

User profile management AIP : User documentation

In order to use or implement this basic project you should have installed the following:

- MySQL database
- Composer
- PHP >= 7.1
- Postman *optional
- Git *optional

Once we have checked the list we proceed to clone or download de project from the GitHub repository:

<https://github.com/sebastian-sulbaran/userApiProject.git>

After downloading, we can use Git console to move inside the project's folder.

First, we need to create a database to our project and modify the .env file in the App root folder, providing the credentials to connect to our database (As shown in the image 1).

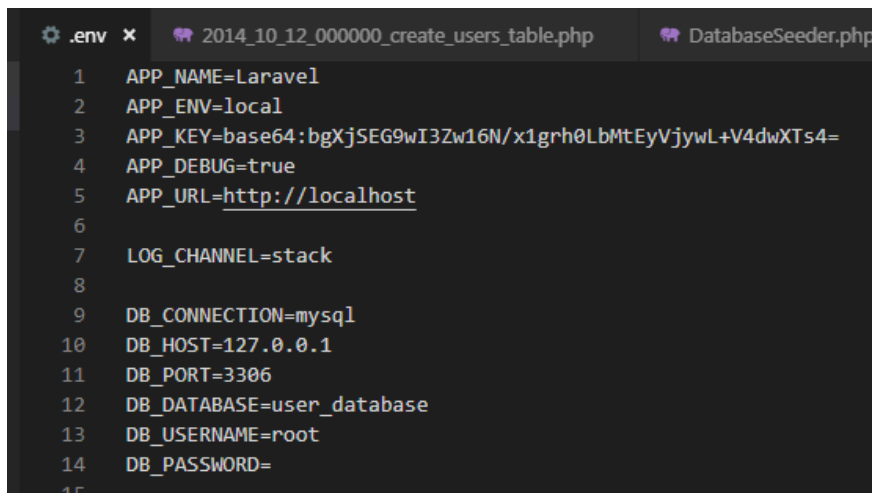


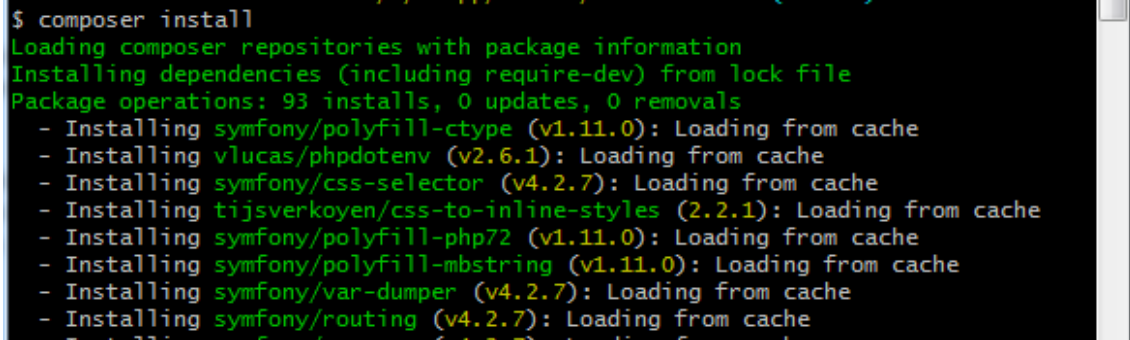
Image 1. Env file.

We have to fill the DB_ related variables with our info.

Once we have completed our info we can proceed to open our Git terminal and execute the command

```
composer install
```

This command will install all dependencies for us to let us work and test this project (As shown in image 2).

