THE AUTHENTICATION API

Before we actually end this section, let me show you how to create special api endpoints for authentication. We've setup Laravel passport and created endpoint for getting access token in the second lesson of this section. But that's not enough if you want to consume your apis for external use like mobile app or web app where you put your apis and user interface in separate application.

In this lesson I just want to show you how to create api endpoint for login, logout and register only. The reset password and account activation are beyond the scope of this lesson.

Alright, let's go ahead and open up our terminal. And then let's create a new branch for this lesson.

```
git checkout -b lesson-55
```

THE LOGIN ENDPOINT

1. Creating Login Endpoint

Let's start by creating api endpoint for logging in the user. In terminal we're going to say:

```
php artisan make:controller Api/Auth/LoginController
```

Note that I want to store my controller inside `Api/Auth`. You can store it in any place you prefer. Also you can call the controller to whatever name you like. In my case I call it `LoginController`.

If you remember we have created `getToken` method in `Controllers/Auth/LoginController.php` to handle access token on the fly. So let's move that method to our newly created controller, then rename it to `store`.

```
# Api/Auth/LoginController.php
<?php

namespace App\Http\Controllers\Api\Auth;

use App\Http\Controllers\Controller;
use Illuminate\Http\Request;

class LoginController extends Controller
{
    public function store(Request $request)
    {
        $request->request->add([
            'grant_type' => 'password',
            'client_id' => 2,
            'client_secret' =>
'cDMM2q9WAr0pSkSmjMPomjTzcL5o8bGEhWmLZ2My',
            'username' => $request->username,
            'password' => $request->password,
        ]);

        $tokenRequest = Request::create(env('APP_URL') .
'/oauth/token', 'post');

        $response = Route::dispatch($tokenRequest);

        return $response;
    }
}
```

Don't forget to import the `Route` namespace at the top before the `LoginController` class definition.

```
use Illuminate\Support\Facades\Route;
```

Also let's validate the inputs before requesting the access token.

```
public function store(Request $request)
{
    $request->validate([
        'username' => 'required|string',
        'password' => 'required|string',
    ]);

    // ...
}
```

And lastly let's go to `api.php` and define a `login` route like so:

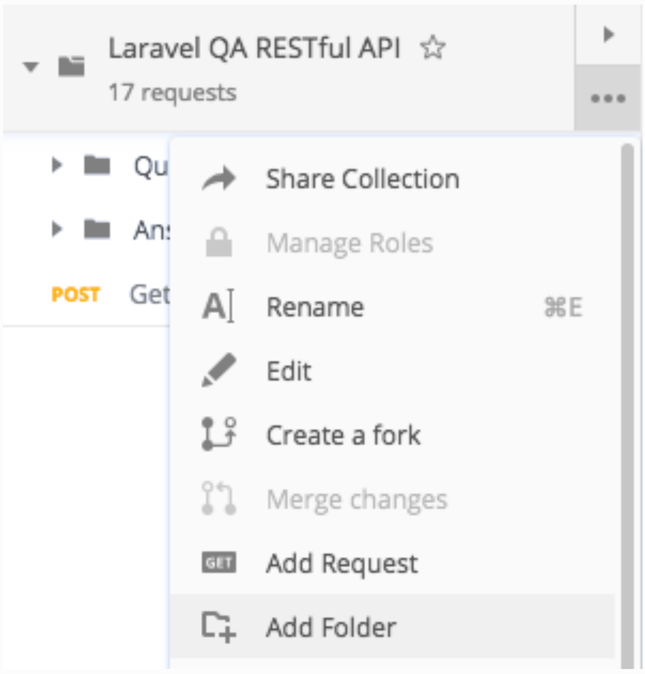
```
Route::post('/login', 'Api\Auth\LoginController@store');
```

You can get rid of the `token` route definition since it no longer needed.

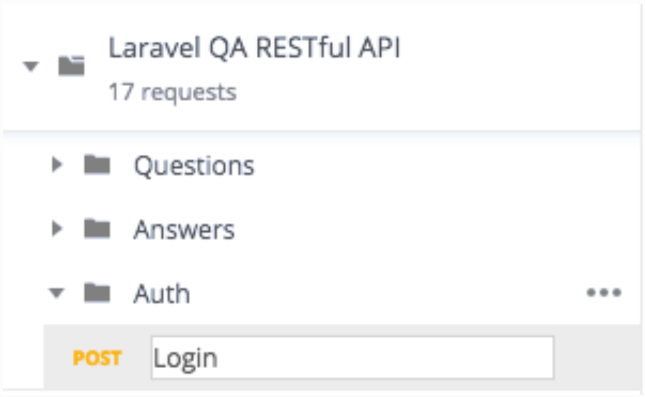
```
Route::post('/token', 'Auth\LoginController@getToken');
```

