REST Controllers, Routing, JSON Responses

This document defines a complete walkthrough of creating a REST API for our Blog application, from setting up the serializer bundle, creating our own API bundle, ending up with fully functioning REST API.

# Create the Project

Open terminal and go to the directory of your project

In terminal, type following command: **symfony new BlogRestApi**

|  |
| --- |
| symfony new --full BlogRestApi |

# Install serialization bundle

For the purpose of this exercise we need functionality which serialize/deserialize our Doctrine entities to JSON and vice versa – JMSSerializerBundle

Open terminal / console and go to the directory of your project. If you still don’t have **Composer** you can download it from here: http://getcomposer.org

In terminal / cmd type following command: **composer require jms/serializer-bundle**

|  |
| --- |
| composer require jms/serializer-bundle |

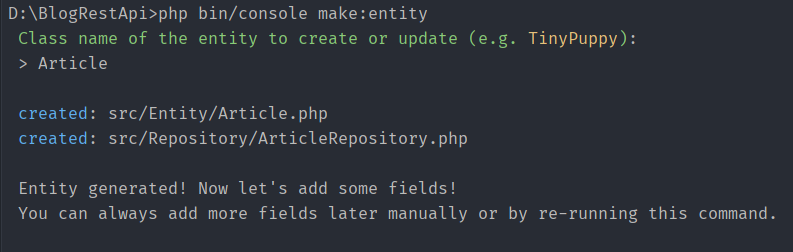
The command will download the bundle and include it automatically to the autoloader. You don’t need to include anything anywhere.

# Create the Article Entity

Open Terminal or Command Prompt (CMD) in the blog project root folder. Let’s model our articles. That means that we are going to create the defining properties of an article. To do that, we need to generate a [Doctrine Entity](http://docs.doctrine-project.org/en/latest/reference/working-with-objects.html). Our entity will describe what are we going to store in our database. The following command will **start entity generator wizard**:

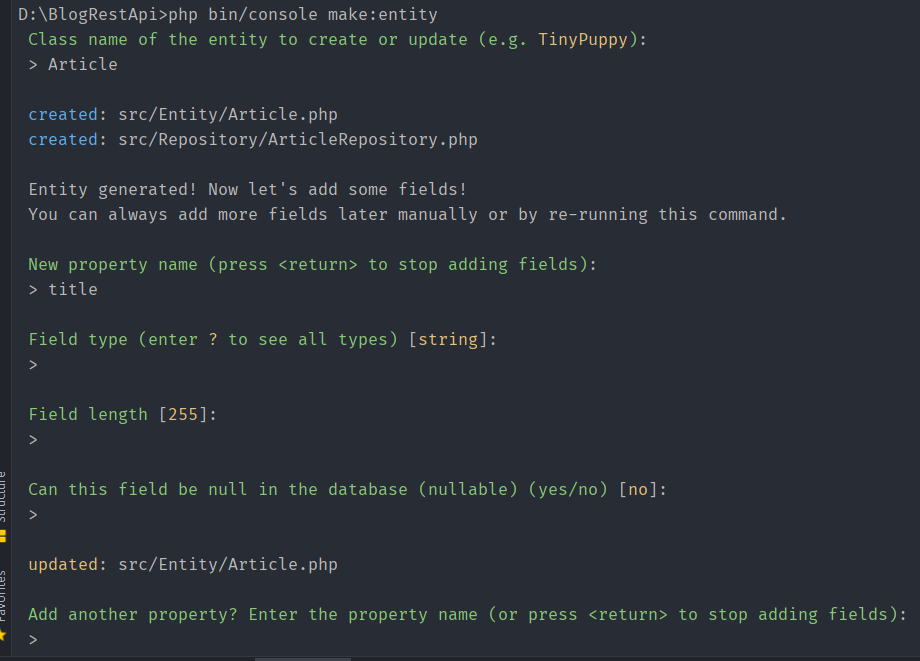
|  |
| --- |
| php bin/console make:entity |

You should see this result:

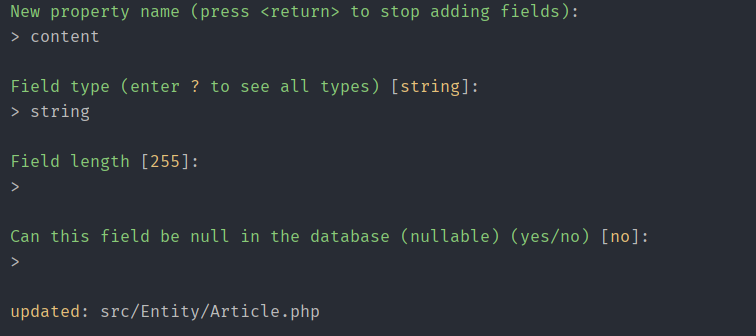


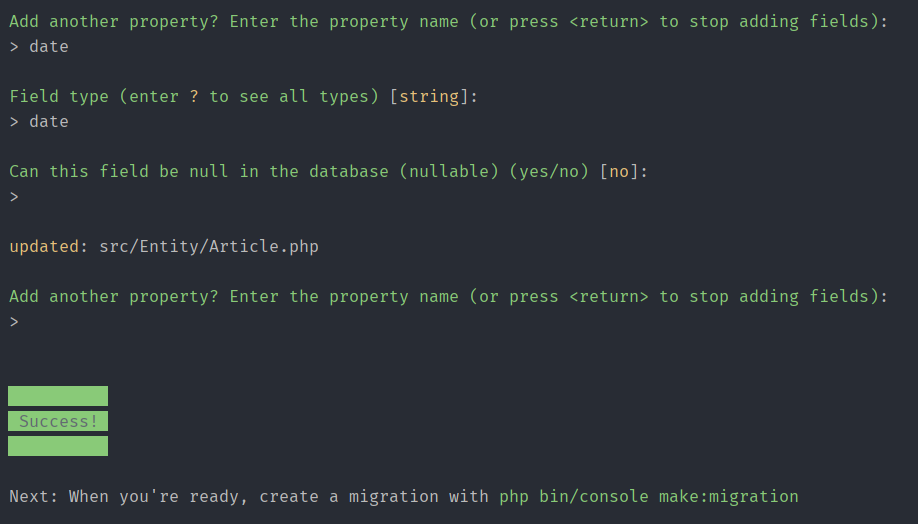
Our first field will be the “Title” of our article. Just write “**title**” and press ‘**Enter**’.

Press ‘**Enter**’. You should see “**Field length [255]**”. Press ‘**Enter**’ again. You will be asked if you want to make the field **nullable**. Press ‘**Enter**’. Finally, you will be asked to make your field **unique**. Just press ‘**Enter**’ one more time



Similar to this, we should create 2 more fields for the “**content**” and “**date**”. Here is how we create them:





# Update Database

Change database connection in .env file

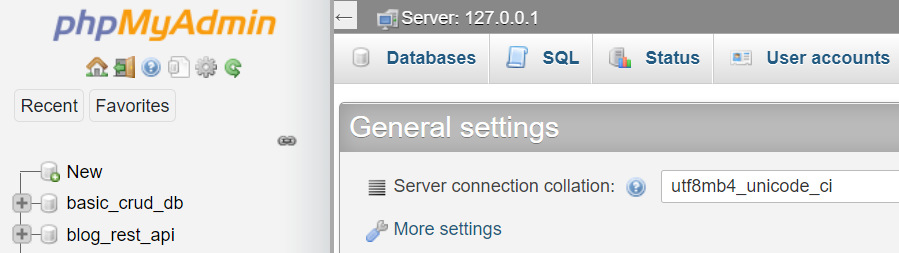
Specific: admin, password and database name

|  |
| --- |
| …  DATABASE\_URL=mysql://root:@127.0.0.1:3306/blog\_rest\_api?serverVersion=mariadb-10.4.11 |

Create database

|  |
| --- |
| php bin/console doctrine:database:create |

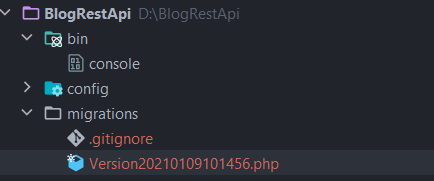
Open phpMyAdmin, you can see the database “blog\_rest\_api” is created.