A terminal window with a black background and green text. The text shows the execution of the 'composer install' command. It starts with '\$ composer install', followed by 'Loading composer repositories with package information' and 'Installing dependencies (including require-dev) from lock file'. Then it says 'Package operations: 93 installs, 0 updates, 0 removals'. A list of packages follows, each preceded by a hyphen and the text 'Installing'. The packages listed are: symfony/polyfill-ctype (v1.11.0), vlucas/phpdotenv (v2.6.1), symfony/css-selector (v4.2.7), tijsverkoyen/css-to-inline-styles (2.2.1), symfony/polyfill-php72 (v1.11.0), symfony/polyfill-mbstring (v1.11.0), symfony/var-dumper (v4.2.7), and symfony/routing (v4.2.7). Each package name is highlighted in green, and the version number is in yellow. The status 'Loading from cache' is shown for each package.

```
$ composer install
Loading composer repositories with package information
Installing dependencies (including require-dev) from lock file
Package operations: 93 installs, 0 updates, 0 removals
- Installing symfony/polyfill-ctype (v1.11.0): Loading from cache
- Installing vlucas/phpdotenv (v2.6.1): Loading from cache
- Installing symfony/css-selector (v4.2.7): Loading from cache
- Installing tijsverkoyen/css-to-inline-styles (2.2.1): Loading from cache
- Installing symfony/polyfill-php72 (v1.11.0): Loading from cache
- Installing symfony/polyfill-mbstring (v1.11.0): Loading from cache
- Installing symfony/var-dumper (v4.2.7): Loading from cache
- Installing symfony/routing (v4.2.7): Loading from cache
```

Image 2. Composer install command execution.

After composer installs all the required dependencies, we need to execute the command

```
php artisan migrate:refresh --seed
```

This command will be executed with no problem if we have provided the right credentials in the .env file. The migrate command will deploy the user database and the parameter seed will add few users to the database (In order to query them later).

Now it's time to run the local PHP server with the command

```
php artisan serve
```

This command will deploy a http server to access to our project.

Testing our project

Now that we have our project running in a local environment, we can use postman tool to access the api function implemented in this project.

We will start doing our series of tests, creating a new user, to access this functionality we have to lead our post request to the following direction:

```
localhost:8000/api/users
```

We have to configure our post request as follows:

localhost:8000/api/users

POST localhost:8000/api/users Send Save

Params Authorization Headers **Body** Pre-request Script Tests Cookies Code Comments

☐ none ☒ form-data ☐ x-www-form-urlencoded ☐ raw ☐ binary

KEY	VALUE	DESCRIPTION	...	Bulk Edit
<input checked="" type="checkbox"/> name	Sebas New TEst			
<input checked="" type="checkbox"/> email	sebastes4111t1@wolox.com			
<input checked="" type="checkbox"/> Image	gmail.jpg X			
Key	Value	Description		

Response

Image 3. First test.

As we can see, there is a field for a profile image. This will create a new user in our project and optionally we can upload an image. If everything is ok we should be getting a 200 OK message with the new user created:

PUT localhost:8000/api/users Bootc DEL localhost:8000/api/users **POST localhost:8000/api/users** PUT localhost:8000/api/users GET localhost:8000/api/users PUT localhost:8000/api/users PUT localhost:8000/api/users GET localhost:8000/api/users + ... No Environment

POST localhost:8000/api/users Send

Pretty Raw Preview **JSON** 🔗

```

1 {
2   "name": "Sebas New TEst",
3   "email": "sebastes4111t1@wolox.com",
4   "api_token": "qLXYx4XwmASdK6tMMRh0ZOCPTFXir5FwwPNirdUcJZpVTIHN9rt4Mc1GPbC",
5   "image_path": "http://localhost:8000/images/9XJ5zSsFg5evPTudR64Dy4u3EyEFhItk15uYU3Js.jpeg",
6   "updated_at": "2019-05-27 16:18:29",
7   "created_at": "2019-05-27 16:18:29",
8   "id": 5
9 }

```

Image 4. First test answer.

Now we can test displaying a list of all users in the system sending a GET request to the url:

localhost:8000/api/users

If everything goes ok we have to receive a 200 OK message with the whole user list.