2. Testing the Login Endpoint

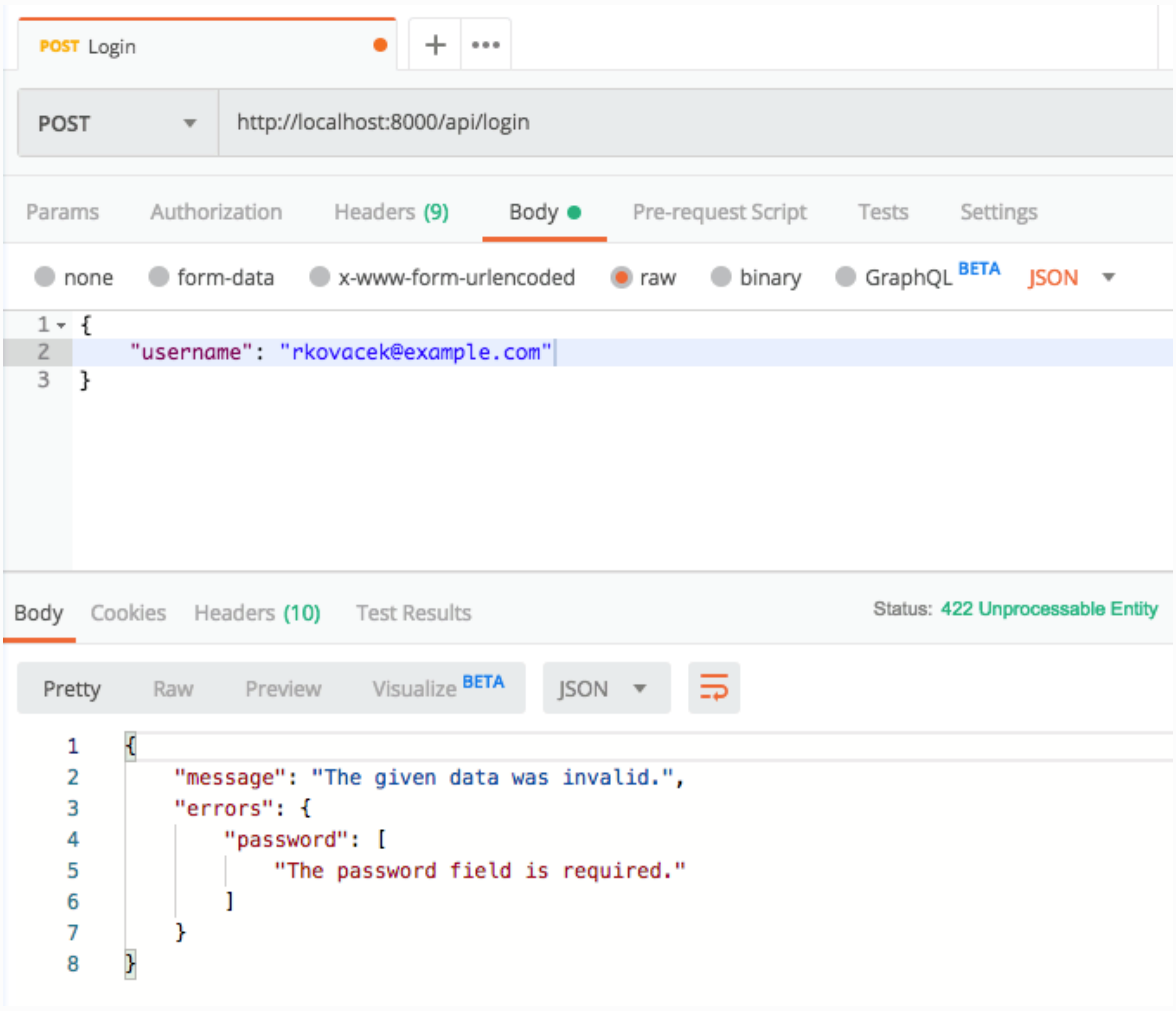
Let's go ahead and open up the Postman. Then create a new folder called `Auth` to hold our authentication api test requests.



We've already had the `Get Token` request which met all requirement for testing our `login` endpoint. So Let's move the `Get Token` request inside the `Auth` folder and rename it to `Login`.



All you need to do is to change the url from `/api/token` to `/api/login`. In **Body** section you can optionally remove the `username` or `password` to test that the validation is working.



