Symfony REST API (without FosRestBundle)

# 1- What is Rest?

Rest as Representational State transfer is an architectural style for developing web services. It is something that can not be ignored because there is a big need for creating Restful applications in today’s ecosystem. This might be due to the continued climb of JavaScript and these frameworks.

Rest API should use HTTP protocol. This mean that when the client makes any request to this web service, it can specify any of the normal HTTP verbs of GET, POST, DELETE and PUT. Below is what would happen If the respective verbs were sent by the client.

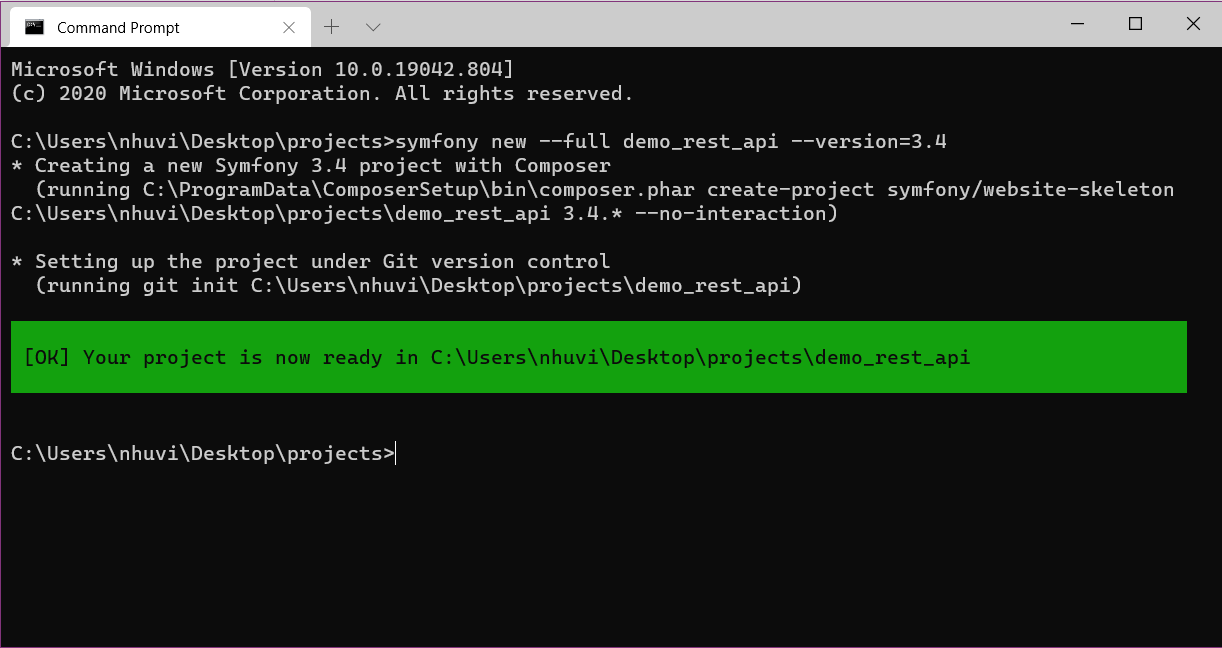
- GET: This would be used to get the list of resources or the details of one.  
- POST: this would be used to create a new resource.  
- PUT: This would be used to update an existing resource.  
- DELETE: This would be used to delete an existing resource.

Rest is stateless and this mean that the server side doesn’t hold any request state. The state should be kept on the client side (example of using JWT for authentication, we are going to secure our RestApi using this). So, when using authentication in our Rest Api, we need to send the authentication header in order to get a correct response in a stateless way.

