CREATING UPDATE & DELETE QUESTION ENDPOINTS

In this lesson we're going to create endpoints for update & delete question. Since we've defined our api resource route In the last lesson so we no longer need to touch the routes api for now. All we have to do is working on the controller and then perform some tests in postman.

Alright, before we move on let's go ahead and open up our terminal then create a new branch:

```
git checkout -b lesson-49
```

CREATE SHOW QUESTION ENDPOINT

Let's start off by modify the `show` method in our `Api/QuestionsController.php`. Basically this method as the name implies is to show a resource.

Now when working on api we no longer need the `create` or `edit` method to show the form. That's why when editing a resource we're going to use this `show` method and get the resource in json data. We can then use that response to populate our form.

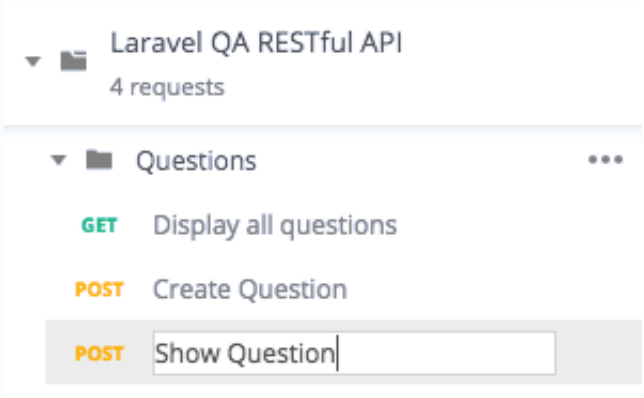
So in this method let's return `response()->json()`. Then put an array inside it. And then inside that array let's specify the response data contains `title`, `body` and `body_html`.

```
// Api/QuestionsController.php
public function show(Question $question)
{
    return response()->json([
        'title'     => $question->title,
        'body'      => $question->body,
        'body_html' => $question->body_html
    ]);
}
```

Let's save and give it a test in postman.

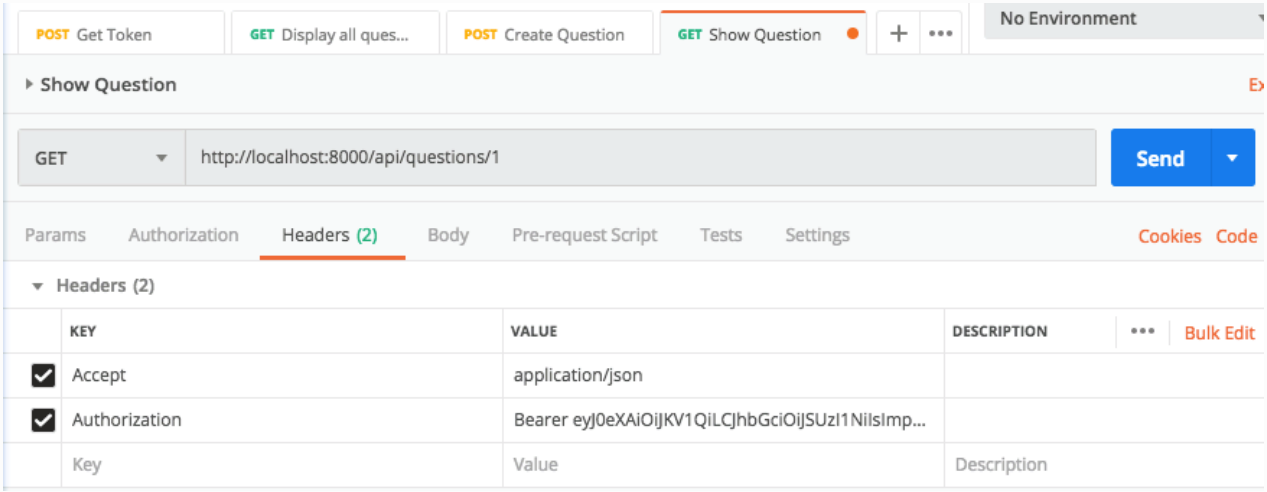
TEST THE SHOW ENDPOINT

Head over to postman. Then duplicate the `Create question` request. And then rename it as `Show Question`.

