

# 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	questions.store	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.

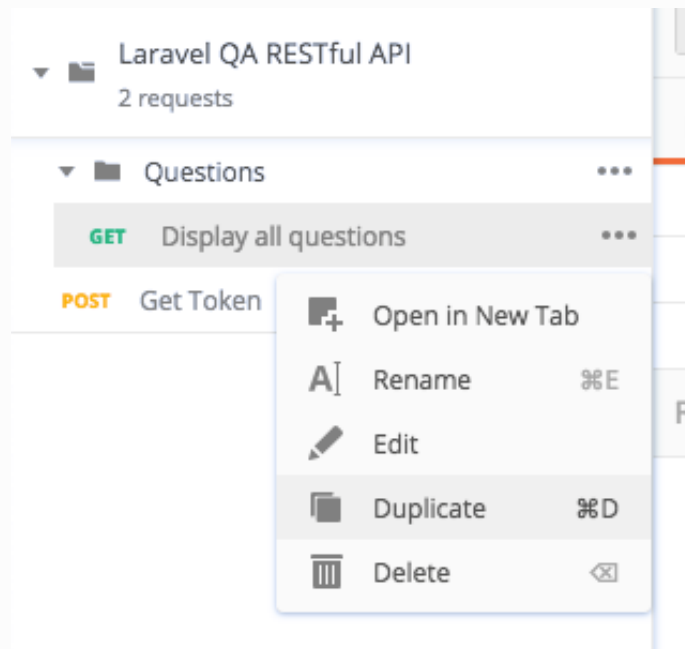
```
use App\Http\Requests\AskQuestionRequest;
```

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

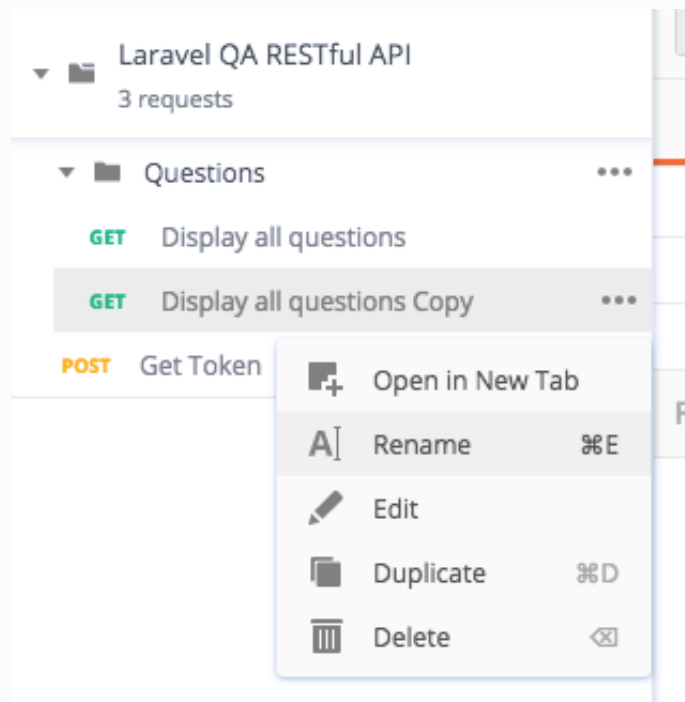
## TEST OUR API ENDPOINT IN POSTMAN

### 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.

**POST** Get Token    GET Display all questions    POST Create Question    + ...    No Environment    [icon] [gear]

---

**POST** http://localhost:8000/api/token    Send    Save

Params    Authorization    Headers (9)    Body ●    Pre-request Script    Tests    Settings    Cookies    Code    Comments (0)