To perform migration, type below command:

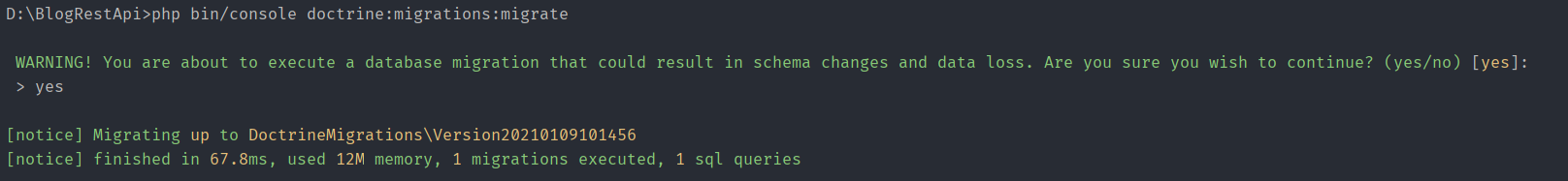
|  |
| --- |
| php bin/console make:migration |

You can see the migration file is created in folder **migrations**

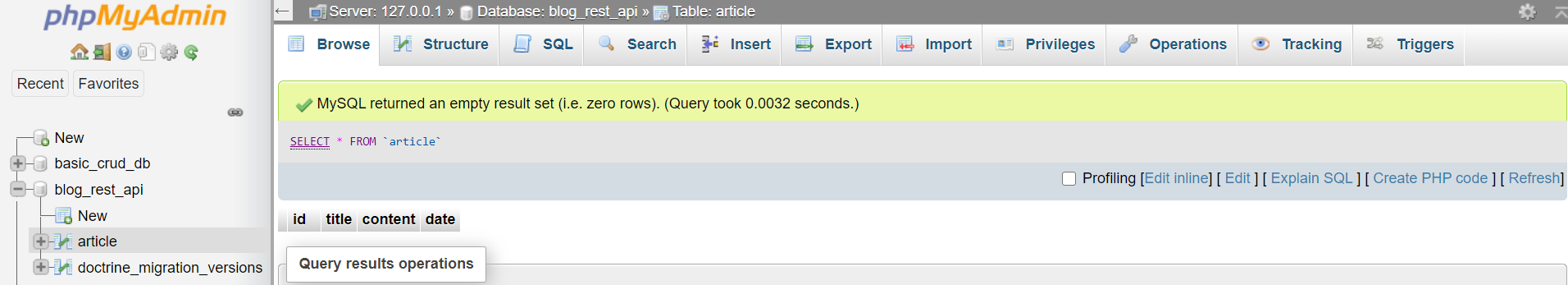


To run that migration, execute command below:

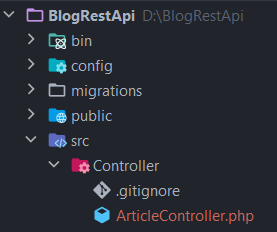
|  |
| --- |
| php bin/console doctrine:migrations:migrate |



Now, you should see the table “article” is created in the database “blog\_rest\_api”