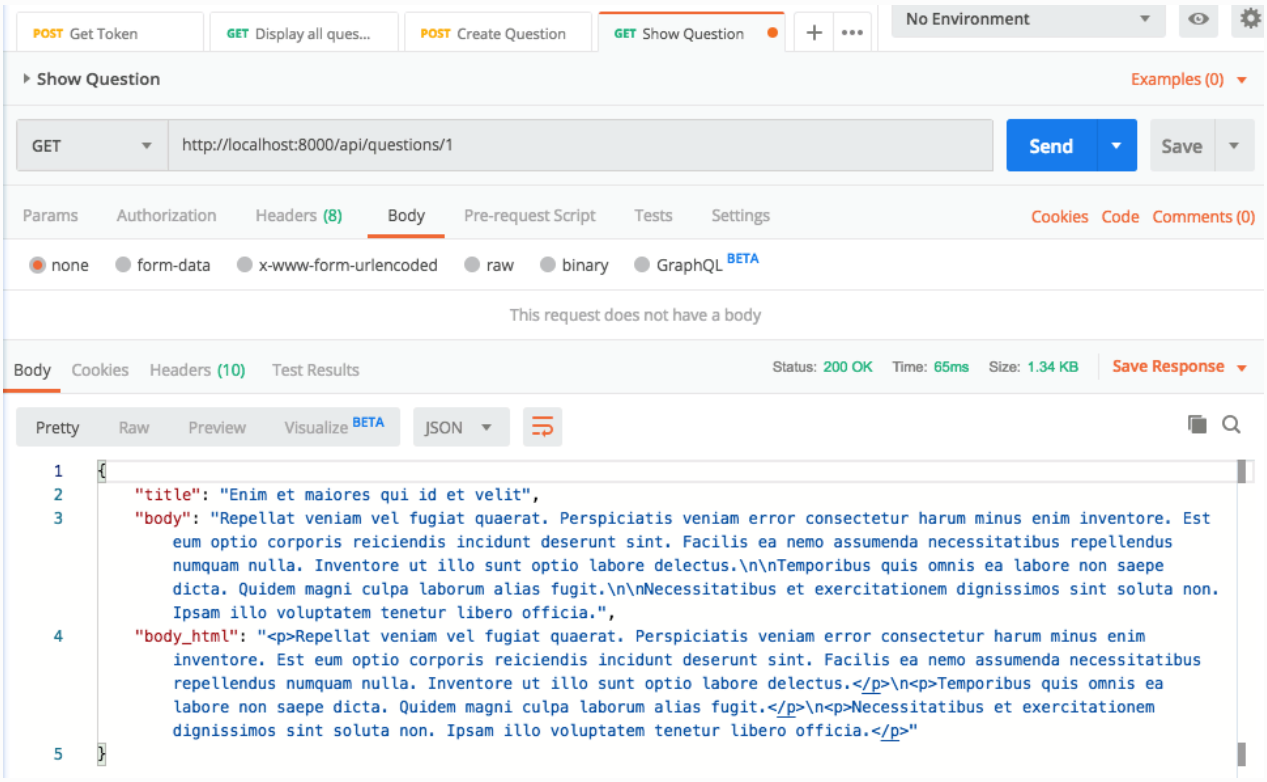


Let's change the http verb to `GET`. The url is gonna be `http://localhost:8000/api/questions/1`. Make sure you have question with id `1`. Or you can change it to any existing number that you have in your `questions` table.

Go to the **headers** section and make sure you have `Accept` and `Authorization` setup. Go to **body** section and choose *none* option.



Now if we hit the **Send** button we'll have a response back.



Hit **Save** button to save the request changes.

CREATE UPDATE & DELETE QUESTION ENDPOINTS

1. The `update` method

We can grab the logics for `update` and `destroy` method from our old `QuestionsController.php`. Let's reopen that file. Then copy the `update` method.

