# CREATE QUESTION ENDPOINT

In this lesson we are going to see how to crate 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::resrouce` does, but it exclude the `edit` and `create` actions.

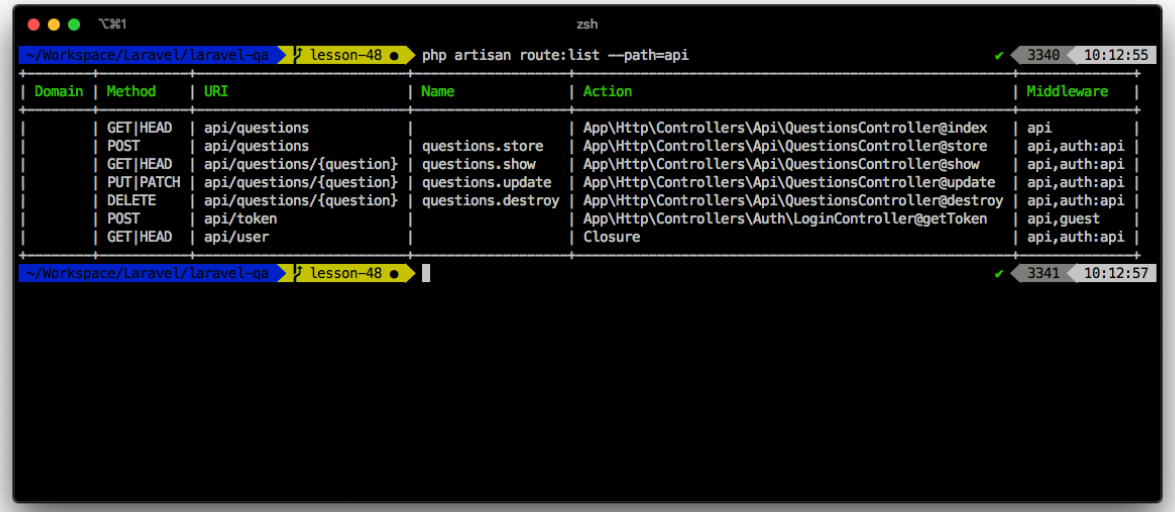
In the first argument let's specify `/question`, and in the second argument it will refer to `Api\QuestionsController`.

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.




### 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.

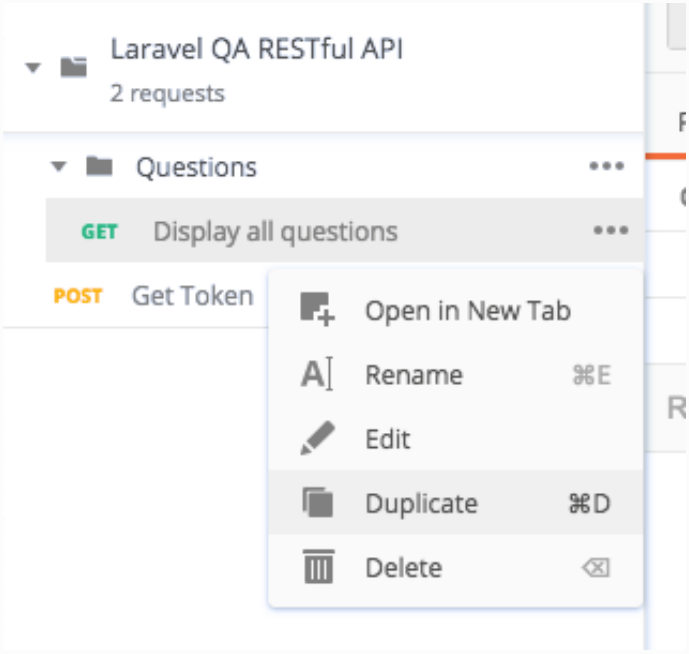
```
use App\Http\Requests\AskQuestionRequest;
```