# Create ArticleController

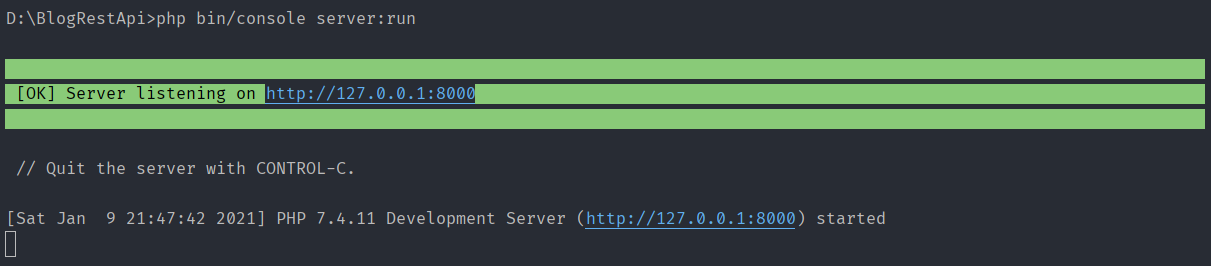


Create articlesAction() - get list of articles

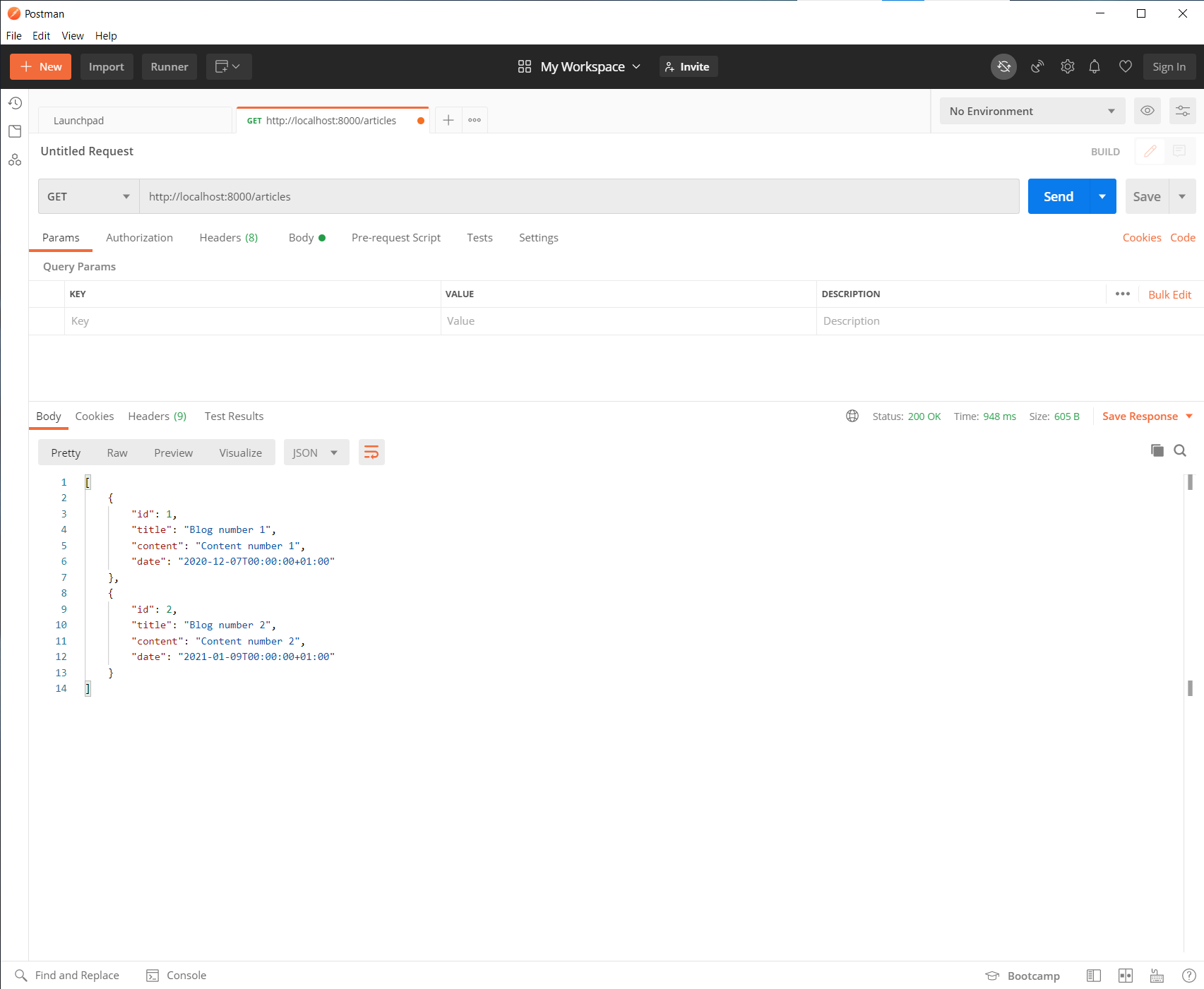
class ArticleController extends AbstractController  
{  
 // Config serializer  
 private $serializer;  
  
 public function \_\_construct(SerializerInterface *$serializer*)  
 {  
 $this->serializer = *$serializer*;  
 }  
  