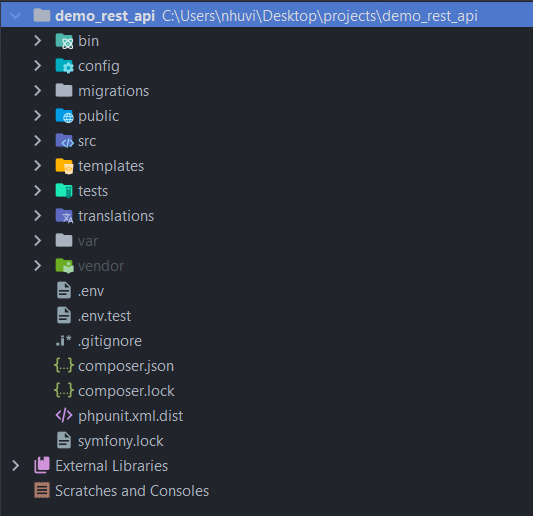
# 2- Create a symfony project:

Firstly, we suppose you have installed php and the composer package manager to create a new symfony project. After that, create a new project using the following command in the terminal:

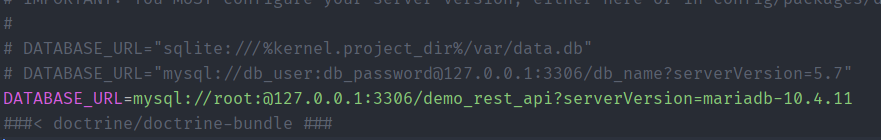
Symfony new –full demo\_rest\_api –version=3.4



Here is what directory structure looks like:



Below is the configuration of database I created which may be set in the .env file.



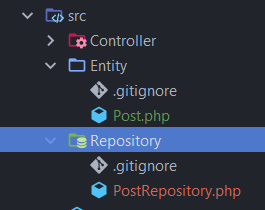
Now, let’s create our first entity. Create a new file called Post.php inside the src/Entity folder.

|  |
| --- |
| <?php  **namespace** App\Entity;  **use** Doctrine\ORM\Mapping **as** ORM;  **use** Symfony\Component\Validator\Constraints **as** **Assert**;  /\*\*  \* @ORM\Entity(repositoryClass=PostRepository::class)  \* @ORM\Table(name="post")  \* @ORM\HasLifecycleCallbacks()  \*/  **class** Post **implements** \JsonSerializable {  /\*\*  \* @ORM\Column(type="integer")  \* @ORM\Id  \* @ORM\GeneratedValue(strategy="AUTO")  \*/  **private** $id;  /\*\*  \* @ORM\Column(type="string", length=100)  \*  \*/  **private** $name;  /\*\*  \* @ORM\Column(type="text")  \*/  **private** $description;  /\*\*  \* @ORM\Column(type="datetime")  \*/  **private** $create\_date;  /\*\*  \* @return mixed  \*/  **public** **function** getId()  {  **return** $this->id;  }  /\*\*  \* @param mixed $id  \*/  **public** **function** setId($id)  {  $this->id = $id;  }  /\*\*  \* @return mixed  \*/  **public** **function** getName()  {  **return** $this->name;  }  /\*\*  \* @param mixed $name  \*/  **public** **function** setName($name)  {  $this->name = $name;  }  /\*\*  \* @return mixed  \*/  **public** **function** getDescription()  {  **return** $this->description;  }  /\*\*  \* @param mixed $description  \*/  **public** **function** setDescription($description)  {  $this->description = $description;  }  /\*\*  \* @return mixed  \*/  **public** **function** getCreateDate(): ?\DateTime  {  **return** $this->create\_date;  }  /\*\*  \* @param \DateTime $create\_date  \* @return Post  \*/  **public** **function** setCreateDate(\DateTime $create\_date): self  {  $this->create\_date = $create\_date;  **return** $this;  }  /\*\*  \* @throws \Exception  \* @ORM\PrePersist()  \*/  **public** **function** beforeSave(){  $this->create\_date = **new** \DateTime('now', **new** \DateTimeZone('Africa/Casablanca'));  }  /\*\*  \* Specify data which should be serialized to JSON  \* @link https://php.net/manual/en/jsonserializable.jsonserialize.php  \* @return mixed data which can be serialized by <b>json\_encode</b>,  \* which is a value of any type other than a resource.  \* @since 5.4.0  \*/  **public** **function** jsonSerialize()  {  **return** [  "name" => $this->getName(),  "description" => $this->getDescription()  ];  }  } |