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

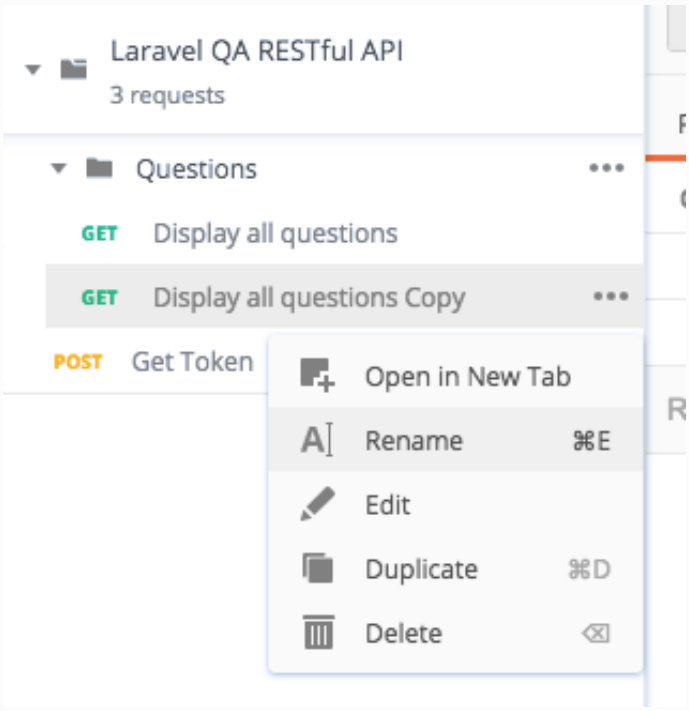
# TEST OUR API ENDPOINT IN POSTMAN

**1. Define a new request**

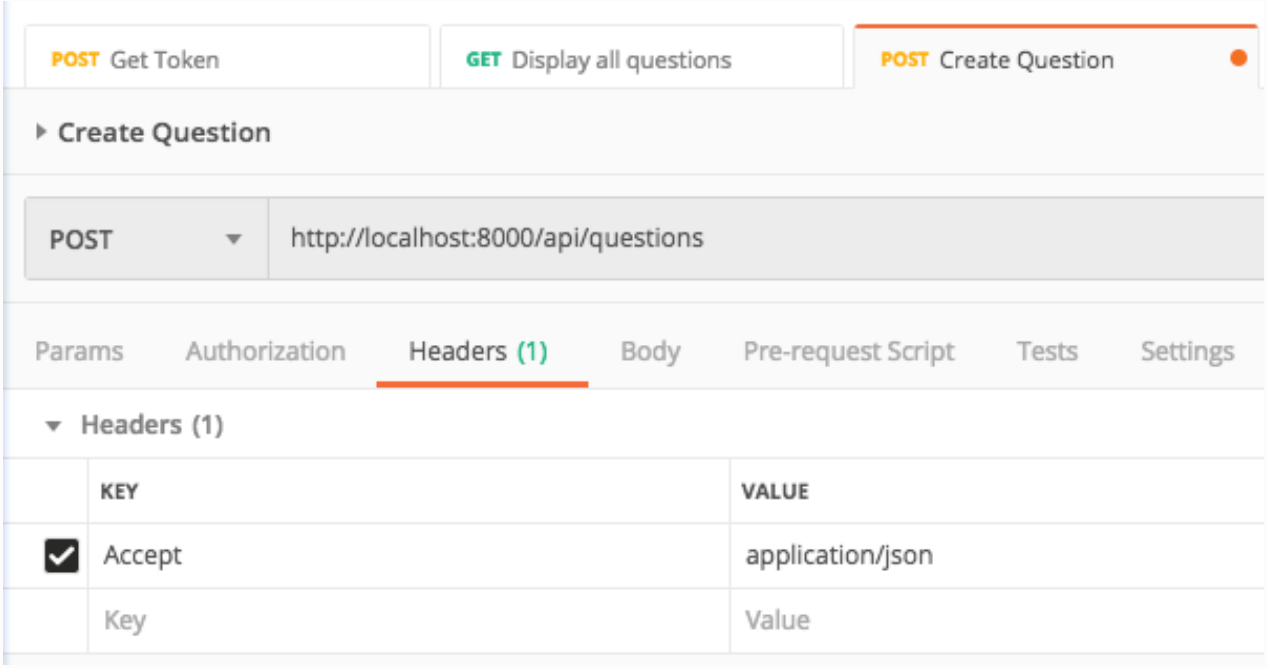
In Postman let's duplicate our `Display all questions` test request.