 /\*\*  
 \* @Route("/articles", methods={"GET"}, name="rest\_api\_articles")  
 \*/  
 public function articlesAction()  
 {  
 // Get all articles in Database  
 $articles = $this->getDoctrine()->getRepository(Article::class)->findAll();  
  
  
 // Convert object articles to JSON  
 $json = $this->serializer->serialize($articles, 'json');  
  
 return new Response($json,  
 Response::HTTP\_OK,  
 array('content-type' => 'application/json')  
 );  
 }

To test the first action, type the following command to run the local server:

|  |
| --- |
| php bin/console server:run |



Check if everything works correctly.



Create new **articleAction() - get single article**

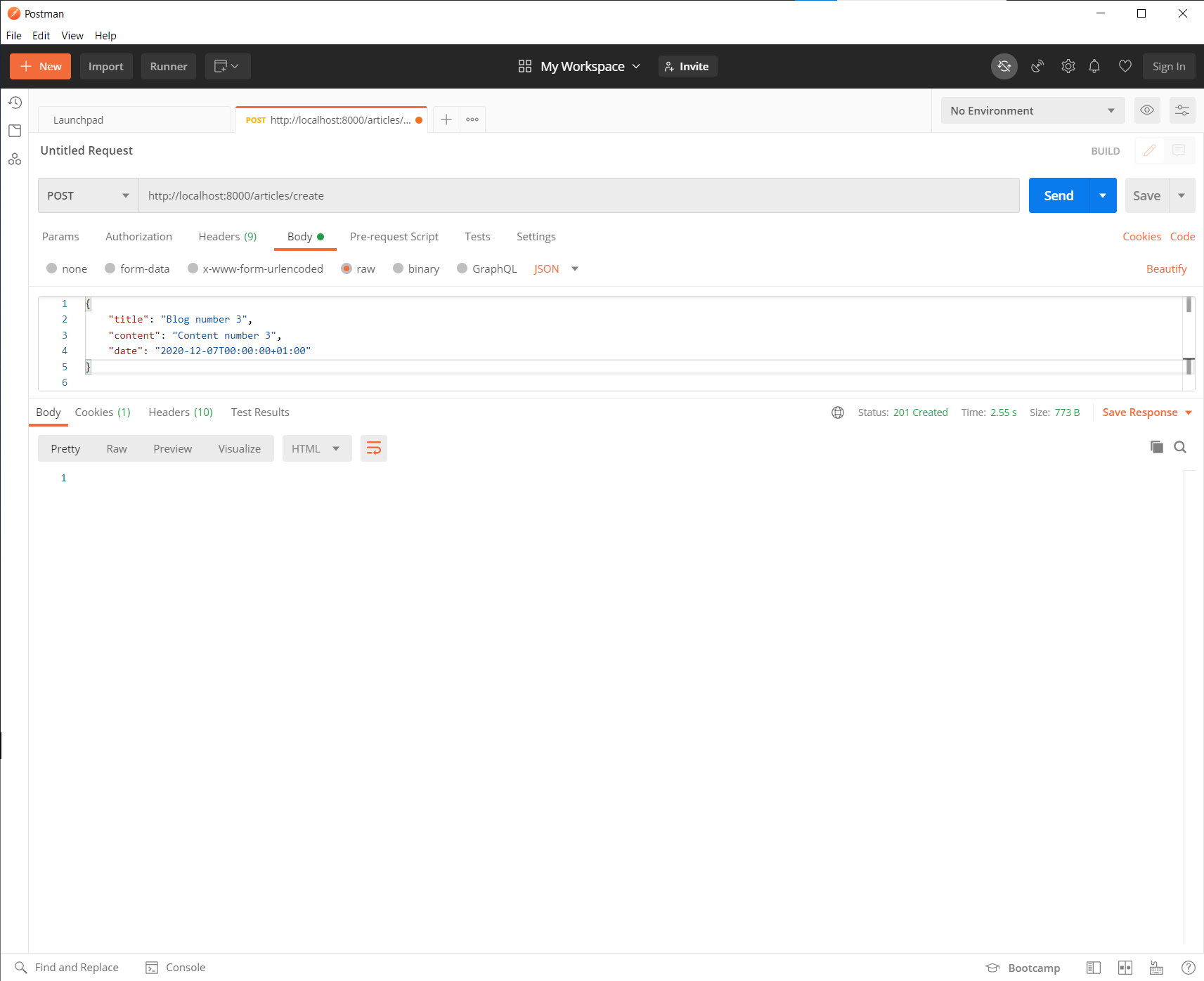
/\*\*  
 \* @Route("/articles/{id}", methods={"GET"}, name="rest\_api\_article")  
 \*/  
public function articleAction(*$id*)  
{  
 // Get a single article from Database  
 $article = $this->getDoctrine()->getRepository(Article::class)->find(*$id*);  
  