In order to generate the repository, use the command

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

Then, place the file *PostRepository.php* under the folder *Repository*

**

Run the command php bin/console doctrine:database:create to create the database *demo\_rest\_api*

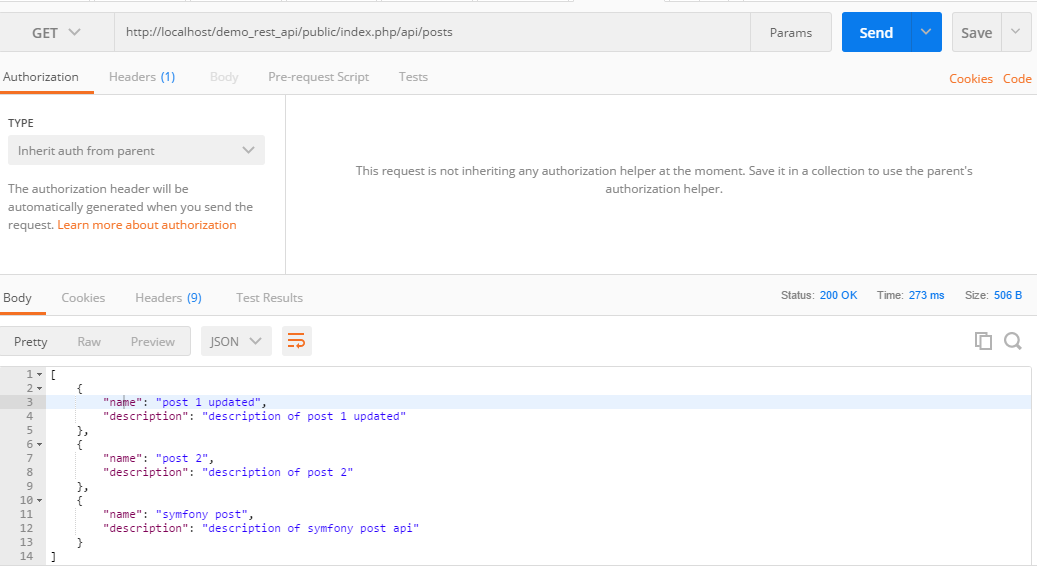
And after run the command: php bin/console doctrine:schema:create to create a database table according to our Post entity.

Now, let’s create a PostController.php where we will add all methods interacting with api. It should be placed inside the folder src/Controller.