We can then rename it as `Create Question`.

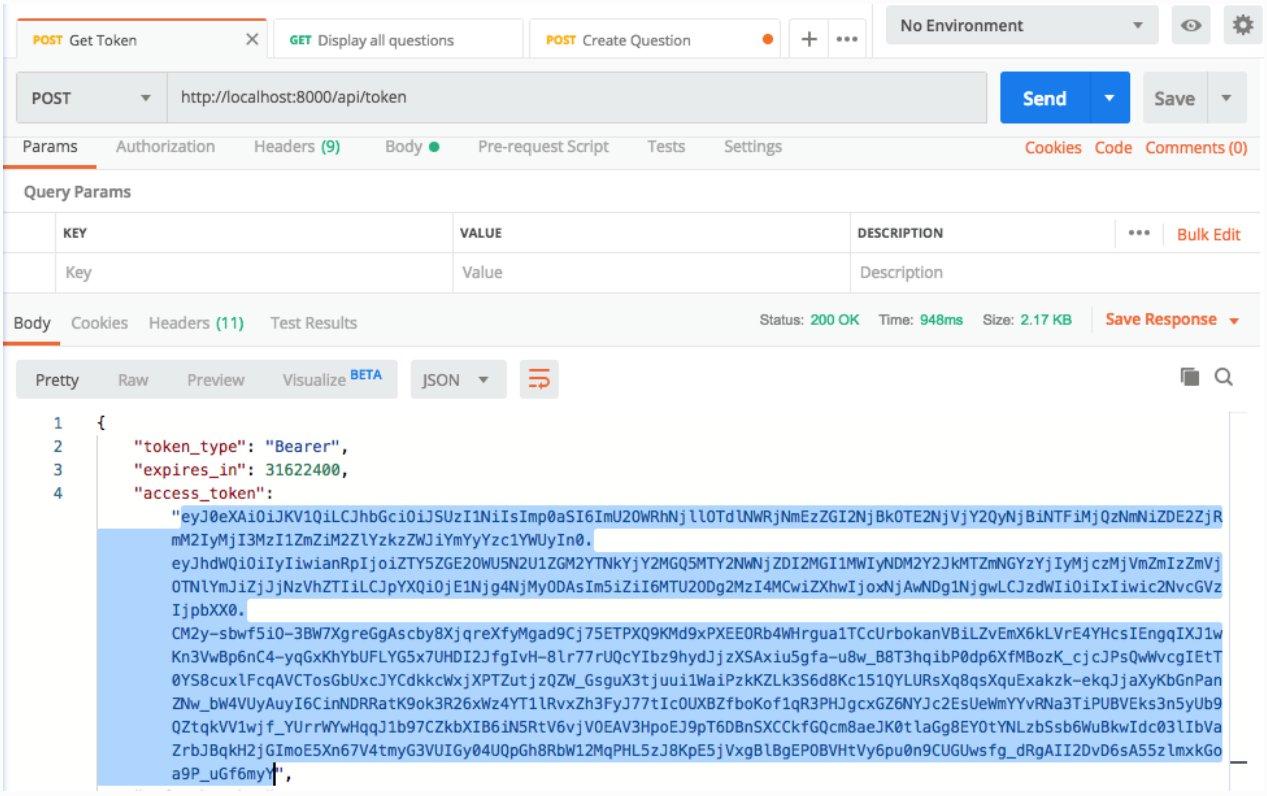



In the right side we can change the HTTP verb to be `POST`, the request url is going to be the same, so we don't need to make any change on it. Let's go to *Headers* section and specify some headers. Firstly, let's add `Accept` in the *Key* column while `Application/json` in the *value*.