POST http://localhost:8000/api/login

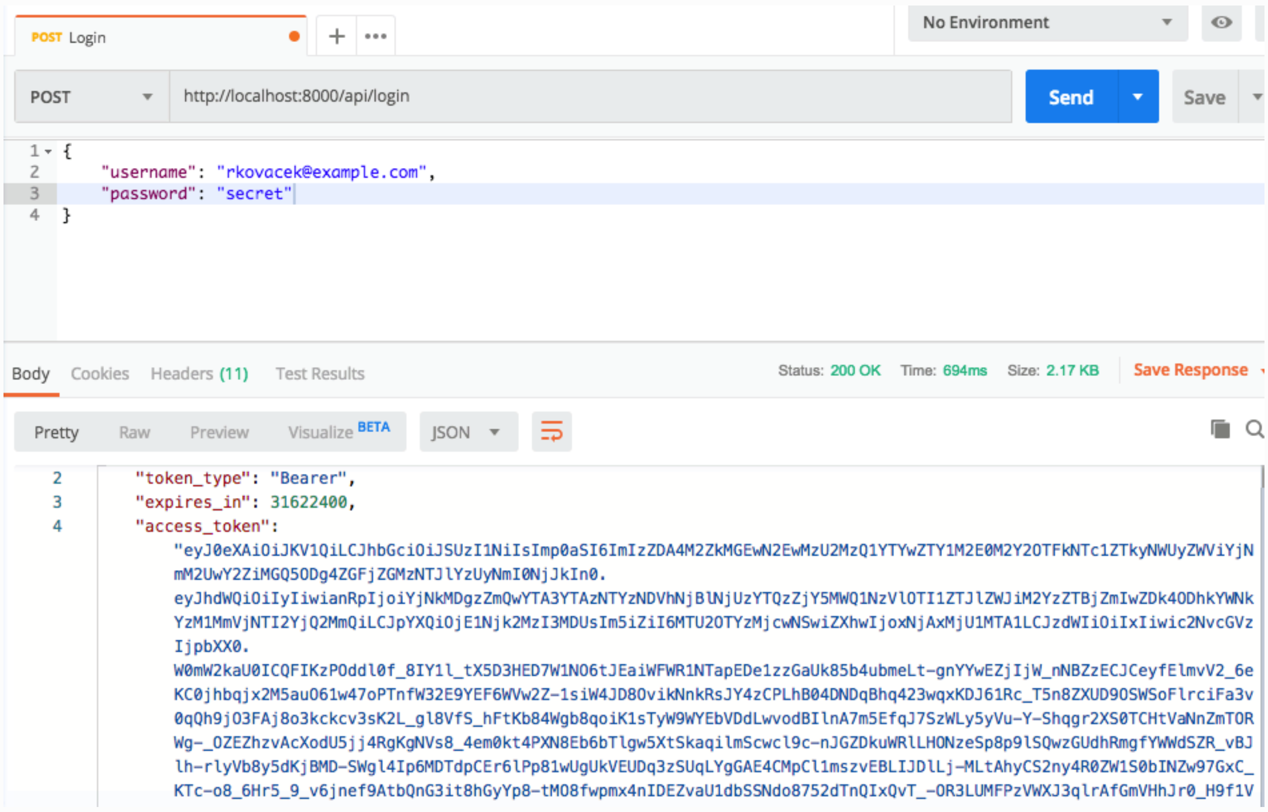
Body: [✓ raw] JSON

```
{
    "username": "rkovacek@example.com"
}
```

Status: 422 Unprocessable Entity

```
{
    "message": "The given data was invalid.",
    "errors": {
        "password": [
            "The password field is required."
        ]
    }
}
```

When you enter the username and password you will get OK status and get access token back.



http://localhost:8000/api/login

```
{
    "username": "rkovacek@example.com",
    "password": "secret"
}
```