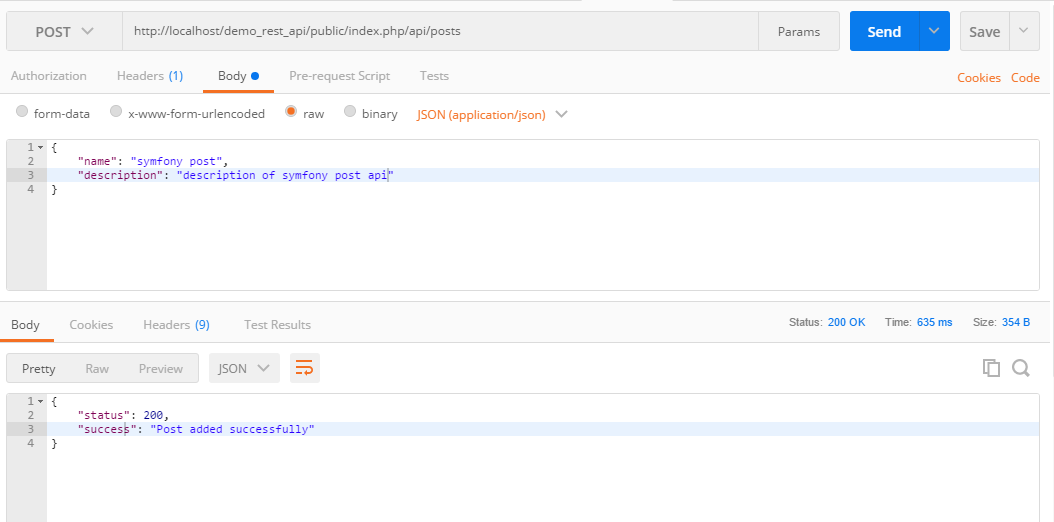
|  |
| --- |
| <?php  **namespace** App\Controller;  **use** App\Entity\Post;  **use** App\Repository\PostRepository;  **use** Doctrine\ORM\EntityManagerInterface;  **use** Symfony\Bundle\FrameworkBundle\Controller\AbstractController;  **use** Symfony\Component\HttpFoundation\JsonResponse;  **use** Symfony\Component\HttpFoundation\Request;  **use** Symfony\Component\Routing\Annotation\Route;  /\*\*  \* Class PostController  \* @package App\Controller  \* @Route("/api", name="post\_api")  \*/  **class** PostController **extends** AbstractController  {  /\*\*  \* @param PostRepository $postRepository  \* @return JsonResponse  \* @Route("/posts", name="posts", methods={"GET"})  \*/  **public** **function** getPosts(PostRepository $postRepository){  $data = $postRepository->findAll();  **return** $this->response($data);  }  /\*\*  \* @param Request $request  \* @param EntityManagerInterface $entityManager  \* @param PostRepository $postRepository  \* @return JsonResponse  \* @throws \Exception  \* @Route("/posts", name="posts\_add", methods={"POST"})  \*/  **public** **function** addPost(Request $request, EntityManagerInterface $entityManager, PostRepository $postRepository){  **try**{  $request = $this->transformJsonBody($request);  **if** (!$request || !$request->get('name') || !$request->request->get('description')){  **throw** **new** \Exception();  }  $post = **new** Post();  $post->setName($request->get('name'));  $post->setDescription($request->get('description'));  $entityManager->persist($post);  $entityManager->**flush**();  $data = [  'status' => 200,  'success' => "Post added successfully",  ];  **return** $this->response($data);  }**catch** (\Exception $e){  $data = [  'status' => 422,  'errors' => "Data no valid",  ];  **return** $this->response($data, 422);  }  }  /\*\*  \* @param PostRepository $postRepository  \* @param $id  \* @return JsonResponse  \* @Route("/posts/{id}", name="posts\_get", methods={"GET"})  \*/  **public** **function** getPost(PostRepository $postRepository, $id){  $post = $postRepository->find($id);  **if** (!$post){  $data = [  'status' => 404,  'errors' => "Post not found",  ];  **return** $this->response($data, 404);  }  **return** $this->response($post);  }  /\*\*  \* @param Request $request  \* @param EntityManagerInterface $entityManager  \* @param PostRepository $postRepository  \* @param $id  \* @return JsonResponse  \* @Route("/posts/{id}", name="posts\_put", methods={"PUT"})  \*/  **public** **function** updatePost(Request $request, EntityManagerInterface $entityManager, PostRepository $postRepository, $id){  **try**{  $post = $postRepository->find($id);  **if** (!$post){  $data = [  'status' => 404,  'errors' => "Post not found",  ];  **return** $this->response($data, 404);  }  $request = $this->transformJsonBody($request);  **if** (!$request || !$request->get('name') || !$request->request->get('description')){  **throw** **new** \Exception();  }  $post->setName($request->get('name'));  $post->setDescription($request->get('description'));  $entityManager->**flush**();  $data = [  'status' => 200,  'errors' => "Post updated successfully",  ];  **return** $this->response($data);  }**catch** (\Exception $e){  $data = [  'status' => 422,  'errors' => "Data no valid",  ];  **return** $this->response($data, 422);  }  }  /\*\*  \* @param PostRepository $postRepository  \* @param $id  \* @return JsonResponse  \* @Route("/posts/{id}", name="posts\_delete", methods={"DELETE"})  \*/  **public** **function** deletePost(EntityManagerInterface $entityManager, PostRepository $postRepository, $id){  $post = $postRepository->find($id);  **if** (!$post){  $data = [  'status' => 404,  'errors' => "Post not found",  ];  **return** $this->response($data, 404);  }  $entityManager->remove($post);  $entityManager->**flush**();  $data = [  'status' => 200,  'errors' => "Post deleted successfully",  ];  **return** $this->response($data);  }  /\*\*  \* Returns a JSON response  \*  \* @param array $data  \* @param $status  \* @param array $headers  \* @return JsonResponse  \*/  **public** **function** response($data, $status = 200, $headers = [])  {  **return** **new** JsonResponse($data, $status, $headers);  }  **protected** **function** transformJsonBody(\Symfony\Component\HttpFoundation\Request $request)  {  $data = **json\_decode**($request->getContent(), **true**);  **if** ($data === **null**) {  **return** $request;  }  $request->request->replace($data);  **return** $request;  }  } |

