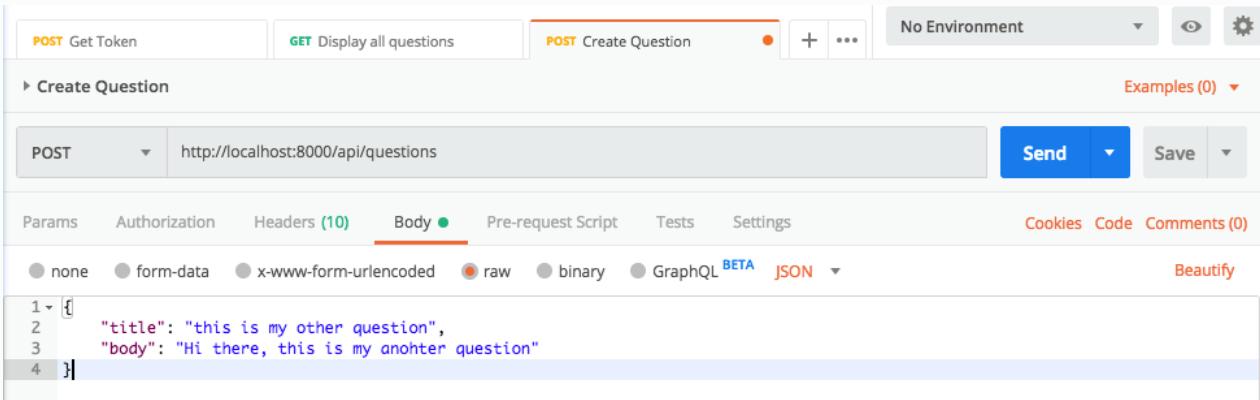
KEY	VALUE	DESCRIPTION	...	Bulk Edit	Presets
Accept	application/json				
Authorization	Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIiSlmp0aSl6ImU2OWRhNjIOTdINWRjNmEzZGI2NjBkOTE2NjVY2QyNjBINTfIMjQzNmNzDE2ZjRmM2lyMjI3MzI1ZmZlM2ZlYzkzZWJjYmYyYzc1YWUyIn0.eyJhdWQiOiIwianRpIjoiZTY5ZGE2OWU5N2U1ZGM2YTnkYjY2MGQ5MTY	Description			
Key					
Response					

2. Test the Create Question request

Now if we hit the **Send** button, we'll get validation errors which is a good sign.

```
{
  "message": "The given data was invalid.",
  "errors": {
    "title": [
      "The title field is required."
    ],
    "body": [
      "The body field is required."
    ]
  }
}
```

Now we can go to **Body** section. Choose the **raw** option and make sure you choose the **JSON** type. And then define the **title** and **body** of the question.



POST Get Token GET Display all questions POST Create Question No Environment Examples (0)