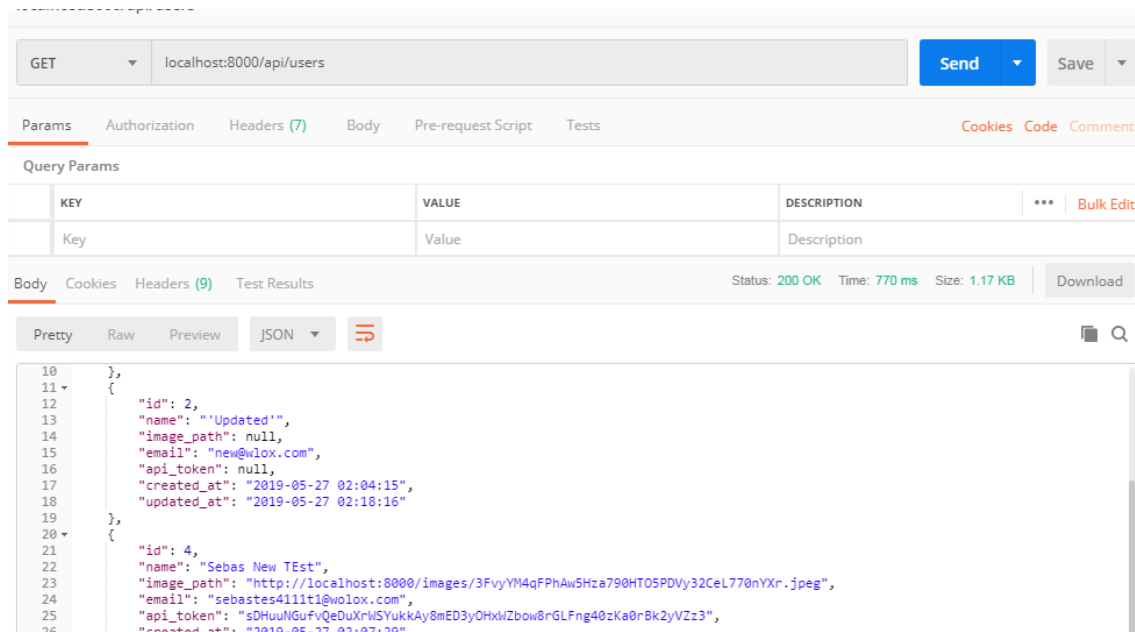


Image 5. Second test.

We can see our new user created in the user list.

Our third test is getting info relative to an user, we can accomplish this by requesting a GET to the follow url:

`localhost:8000/api/users/2`

We can see the answer en the follow image

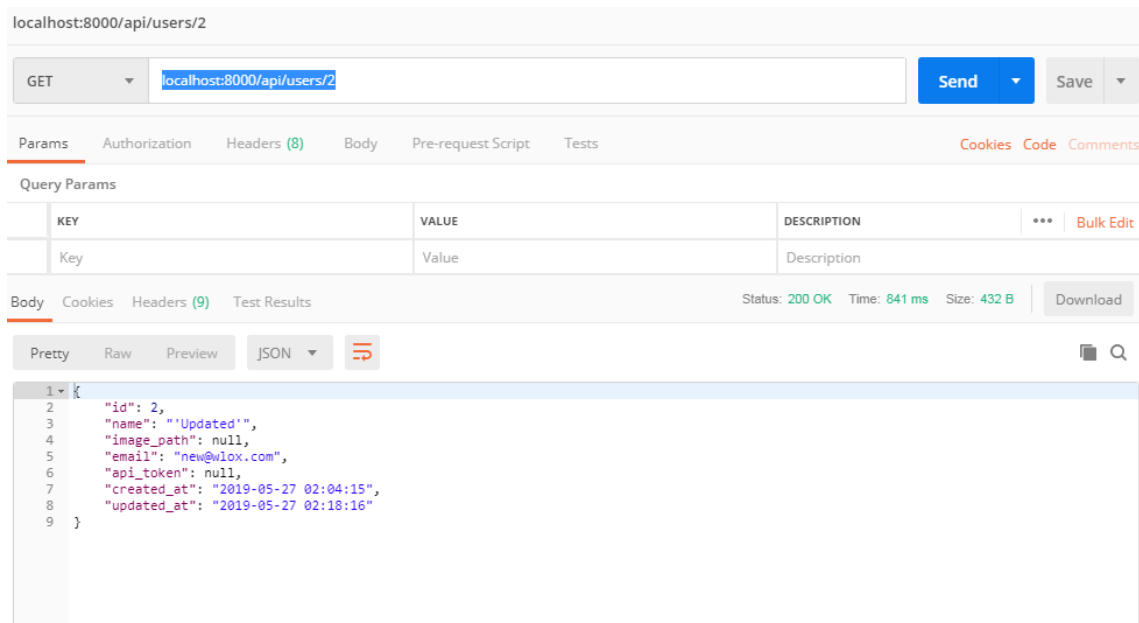


Image 6. Third test

This user can be updated with or forth test with a PUT request to the follow url:

We send the data as a raw json

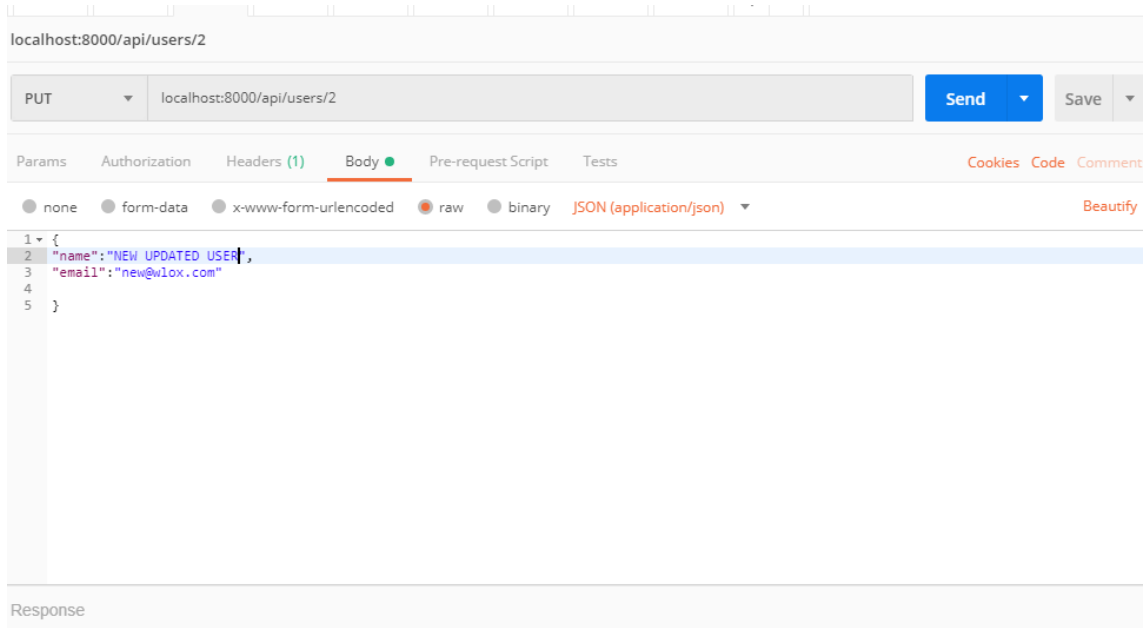


Image 7. Fourth Test.

And we get the answer:

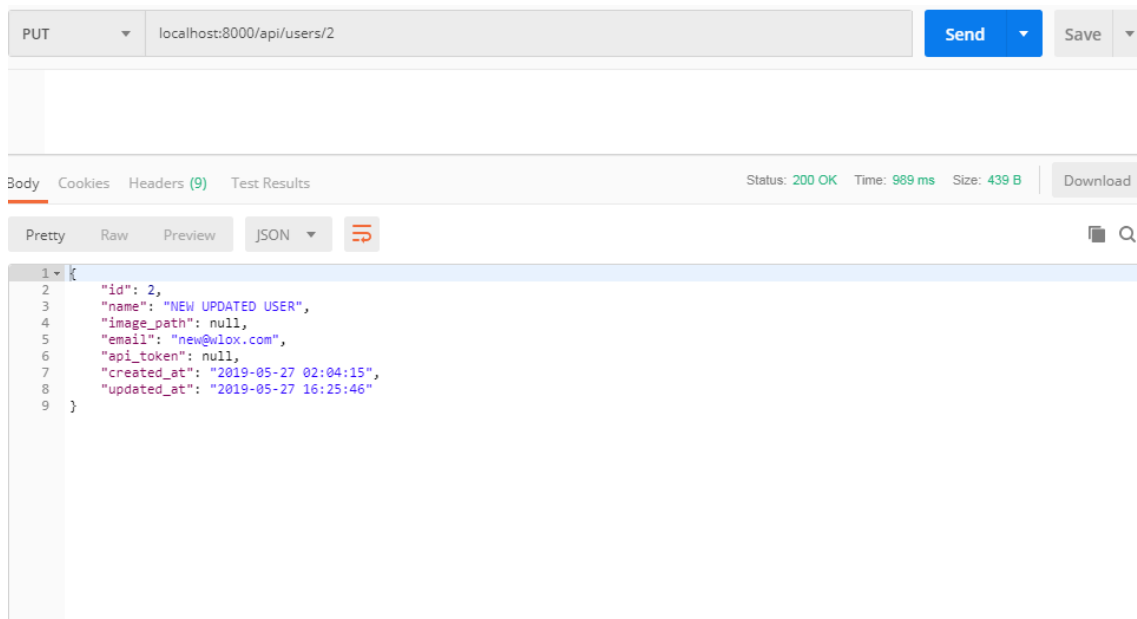


Image 8. Fourth test answer.

Our fifth and last test is deleting an user by sending a delete request to the specific user URL
`localhost:8000/api/users/2`

The answer is shown in the next image:

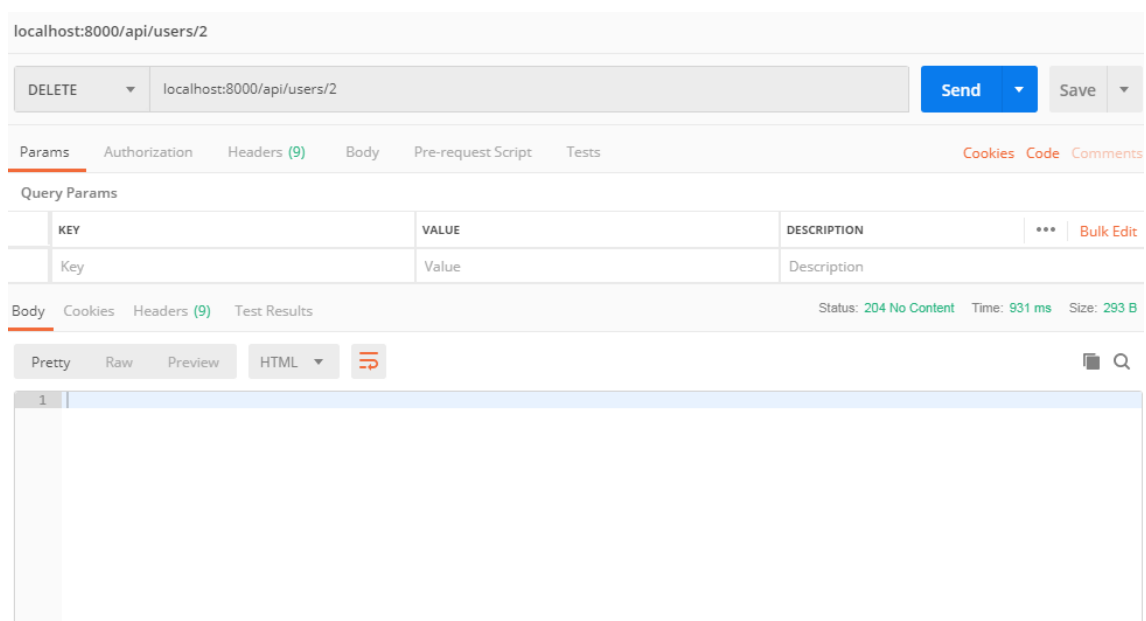


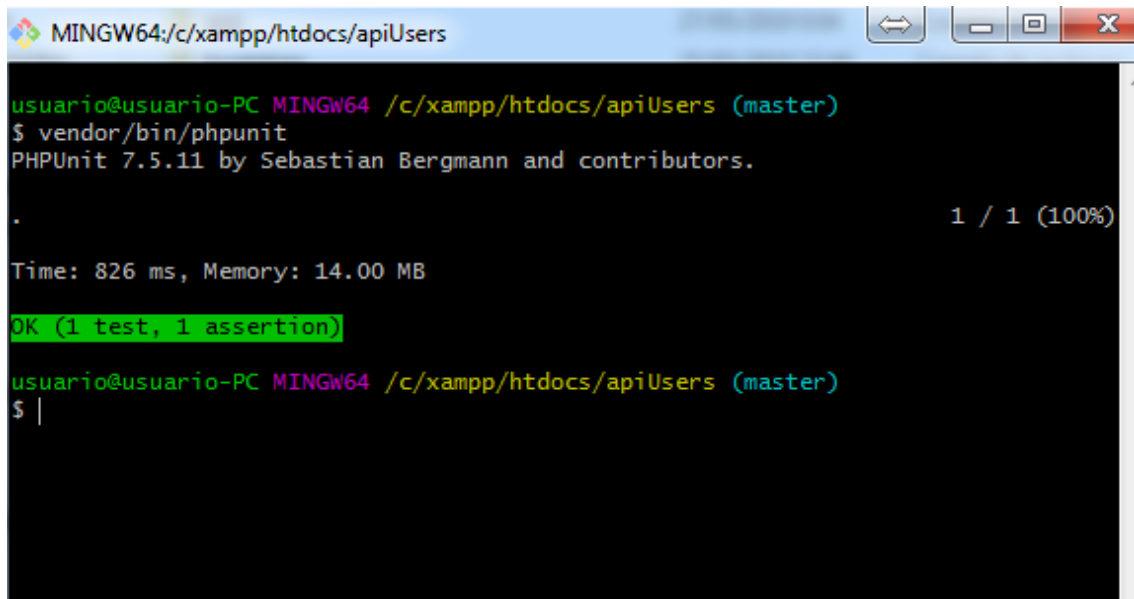
Image 9. Fifth Test

We received a 204 telling us that the resource was deleted.

Additionally, in this project was implemented one unit test, we can find it in the directory test in the root app directory.

To execute this unit test in the root app directory on a Git terminal, we need to execute this command:

```
vendor/bin/phpunit
```

A screenshot of a terminal window titled 'MINGW64:/c:/xampp/htdocs/apiUsers'. The terminal shows the command 'vendor/bin/phpunit' being executed. The output indicates that the test passed: 'PHPUnit 7.5.11 by Sebastian Bergmann and contributors.', '1 / 1 (100%)', 'Time: 826 ms, Memory: 14.00 MB', and 'OK (1 test, 1 assertion)'. The 'OK' text is highlighted in green. The prompt 'usuario@usuario-PC MINGW64 /c:/xampp/htdocs/apiUsers (master)' is visible at the top and bottom of the terminal.

```
MINGW64:/c:/xampp/htdocs/apiUsers
usuario@usuario-PC MINGW64 /c:/xampp/htdocs/apiUsers (master)
$ vendor/bin/phpunit
PHPUnit 7.5.11 by Sebastian Bergmann and contributors.

.                                                                1 / 1 (100%)
Time: 826 ms, Memory: 14.00 MB
OK (1 test, 1 assertion)
usuario@usuario-PC MINGW64 /c:/xampp/htdocs/apiUsers (master)
$ |
```

Image 10. Unit test.

This is a simple test that request /api/users and expects an 200 OK message.

```
<?php

namespace Tests\Feature;

use Tests\TestCase;
use Illuminate\Foundation\Testing\WithFaker;
use Illuminate\Foundation\Testing\RefreshDatabase;

class UserModuleTest extends TestCase
{
    /**
     * A basic feature test example.
     *
     * @return void
     */
    public function testExample()
    {
        $response = $this->json('GET', '/api/users');

        $response->assertStatus(200);
    }
}
```

Image 11. Test code.