Back to `Api/QuestionsController.php` then replace the `update` method boilerplate. And then remove the if statement as well as redirection code. So here our `update` method look like:

```
// Api/QuestionsController.php
public function update(AskQuestionRequest $request, Question $question)
{
    $this->authorize("update", $question);

    $question->update($request->only('title', 'body'));

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

2. The `delete` method

Let's go back to our old `QuestionsController.php` and copy the `destroy` method. Back to our `Api/QuestionsController.php`. Then inside that file let's replace the `destroy` method boilerplate.

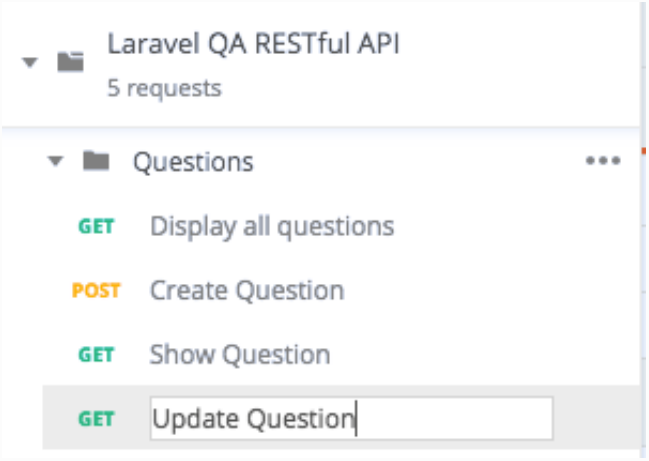
Don't forget to get rid of the if statement as well as redirection code. And here is the `destroy` method look like:

```
// Api/QuestionsController.php
public function destroy(Question $question)
{
    $this->authorize("delete", $question);

    $question->delete();

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

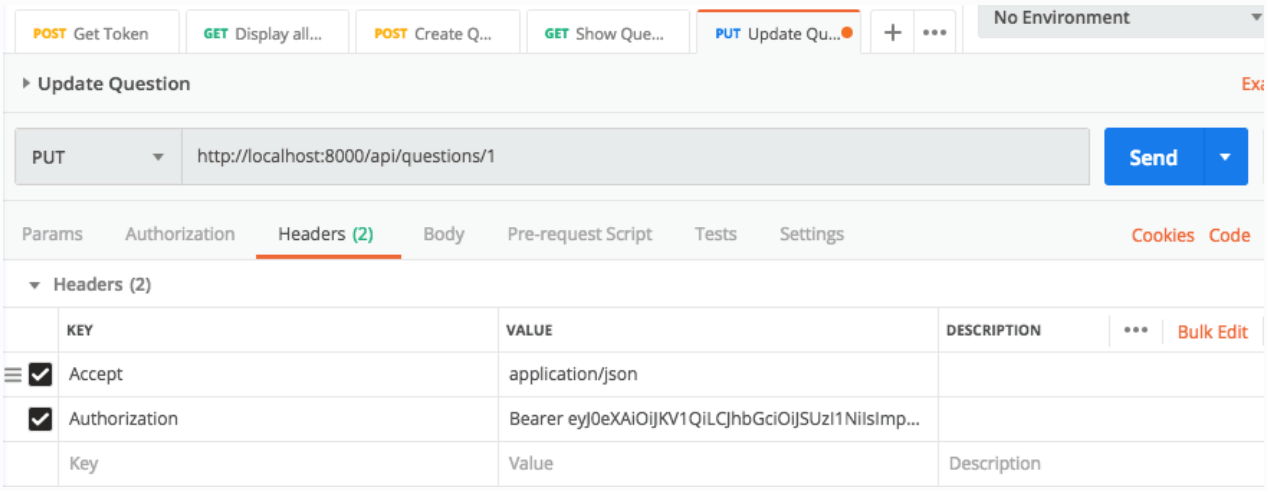
Save all changes. And now let's test them in postman.

TEST THE UPDATE QUESTION ENDPOINT

Let's switch back to Postman. Then duplicate the `Show question` test request. And then rename it to `Update Question`.



Let's change the HTTP method to `PUT`. The url is going be the same. But you can change the id to any existing number if you want. Make sure you have `Accept` and `Authorize` being set in the **Headers** section. And for now we can leave the request body empty.