POST http://localhost:8000/api/questions

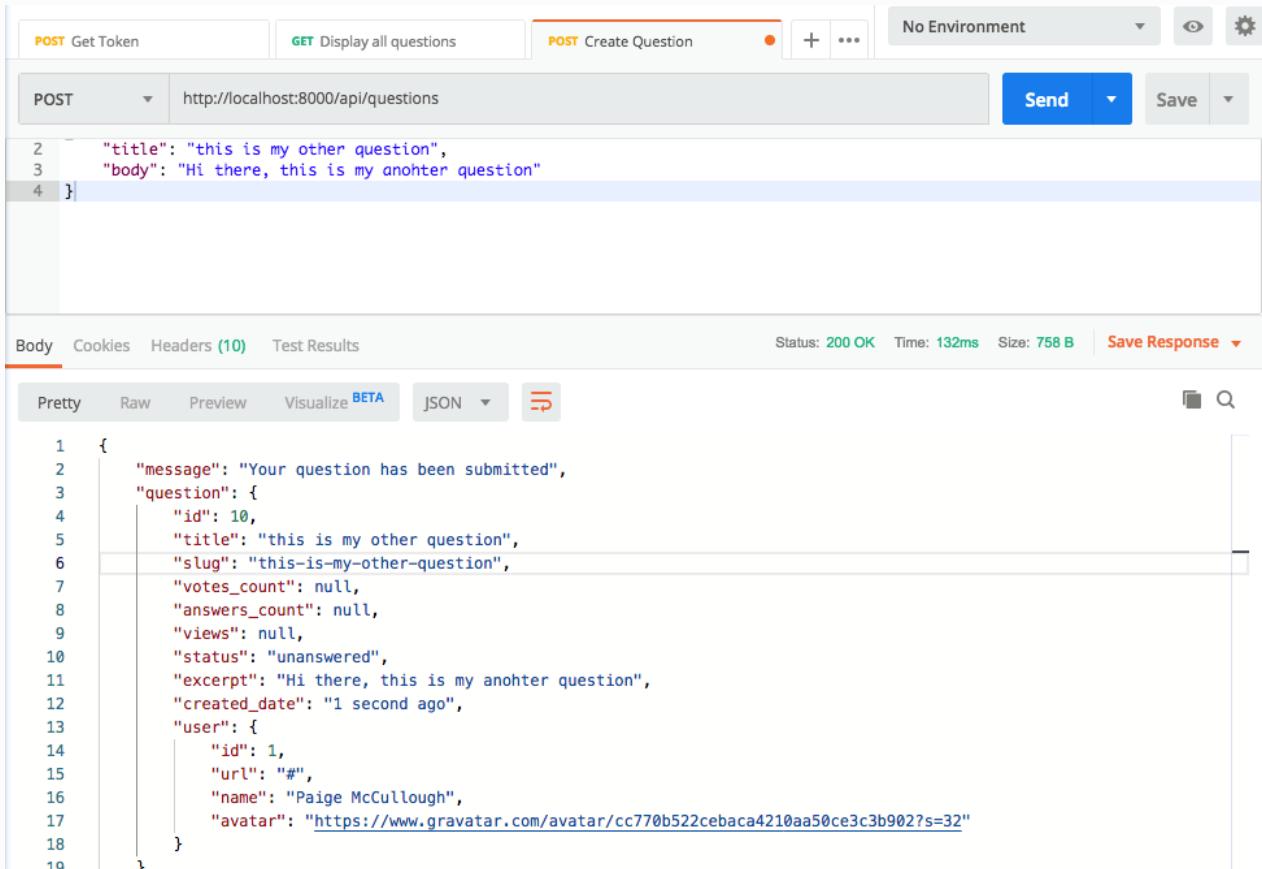
Params Authorization Headers (10) Body Pre-request Script Tests Settings Cookies Code Comments (0)

Body (10)

raw JSON Beautify

```
1 {  
2   "title": "this is my other question",  
3   "body": "Hi there, this is my another question"  
4 }
```

If we hit the **Send** button one more time. We'll have expected response which contains **message** and our newly created question.



The screenshot shows the Postman interface. At the top, there are buttons for 'POST Get Token', 'GET Display all questions', and 'POST Create Question'. The 'POST Create Question' button is highlighted with an orange border. To the right, there are buttons for 'No Environment', a plus sign, and three dots. Below the header, the method 'POST' is selected, the URL is 'http://localhost:8000/api/questions', and there are 'Send' and 'Save' buttons. The request body contains the following JSON:

```
2 "title": "this is my other question",
3 "body": "Hi there, this is my another question"
4 }
```

Below the request, the response status is 'Status: 200 OK', time is '132ms', and size is '758 B'. There are tabs for 'Body', 'Cookies', 'Headers (10)', and 'Test Results'. The 'Body' tab is selected and shows the response in 'Pretty' format:

```
1 {
2   "message": "Your question has been submitted",
3   "question": {
4     "id": 10,
5     "title": "this is my other question",
6     "slug": "this-is-my-other-question",
7     "votes_count": null,
8     "answers_count": null,
9     "views": null,
10    "status": "unanswered",
11    "excerpt": "Hi there, this is my another question",
12    "created_date": "1 second ago",
13    "user": {
14      "id": 1,
15      "url": "#",
16      "name": "Paige McCullough",
17      "avatar": "https://www.gravatar.com/avatar/cc770b522cebaca4210aa50ce3c3b902?s=32"
18    }
19  }
```

At the bottom right of the response area, there are icons for copy and search.

SUMMARY

In this lesson, we looked at how we can create api endpoint for adding new question. We've learned how to use `apiResource` to define resource api easily. And we've also learned how to hit our protected api endpoint by manually attach the access token to the request header.

Now Let's move on to the next lesson and see how to create api endpoints for updating and deleting question.

Don't forget to me commit all changes into your git repo:

```
git add .
git commit -m "Create an endpoint to create question"
git push origin lesson-48
```