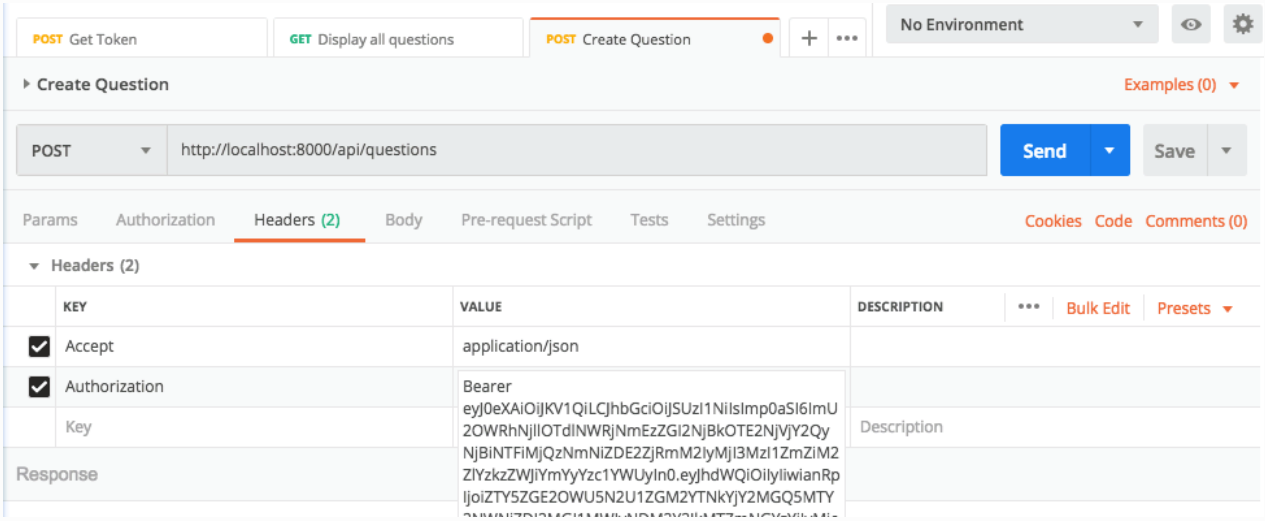



Second, we let's specify the `Authorization` in the *key* column because remember we have put our route in `auth:api` middleware.

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.




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.