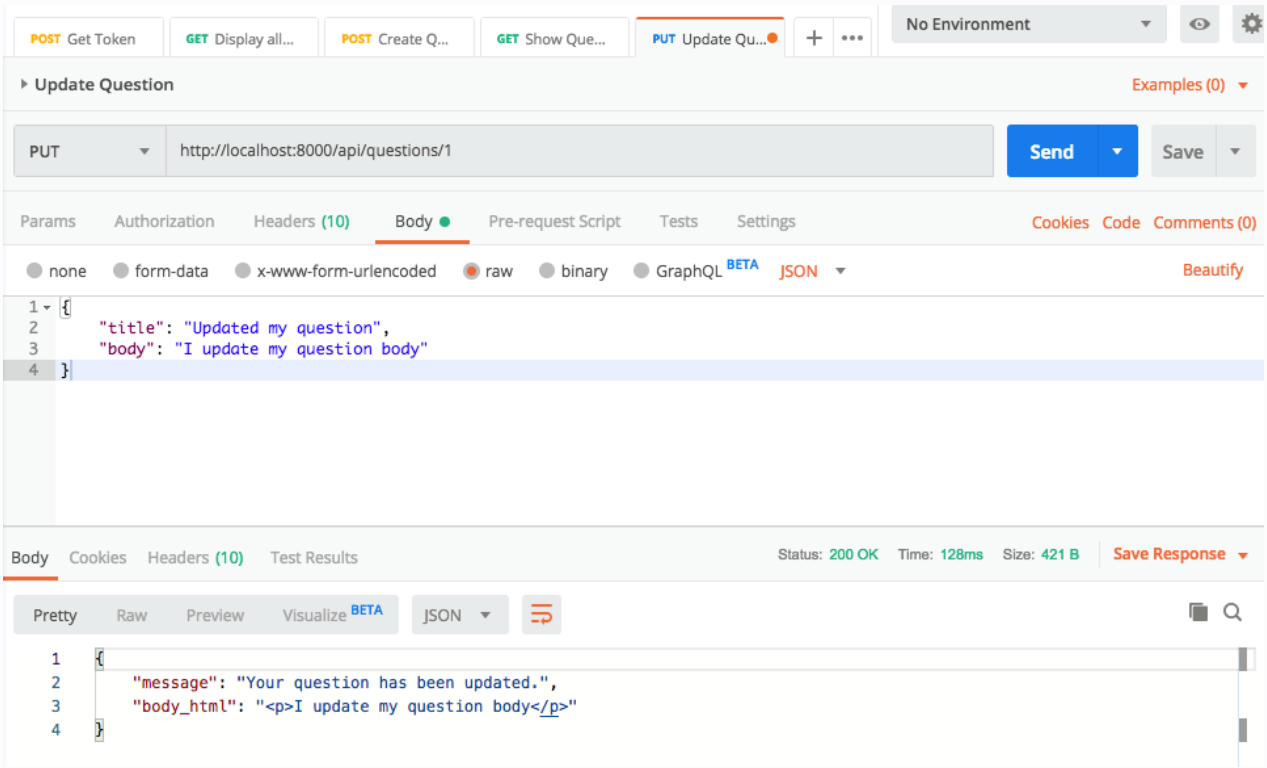
Now if we hit the **Send** button sure enough we got validation error message.

```
{
    "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. Select **raw** option and select the **JSON** type. And then specify the request body like so:

```
{
  "title": "Updated my question",
  "body": "I update my question body"
}
```

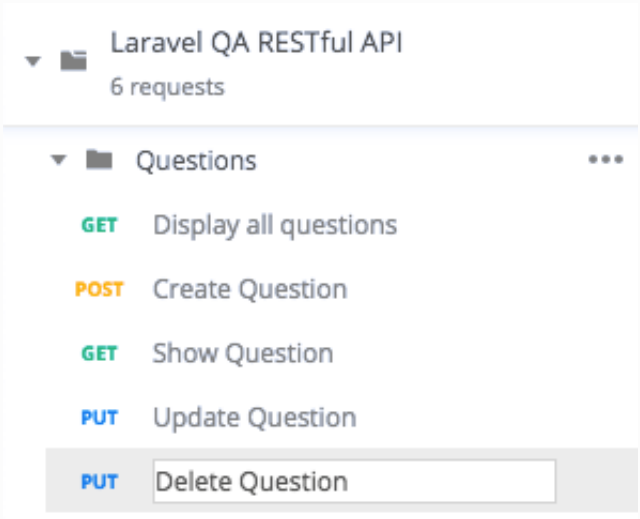
If we hit the **Send** button again. Now we get back the expected response.



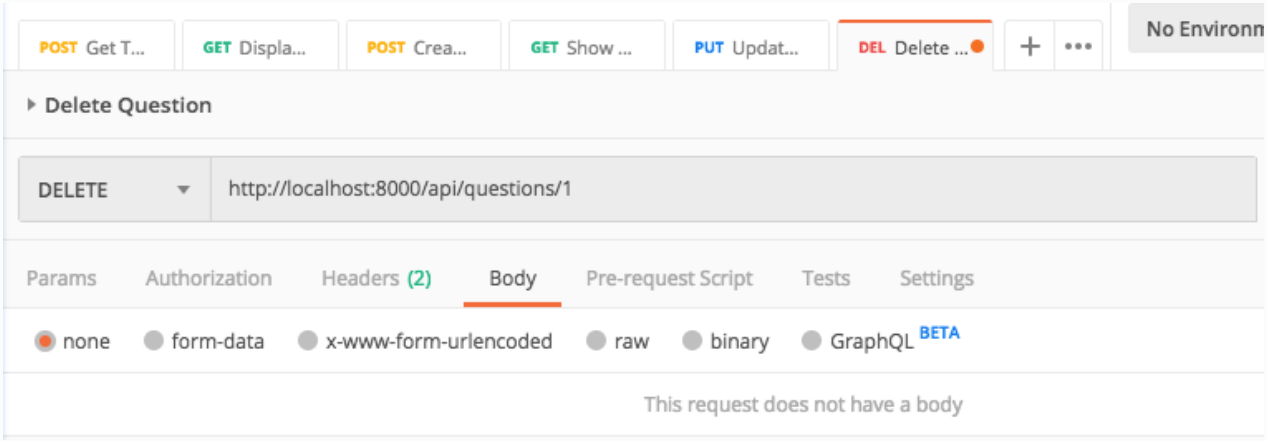
Alright, now let's hit the **Save** button to save the change in our Update Question test request. And now let's test the destory question endpoint.

TEST THE DESTROY QUESTION ENDPOINT

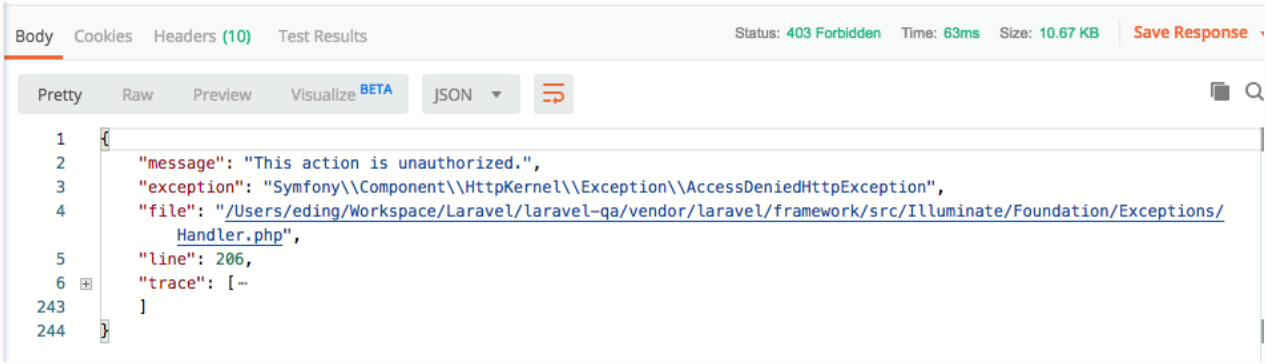
Let's duplicate the `Update question` request. Then rename it to `Delete Qeustion`.



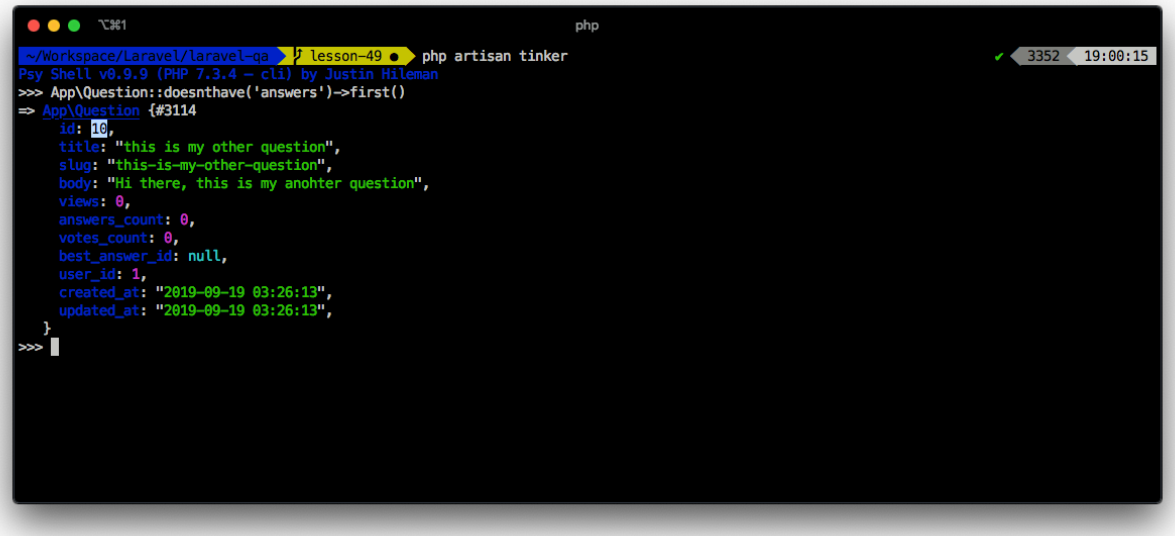
Change the HTTP method to `DELETE`. Keep the `Accept` and `Authorization` in the **Headers** section and remove the request body in the **Body** section by choosing the **none** option.



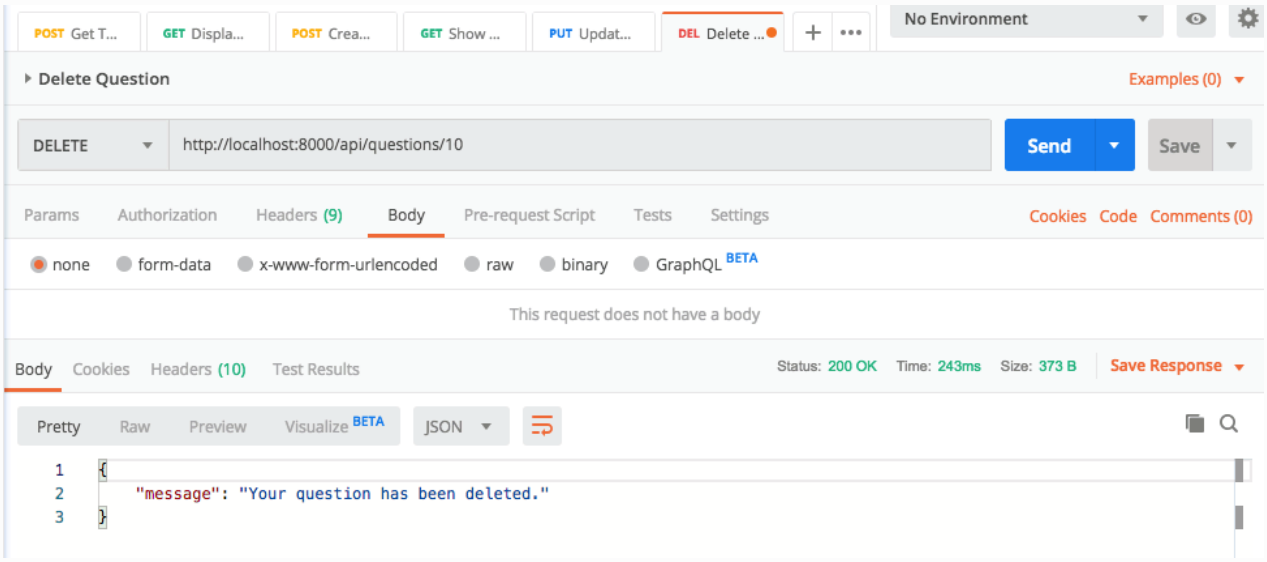
Now if we hit the **Send** button. Probably you will get this such response error:



That error occurred because you tried to remove a question and that question has one or more related answers. So to fix that you can simply go to **tinker** then find the question which does not any related answer. In my case I have question with id `10`.



Now if change the id in the url and hit the **Send** button again. Here I'll get the expected response.



SUMMARY

In this lesson, we looked at how we can create api endpoints for updating and deleting existing question. We've learned how to properly test our api endpoints in postman. And we've also briefly seen how to find record which does not have any relationship record using eloquent `doesnthave()` method.

Let's commit all changes into our git repo. When you ready let's move on to the next lesson.

```
git add .
git commit -m "Create update & delete question endpoints"
git push origin lesson-49
```