 // Article not found  
 if ($article == null) {  
 return new Response(json\_encode(array('error' => 'article not found')),  
 Response::HTTP\_NOT\_FOUND,  
 array('content-type' => 'application/json')  
 );  
 }  
   
 // Article found  
 $json = $this->serializer->serialize($article, 'json');  
 return new Response(  
 $json,  
 Response::HTTP\_OK,  
 array('content-type' => 'application/json')  
 );  
}

The code is pretty much the same as previous one with a little difference.  
If requested article is not found **Response** object is returned containing error message encoded in **JSON**The **Response code** for not found resources is **404** or in Symfony **Response::HTTP\_NOT\_FOUND**For successful requests response code should be **200 OK .**

In Symfony - Resonse::HTTP\_OK  
Create new action: createAction() - responsible for creating new articles

/\*\*  
 \* @Route("/articles/create", methods={"POST"}, name="rest\_api\_article\_create")  
 \*/  
public function createAction(Request *$request*)  
{  
 try {  
 // Get data in Request from client  
 $article = new Article();  
 $data = json\_decode(*$request*->getContent(), true);  
 $article->setTitle($data['title']);  
 $article->setContent($data['content']);  
 $article->setDate(\DateTime::**createFromFormat**('Y-m-d', $data['date']));  
  
 // Try to insert new article to Database  
 $em = $this->getDoctrine()->getManager();  
 $em->persist($article);  
 $em->flush();  
  
 // Return OK if created successfully  
 return new Response(null, Response::HTTP\_CREATED);  
 } catch (\Exception $e) {  
 // Return BAD\_REQUEST if created unsuccessfully  
 return new Response(null, Response::HTTP\_BAD\_REQUEST);  
 }  
}