Here we defined the five routes:

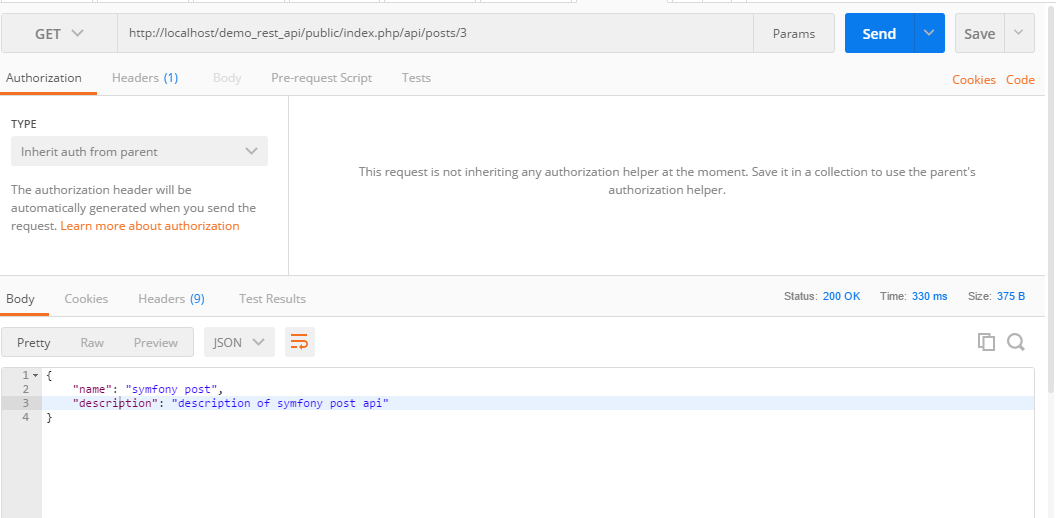
* GET /api/posts: will return a list of the Posts.



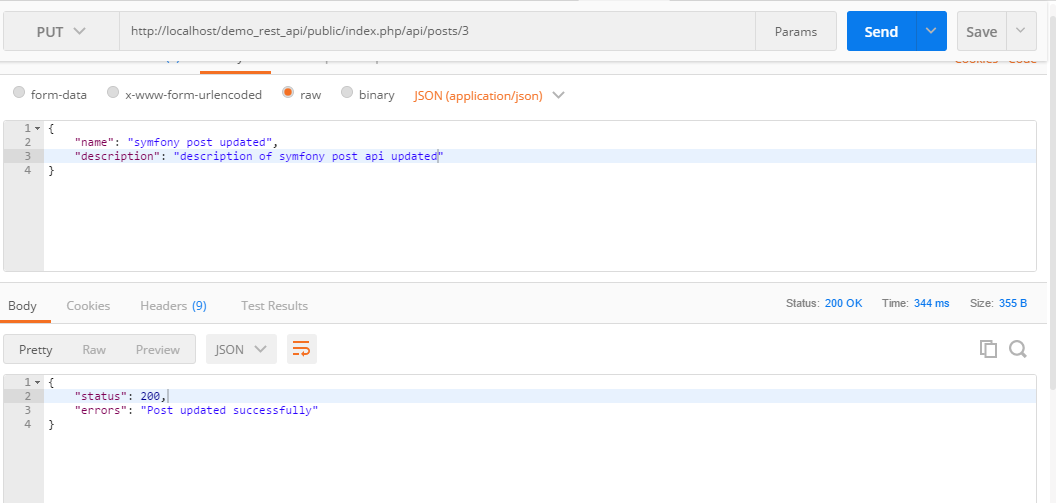
* POST /api/posts: will create a new Post.