Status: 200 OK Time: 694ms Size: 2.17 KB

```
    "token_type": "Bearer",
    "expires_in": 31622400,
    "access_token":
        "eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsImp0aSI6ImIzZDA4M2ZkMGEwN2EwMzU2MzQ1YTYwZTY1M2E0M2Y2OTFkNTc1ZTkyNWUyZWViYjN
        mM2UwY2ZiMGQ5ODg4ZGFjZGMzNTJlYzUyNmI0NjJkIn0.
        eyJhdWQiOiIyIiwianRpIjoiYjNkMDgzZmQwYTA3YTAzNTYzNDVhNjBlNjUzYTQzZjY5MWQ1NzVlOTI1ZTJlZWJiM2YzZTBjZmIwZDk4ODhkYWNk
        YzM1MmVjNTI2YjQ2MmQiLCJpYXQiOjE1Njk2MzI3MDUsIm5iZiI6MTU2OTYzMjcwNSwiZXhwIjoxNjAxMjU1MTA1LCJzdWIiOiIxIiwic2NvcGVz
        IjpbXX0.
        W0mW2kaU0ICQFIKzPOddl0f_8IY1l_tX5D3HED7W1NO6tJEaiWFWR1NTapEDe1zzGaUk85b4ubmeLt-gnYYwEZjIjW_nNBZzECJCeyfElmvV2_6e
        KC0jhbqjx2M5au061w47oPTnfW32E9YEF6WVw2Z-1siW4JD8OvikNnkRsJY4zCPLhB04DNDqBhq423wqxKDJ61Rc_T5n8ZXUD9OSWSoFlrciFa3v
        0qQh9jO3FAj8o3kckcv3sK2L_gl8VfS_hFtKb84Wgb8qoiK1sTyW9WYEbVDdLwvodBIlnA7m5EfqJ7SzWLy5yVu-Y-Shqgr2XS0TCHtVaNnZmTOR
        Wg-_OZEZhzvAcXodU5jj4RgKgNVs8_4em0kt4PXN8Eb6bTlgw5XtSkaqilmScwcl9c-nJGZDkuWRlLHONzeSp8p9lSQwzGUdhRmgfYWWdSZR_vBJ
        lh-rlyVb8y5dKjBMD-SWgl4Ip6MDTdpCEr6lPp81wUgUkVEUDq3zSUqLYgGAE4CMpCl1mszvEBLIJDlLj-MLtAhyCS2ny4R0ZW1S0bINZw97GxC_
        KTc-o8_6Hr5_9_v6jnef9AtbQnG3it8hGyYp8-tMO8fwpmx4nIDEZvaU1dbSSNdo8752dTnQIxQvT_-OR3LUMFPzVWXJ3qlrAfGmVHhJr0_H9f1V
```

THE LOGOUT ENDPOINT

1. Creating The Logout Endpoint

Let's go back to `Api/Auth/LoginController.php`. Inside this file let's define another method called `destroy`. In that method we can get the access token belongs to current user login by calling `$request->user()->token()`. And then we can revoke the token by chaining in the `revoke` method.

We can then return *204 No content* status by returning `response()->noContent()`.

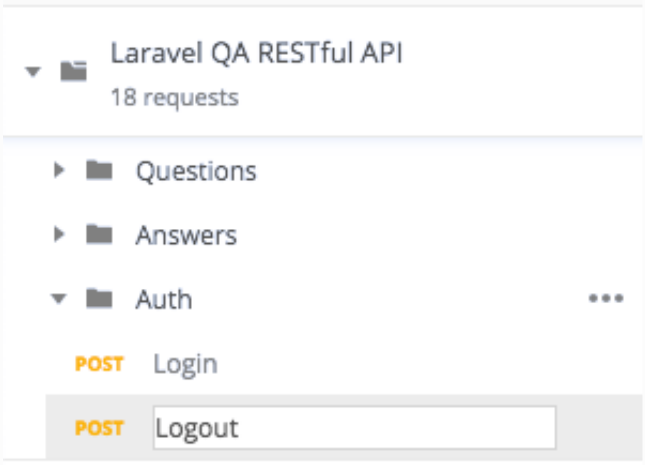
```
class LoginController extends Controller
{
    // ...
    public function destroy(Request $request)
    {
        $request->user()->token()->revoke();

        return response()->noContent();
    }
}
```

And lastly, let's go to `api.php` then define the `logout` route. This route can only be accessed by logged in user. So we need to call `auth:api` middleware.