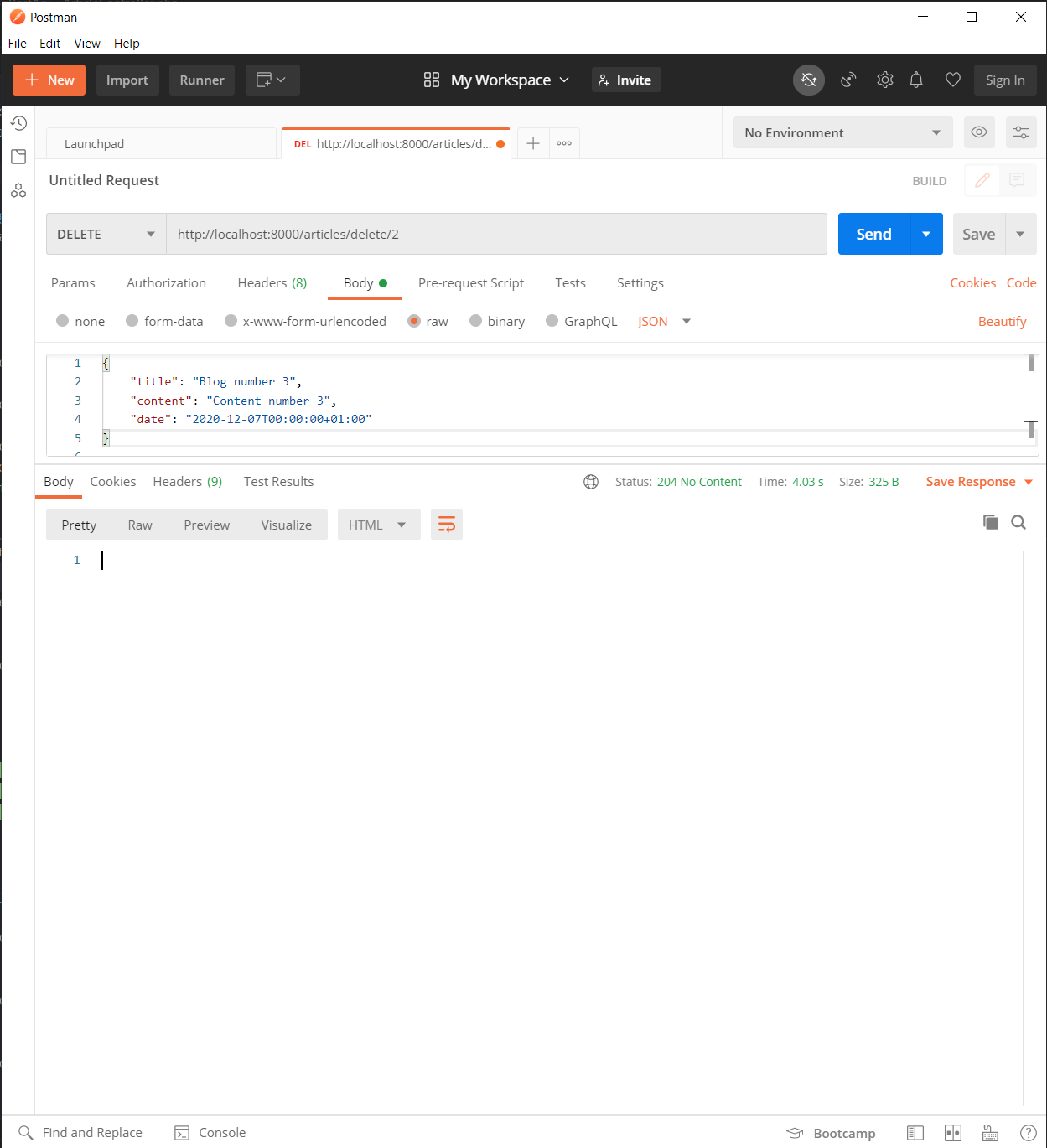
Test to see if it works correctly



Create new **deleteAction() - delete resource**

/\*\*  
 \* @Route("/articles/delete/{id}", methods={"DELETE"}, name="rest\_api\_delete")  
 \*/  
public function deleteAction(*$id*)  
{  
 try {  
 // Get id from Request and try to delete  
 $article = $this->getDoctrine()->getRepository(Article::class)->find(*$id*);  
 if ($article == null) {  
 $statusCode = Response::HTTP\_NOT\_FOUND;  
 } else {  
 $em = $this->getDoctrine()->getManager();  
 $em->remove($article);  
 $em->flush();  
  
 $statusCode = Response::HTTP\_NO\_CONTENT;  
 }  
  
 return new Response(null, $statusCode);  
 } catch (\Exception $e) {  
 // Return BAD\_REQUEST if something went wrong  
 return new Response(null, Response::HTTP\_BAD\_REQUEST);  
 }  
}

Test to see if “delete” action works correctly



Create new action: **editAction() - edit article**

/\*\*  
 \* @Route("/articles/edit/{id}", methods={"PUT"}, name="rest\_api\_edit")  
 \*/  
public function editAction(Request *$request*, *$id*)  
{  
 // Find if article with id exsited  
 $article = $this->getDoctrine()->getRepository(Article::class)->find(*$id*);  
 if ($article == null) {  
 $statusCode = Response::HTTP\_NOT\_FOUND;  
 } else {  
 $data = json\_decode(*$request*->getContent(), true);  
 $article->setTitle($data['title']);  
 $article->setContent($data['content']);  
 $article->setDate(\DateTime::**createFromFormat**('Y-m-d', $data['date']));  
  
 $em = $this->getDoctrine()->getManager();  
 $em->persist($article);  
 $em->flush();  
  
 $statusCode = Response::HTTP\_NO\_CONTENT;  
 }  
 return new Response(null, $statusCode);  
}