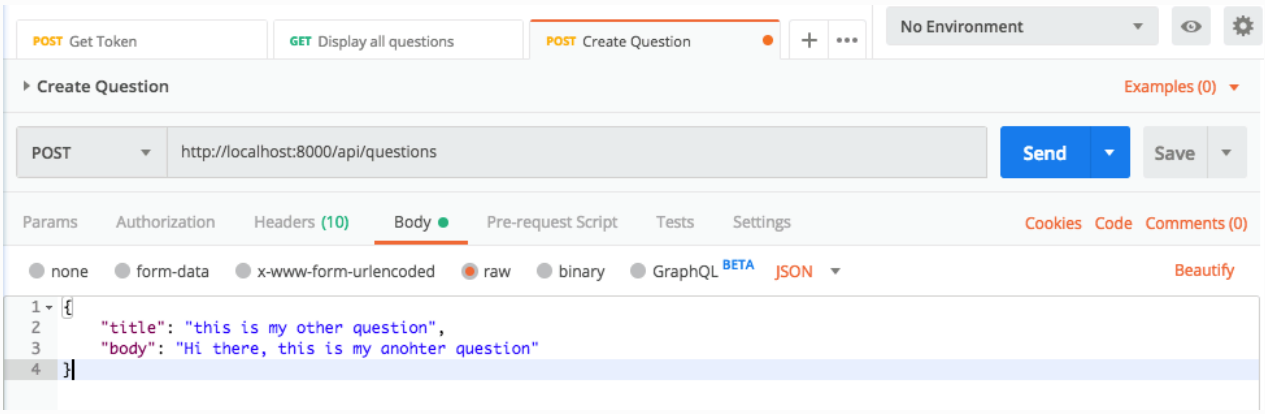



### 2. Test the Create Question request

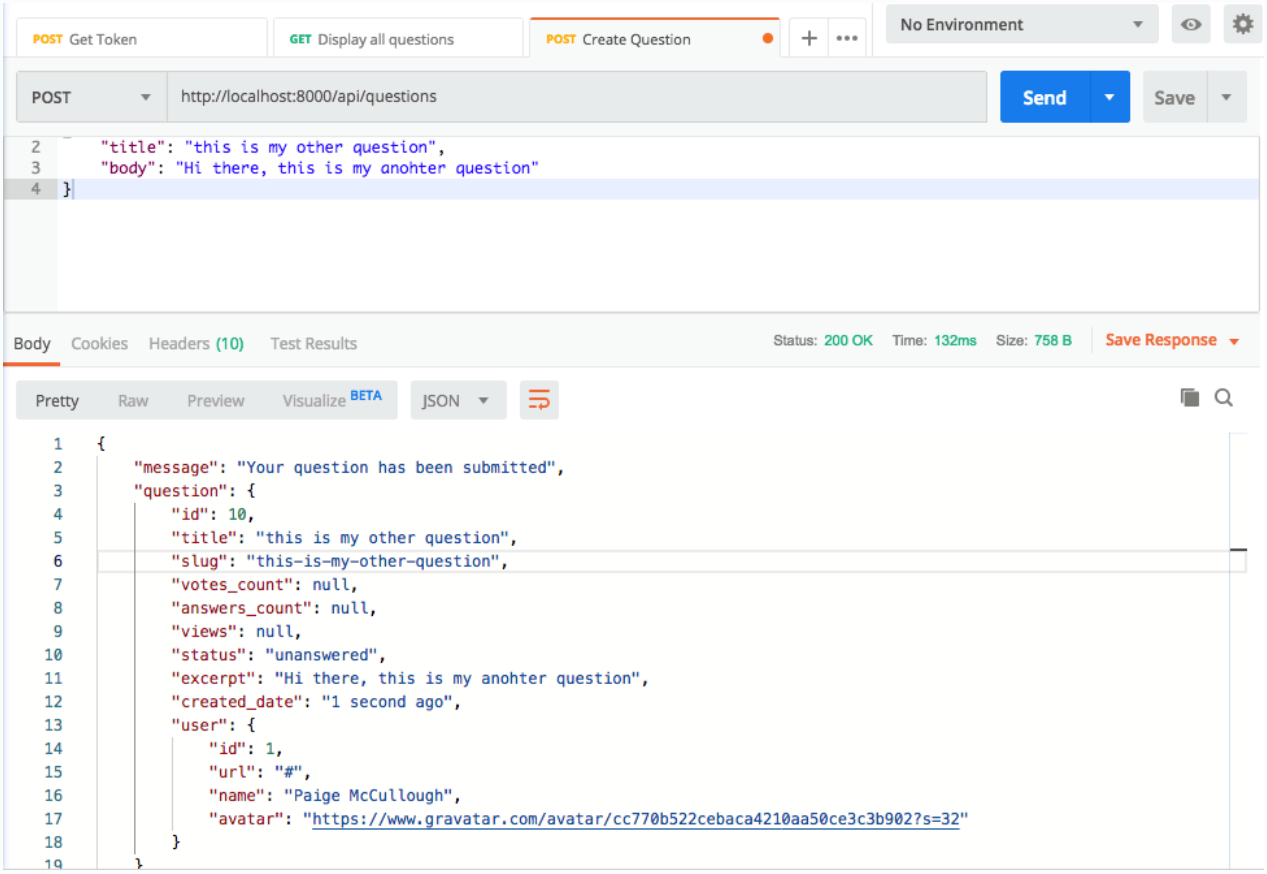
Now if we hit the `Send` button, we'll get validation errors which is a good sign.

```
{
    "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.




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




# 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
```