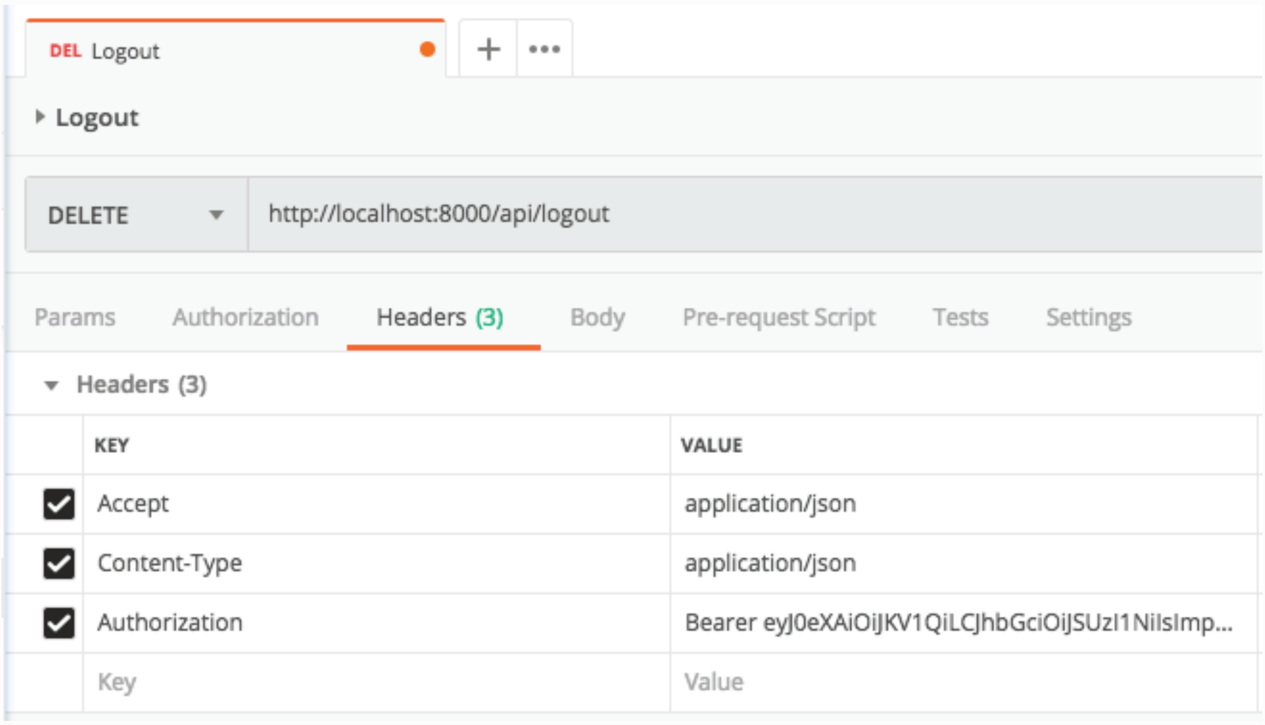
```
Route::delete('/logout', 'Api\Auth\LoginController@destroy')-
>middleware('auth:api');
```

2. Testing The Logout Endpoint

In Postman we can duplicate `Login` request then rename it to `Logout`.

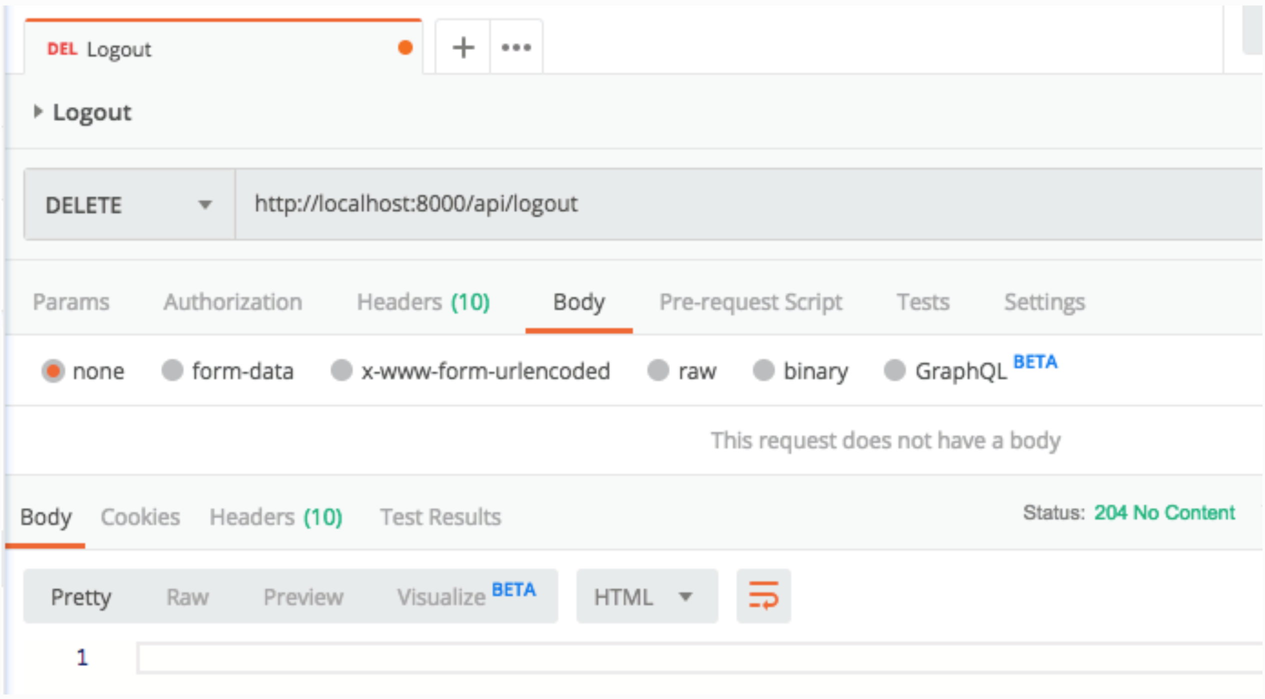


Change the http method to `DELETE` and change the url to `/api/logout`. Since we protected our route in `auth:api` we need to specify the `Authorization` in the **Headers** section and get the access token from the `Login` request.



	KEY	VALUE
✓	Accept	application/json
✓	Content-Type	application/json
✓	Authorization	Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsImp...
	Key	Value

You can optionally remove the request body in **Body** section by choosing the **none** option. When you hit the **Send** button you'll see 204 No content.


DEL Logout

Logout

DELETE http://localhost:8000/api/logout

This request does not have a body

Status: 204 No Content

THE REGISTER ENDPOINT

1. Creating The Register Endpoint

Let's head over to our terminal and create a brand new controller. Here I'll call it `RegisterController` and I also specify `-i` flag to tell artisan to make invokable controller.

```
php artisan make:controller Api/Auth/RegisterController -i
```

All logics for registering new user can be found in `Controllers/Auth/RegisterController.php` . So let's open that file and copy everything inside `create` method.

Let's open `Api/Auth/RegisterController.php` then paste the previous code inside the `__invoke` method.

```
public function __invoke(Request $request)
{
    return User::create([
        'name' => $data['name'],
        'email' => $data['email'],
        'password' => Hash::make($data['password']),
    ]);
}
```

Back to `Controllers/Auth/RegisterController.php` then copy the validation rules in `validator` method. In the `__invoke` method let's call `$request->validate()` and pass the validation rules in.

```
public function __invoke(Request $request)
{
    $data = $request->validate([
        'name' => ['required', 'string', 'max:255'],
        'email' => ['required', 'string', 'email', 'max:255',
'unique:users'],
        'password' => ['required', 'string', 'min:8'],
    ]);

    // ...
}
```

Since we call the `User::create()` and `Hash::make()` we need to import the `User` as well as `Hash` namespaces at the top before `RegisterController` class definition.

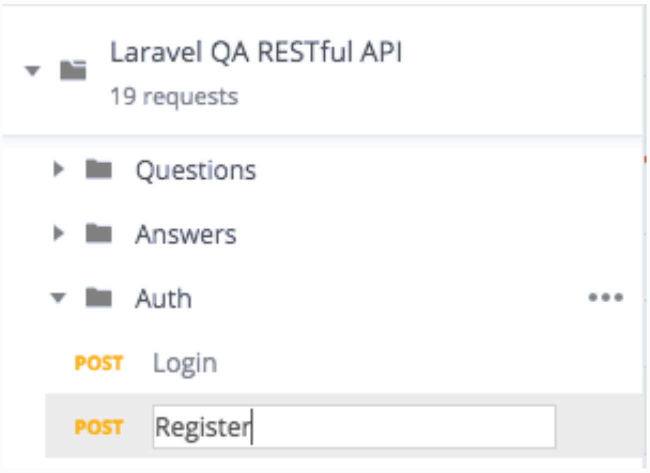
```
use Illuminate\Support\Facades\Hash;
use App\User;

class RegisterController extends Controller
{
    // ...
}
```

Last, let's define an api route for the `RegisterController` in `api.php`.

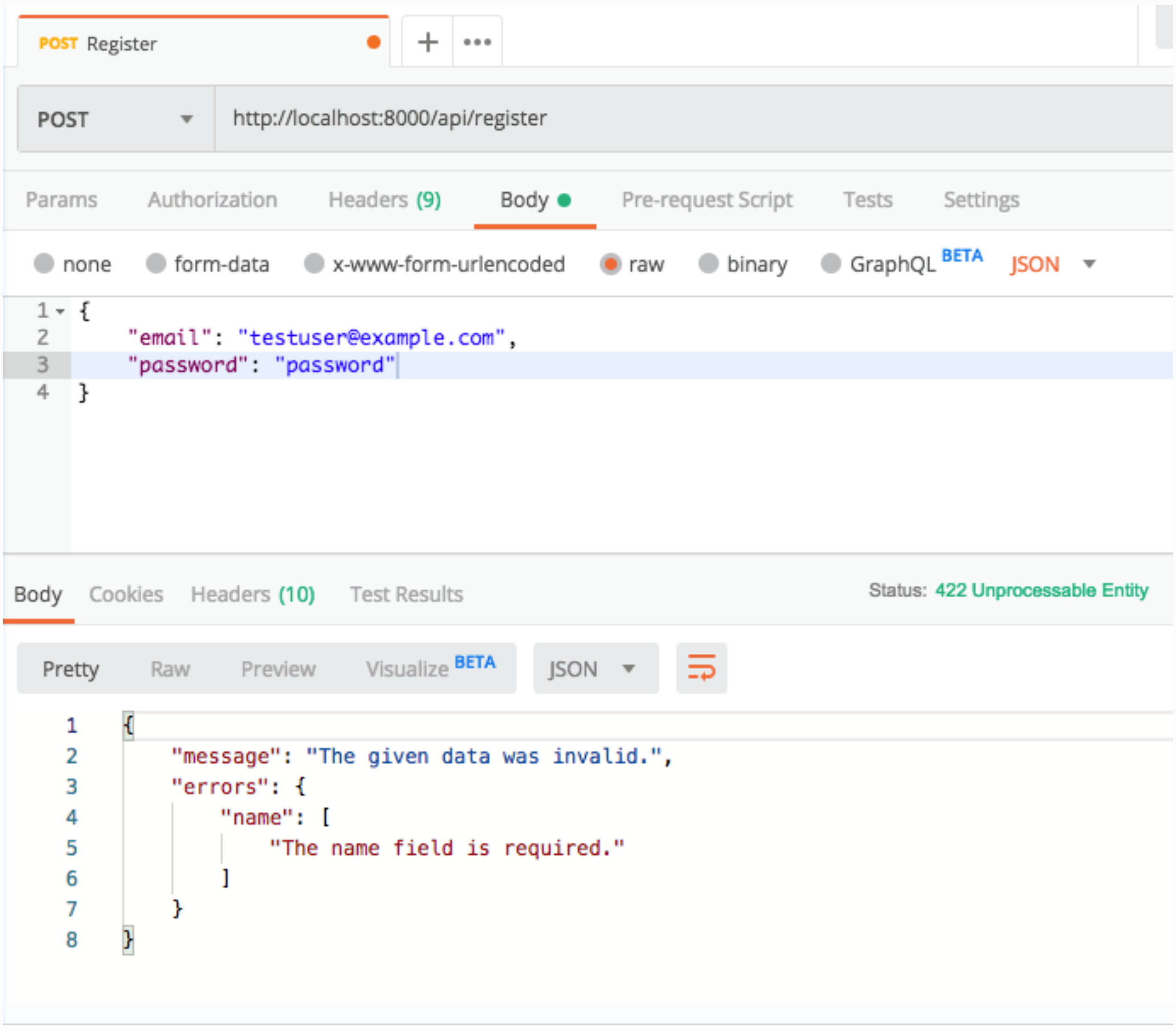
```
Route::post('/register','Api\Auth\RegisterController');
```

2. Testing The Register Endpoint

Back to the Postman then duplicate the `Login` request. And then rename it to `Register`.



Change the url to `/api/register`. Then go to **Body** section and specify the email, password and name. You can optionally ignore one or more input to make sure that the validation working.


POST Register

POST http://localhost:8000/api/register

Params | Authorization | Headers (9) | Body | Pre-request Script | Tests | Settings

none | form-data | x-www-form-urlencoded | [✓ raw] | binary | GraphQL BETA | JSON

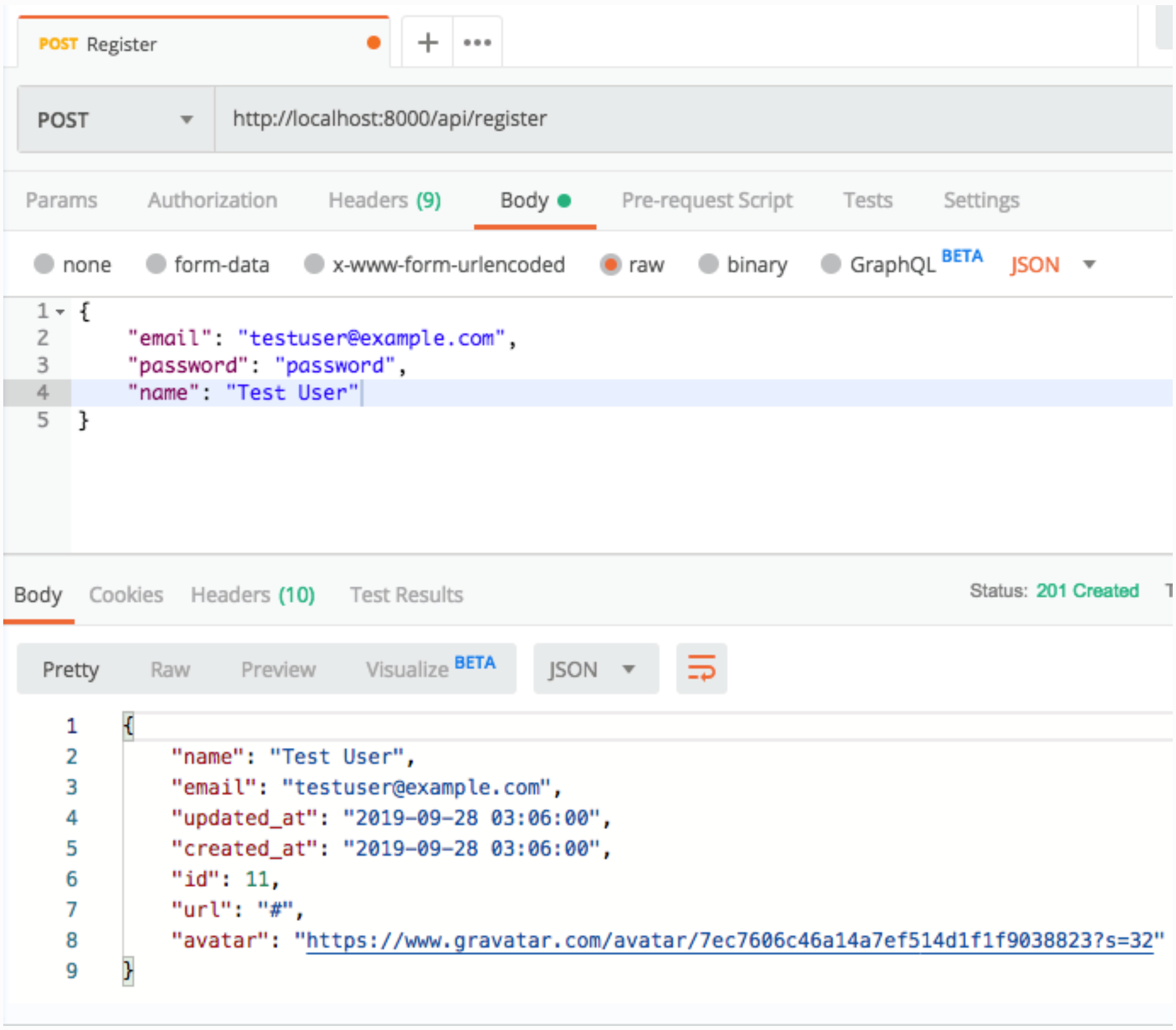
```
{
    "email": "testuser@example.com",
    "password": "password"
}
```

Body | Cookies | Headers (10) | Test Results — Status: 422 Unprocessable Entity

Pretty | Raw | Preview | Visualize BETA | JSON

```
{
    "message": "The given data was invalid.",
    "errors": {
        "name": [
            "The name field is required."
        ]
    }
}
```

But if you enter all those three values you will get back *201 Created* status as well as the created user you just created.


POST Register

POST http://localhost:8000/api/register

Params Authorization Headers (9) Body ● Pre-request Script Tests Settings

none form-data x-www-form-urlencoded ● raw binary GraphQL BETA JSON

```
{
    "email": "testuser@example.com",
    "password": "password",
    "name": "Test User"
}
```

Body Cookies Headers (10) Test Results Status: 201 Created

Pretty Raw Preview Visualize BETA JSON

```
{
    "name": "Test User",
    "email": "testuser@example.com",
    "updated_at": "2019-09-28 03:06:00",
    "created_at": "2019-09-28 03:06:00",
    "id": 11,
    "url": "#",
    "avatar": "https://www.gravatar.com/avatar/7ec7606c46a14a7ef514d1f1f9038823?s=32"
}
```

SUMMARY

In this lesson you've seen how to create api endpoints for authentication such as login, logout and register. As I've mentioned that these endpoints will helpful if you want to use your apis for external use such as mobile app or web app where the api and frontend in separate application. If you have your apis and user interface in single application then you can still use the default Laravel authentication.

Alright, let's go ahead and commit our changes that we made today into our git repo.

```
git add .
git commit -m "Create Api endpoints for Authentication"
git push origin lesson-55
git checkout master
git merge lesson-55
git push origin master
```