### Query Params

	KEY	VALUE	DESCRIPTION	...	Bulk Edit
	Key	Value	Description		

**Body**    Cookies    Headers (11)    Test Results    Status: 200 OK    Time: 948ms    Size: 2.17 KB    Save Response ▼

Pretty    Raw    Preview    Visualize BETA    JSON ▼    [icon] [magnifying glass]

```

1 {
2   "token_type": "Bearer",
3   "expires_in": 31622400,
4   "access_token":
    eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsImpoOiSIWmUwRmhjLl0tOTdUNWRjNmEiZGI2NjBkOTE2NjVjY2QybnJlbnRfImJqZnMmNiZDE2ZjRL
    mM2IyMjI3MzI1ZmZmZWZlYzkzZWJiYmYyZnczY2YyIn0.,
    eyJhdWoiOiIyIiwianRpIjoia2YTZYSGE2OWU5NU1ZGM2YTnkYjY2MGQ5MTY2NWJjZDI2MIIMWIYNDM2Y2JkMTZmNGYzYjYyMjczMjVmZmZzZmVj
    OTNLyMjiZjJjNzhvZTIiLCJpcXkiOiJlNjg4NjMyODAsIm5iZiI6MTU0Dg2MzI4MCwiZXhwIjoxeNjAwNDg1NjgwLClJzdWIioiIxIiwic2NvcGVZ
    IjpibXX0.,
    CM2y-sbwf5i0-3BW7xgreGgAscby8XjqreXfyMgad9Cj75ETPXQ9KMd9XPXEEOrb4WHrguaITCcUrbokanVBilZEvEm6KLVrE4YHcsIEngqIXJ1w
    Kn3VvbP6nC4-yq6KhYbUFLYG5x7UHDI2jfgIvH-8lr77rUocYIBz9hydJjzXSAXiuSgfa-u8u_BBT3hqibP0dp6XFmbBozK_cjcJPsqWvwcgIEtT
    0YS8cuxlfqcAVCTosGbUxcJYckdcKwxjXPTzutjzQZW_GsguX3tjuuilWaIPzkKLZk3S6d8Kc151QLURsxq8qsXquExakzk-ekqJjaXyKbGnPAn
    ZNw_bw4VUYAuyIG6cinNDRratK9Ok3R26xWz4YT1LRvxZh3FyJ77tICOUXBZFboKof1QR3PHJgcxGZ6NYJC2EsUelwmYYvrNA3TiPUBVEks3n5yb9
    QztqvKV1wjf_YUrrwYWqqJlb97CKzbXIb6IN5Rtv6vjVOEAHV3HpoeJ9Pt6DBnsXCCKfcQGcm8aeJK0tlagG8EYOtYNLzb5sb6wuBkwIdc03libVa
    ZrbJBqKH2jGimoE5Xn67V4tmYG3VUIgy04UUpGh8RBw12MqPHLSzJ8KPE5jVxgBlBgEP0BVhtVy6pu0n9CUGUwsfg_dRGAIIDvDV6sa5SzlmxKGa
    a9P_uGf6myt",
  
```

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

POSTGet Token

GETDisplay all questions

POSTCreate Question

+...No Environment

Create QuestionExamples (0)

POSThttp://localhost:8000/api/questions

SendSave

ParamsAuthorizationHeaders (2)BodyPre-request ScriptTestsSettingsCookiesCodeComments (0)

▼ Headers (2)

KEY	VALUE	DESCRIPTION
<input checked="" type="checkbox"/> Accept	application/json	
<input checked="" type="checkbox"/> Authorization	Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsImp0aSI6ImU2OWRhNjllOTdlNWYyZmMlNmEzZGI2NjBkOTE2NjVjY2QyNjBINTFMjQzMmNiZWDE2JRmM2lyMjI3Mzl1ZmZIM2ZYzkcWJlYmYyYzc1YWUyIn0.eyJhdWQiOiIyIiwianRpIjoiaWZITDY2ZGE2OUU2NTU1ZGM2YTNYjY2MGQ5MTY	Description
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

Create Question

Examples (0)

POST

http://localhost:8000/api/questions

Send

Save

Params

Authorization

Headers (10)

Body

Pre-request Script

Tests

Settings

Cookies

Code

Comments (0)

none

form-data

x-www-form-urlencoded

raw

binary

GraphQL BETA

JSON

Beautify

1 {

2 "title": "this is my other question",

3 "body": "Hi there, this is my anohter 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 a REST client interface with a top bar containing tabs for 'POST Get Token', 'GET Display all questions', and 'POST Create Question'. The 'POST Create Question' tab is active. Below the tabs, the request details are shown: Method 'POST', URL 'http://localhost:8000/api/questions', and a 'Send' button. The request body is a JSON object: 

```
{  "title": "this is my other question",  "body": "Hi there, this is my anohter question"}
```

. Below the request, the response is displayed. The status is '200 OK', time is '132ms', and size is '758 B'. The response body is a JSON object: 

```
{  "message": "Your question has been submitted",  "question": {    "id": 10,    "title": "this is my other question",    "slug": "this-is-my-other-question",    "votes_count": null,    "answers_count": null,    "views": null,    "status": "unanswered",    "excerpt": "Hi there, this is my anohter question",    "created_date": "1 second ago",    "user": {      "id": 1,      "url": "#",      "name": "Paige McCullough",      "avatar": "https://www.gravatar.com/avatar/cc770b522cebaca4210aa50ce3c3b902?s=32"    }  }}
```

## 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 leaned 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
```