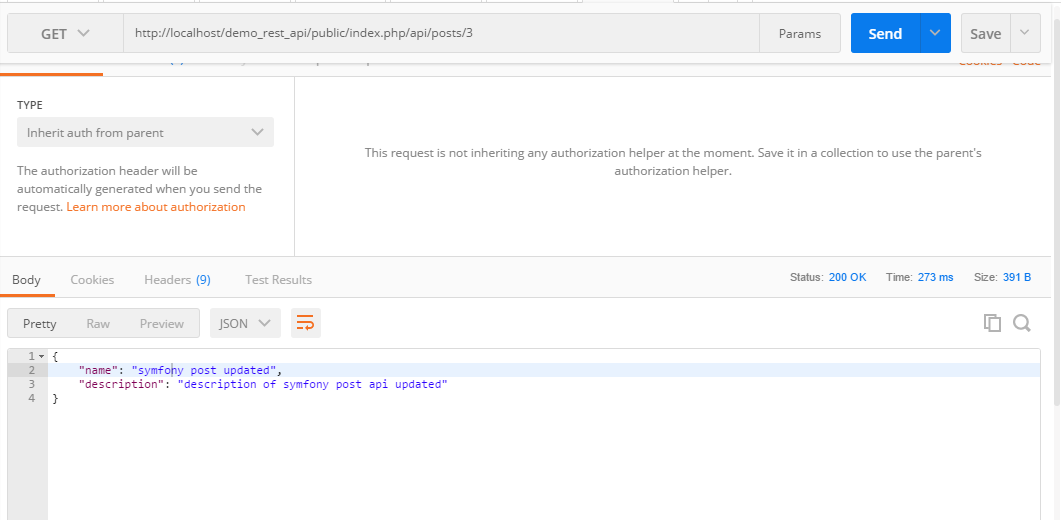
* GET /api/posts/id: will return a Post matching the specific identifer



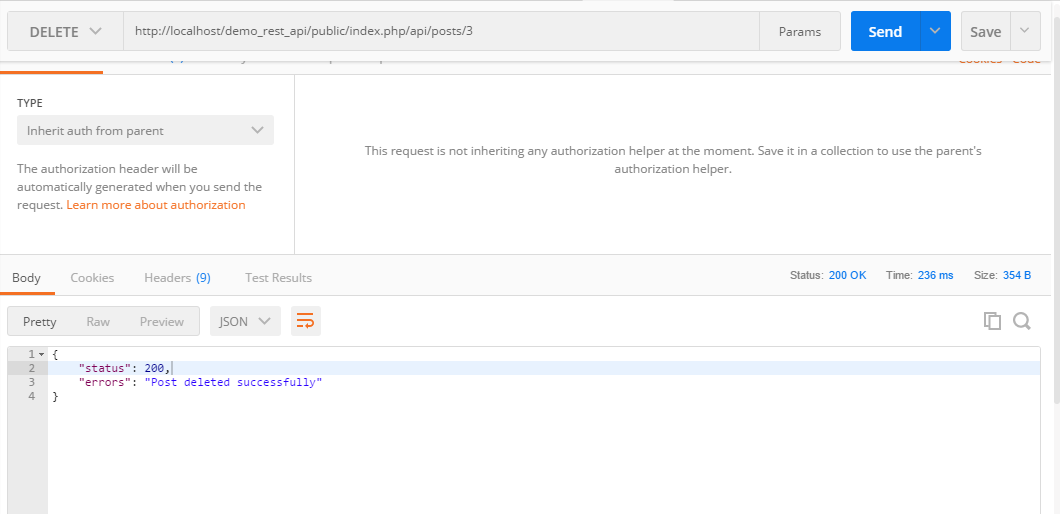
* PUT /api/posts/id : will update the Post.



This is the result after update:



* DELETE /api/posts/id: will delete the Post.



This is the result of get all posts after delete the post with the ID 3:

