الجمهورية العربية السورية
جامعة دمشق
كلية الهندسة المعلوماتية

Programming Languages

Lecture Five

PHP Laravel – Authentication/Authorization & Middleware

Prepared By

Eng. Rawan Koroni

Eng. Ahmed Al-Ahmed

1- Authentication/Authorization

- Authentication is the process of identifying the user credentials. It provides access control for systems by checking to see if a user's credentials match the credentials in a database of authorized users or in a data authentication server. In doing this, authentication assures secure systems, secure processes and enterprise information security.
- Authorization is the function of specifying access rights/privileges to resources, which is related to general information security and computer security, and to access control in particular. More formally, "to authorize" is to define an access policy.
- In general web apps, we implement the authorization/ authentication in RESTful-based applications within the usage of TOKEN, token simply is an encoded/encrypted fixed length string of random characters contains some information about the current session/user who made the request.
- Because of Restful is a stateless so we cannot handle the session just like in normal/simple stateful applications, so we need to pass something from user to identify his session/ID and to provide him the ability to access the requested resource.

2- Authentication/Authorization Example

- If I logged in with the current credentials:
  Email: Ahmed@gmail.com
  Password: 12345678
- We need to pass some kind of data to the server to tell it that I already logged in, this type of data called token.

- Let's build a simple JSON object with some information for example:

- As we see, the JSON object does not contains the password & (in real applications, we do not pass the email as well, just a UUID)
- But to send this data we need to encode it, since it will be sent as request header not in the body, so we can apply any type of text encoding like BASE64, BASE32, …
- After encoding the above json content using any BASE64 online encoder, we will get the following string:

ewogICJlbWFpbCI6ICJhaG1lZEBnbWFpbC5jb20iLAogICJsb2dnZWRfYXQiOiAiMjAyMS8xMS8yNCIsCiAgImV4cGlyZWRfYXQiOiAiMjAyMS8xMS8zMCIsCiAgInVzZXJfcm9sZSI6ICJzdXBlcl9hZG1pbiIKfQ==

- Now, we can send this string within the request headers.

3- Middleware & Full Example

- In Laravel, it's a simple class, which handle the request before it arrives to the controller, so we usually use middleware to intercept the requests and filter them out in case of security issues like invalid tokens, absence of tokens, fake UUIDs, etc.
- We can create the middleware by running the command:

- And if we checked the generated middleware in App/Http/Middleware/ we will find the middleware with this default codebase:

- As we see there is simply a single function, this function takes 2 parameters, the first represent the current request object, and the second represent a simply closure/method used to pass the execution to the next middleware (if any) or to the target controller.

- Let’s add a simple controller with a single route as bellow:

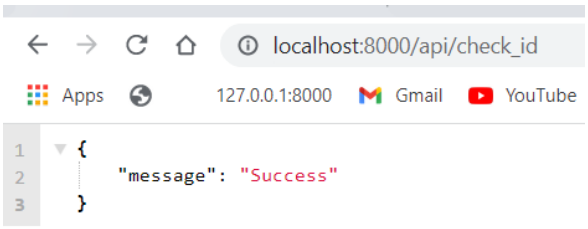
JSON File

```php
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class FirstController extends Controller
{
    public function check(Request $request)
    {
        return response()->json(['message' => 'Success']);
    }
}
```

Route

```php
<?php

use Illuminate\Support\Facades\Route;

Route::get('check_id', 'FirstController@check');
```

- If we test it using the browser, we will get the following response:



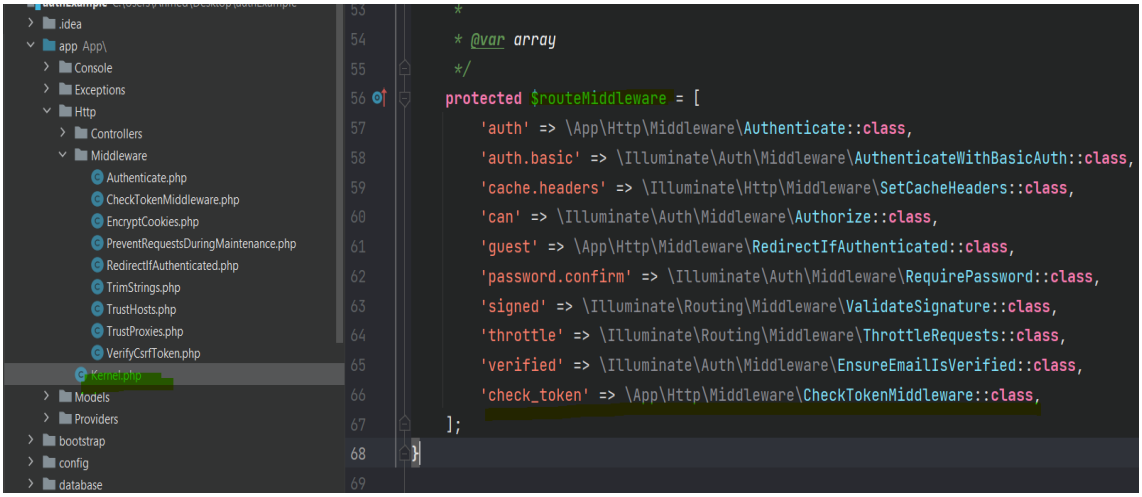
Which indicates that all is good & worked successfully!

- Let's suppose that this end-point does not allow everyone to access it, just who has a valid token, so let's make some changes.
- In middleware file, check if the X-ITE-Token is existed within the headers or not:

Route

```php
<?php

namespace App\Http\Middleware;

use Closure;
use Illuminate\Http\Request;

class CheckTokenMiddleware
{
    public function handle(Request $request, Closure $next)
    {
        if (!$request->hasHeader('X-ITE-TOKEN')) {
            return response()->json(['message' => 'Missing token'], 401);
        }
        return $next($request);
    }
}
```

- But to allow the controller to use this middleware, we should register this middleware in the Kernel file like bellow:

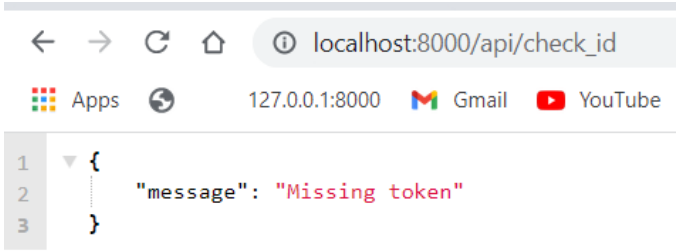


- We can assign any name to that middleware, now let's add this middleware to route:

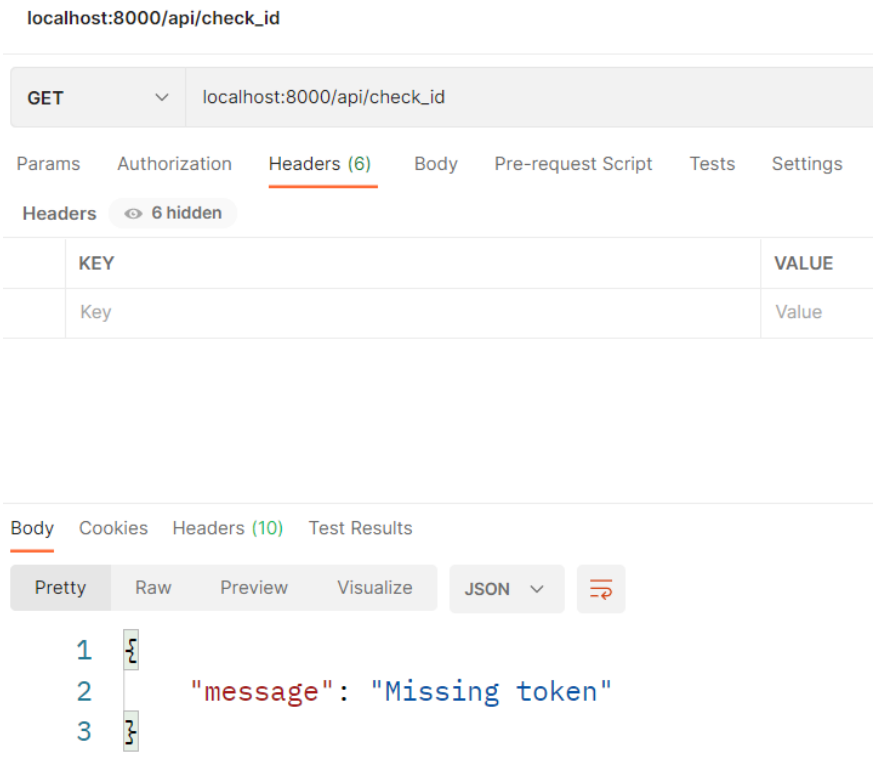
Route

```php
<?php

use Illuminate\Support\Facades\Route;

Route::get('check_id', 'FirstController@check')->middleware(['check_token']);
```

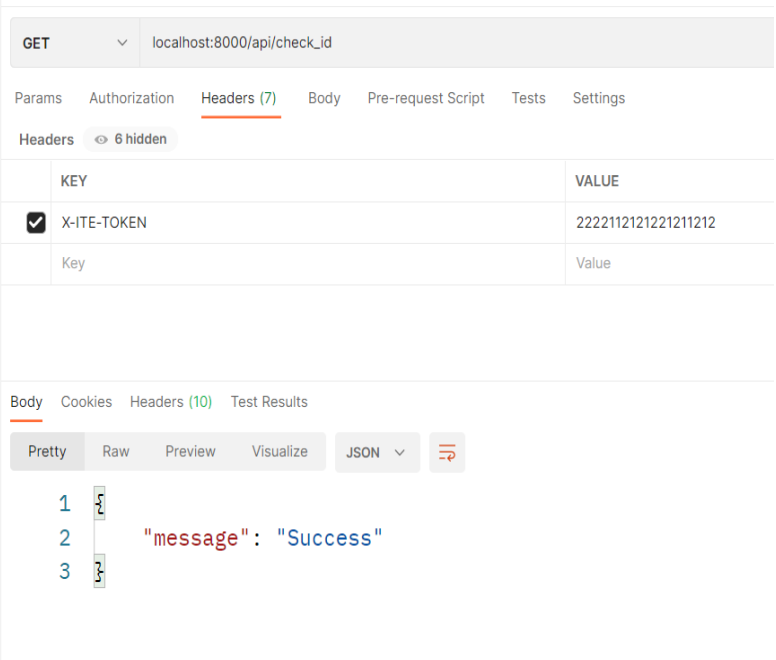
- We assigned this middleware to handle the requests to this end-point, so if we refresh the browser we will see the following response:



- Let's try using Postman:



- For sure, the same response, but if we tried to add any string within the requested header:

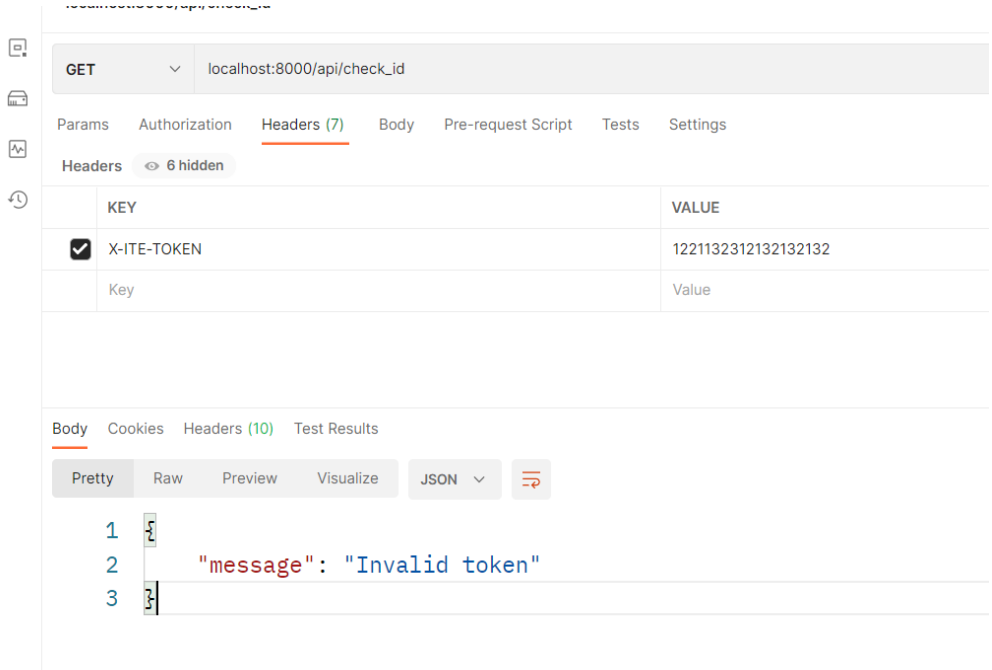


- Now all is good, we could access that end-point because we sent the header, let's add a simple validation on that token header and try again:

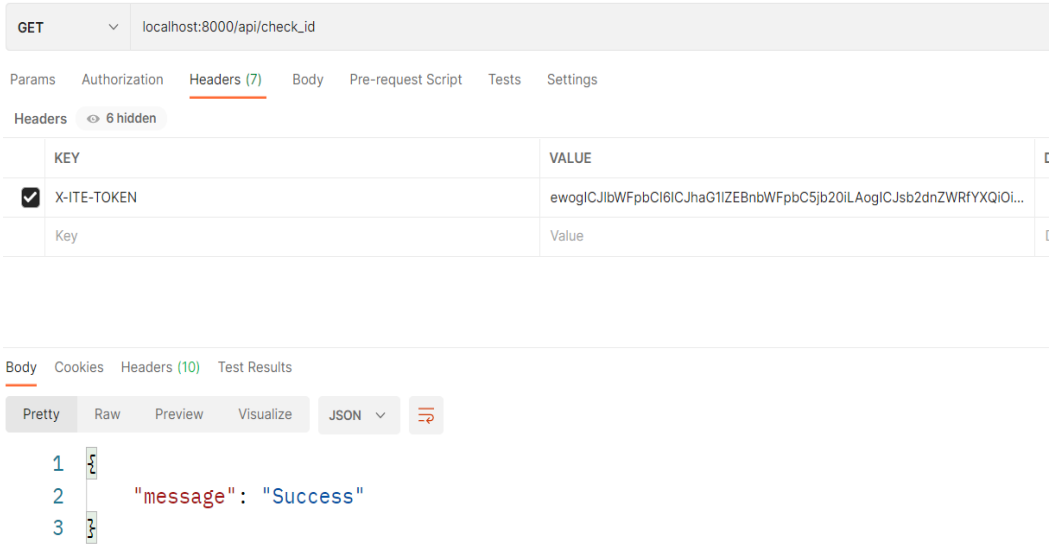
Middleware

```php
<?php

namespace App\Http\Middleware;

use Closure;
use Illuminate\Http\Request;

class CheckTokenMiddleware
{
    private $allowedEmails = [
      'ahmed@gmail.com',
    ];
    public function handle(Request $request, Closure $next)
    {
        $error = false;
        if (!$request->hasHeader('X-ITE-TOKEN')) {
            $error = true;
        }
        $token = $request->header('X-ITE-TOKEN');
        try {
            $jsonStr = base64_decode($token);
            $jsonPayload = json_decode($jsonStr, true);
            if (!$jsonPayload) {
                $error = true;
            }
            if (!isset($jsonPayload['email'])) {
                $error = true;
            }
            if (!in_array($jsonPayload['email'], $this->allowedEmails)) {
                $error = true;
            }
        }catch (\Exception $exception) {
            $error = true;
        }

        if ($error) {
            return response()->json(['message' => 'Invalid token'], 401);
        }
        return $next($request);
    }
}
```

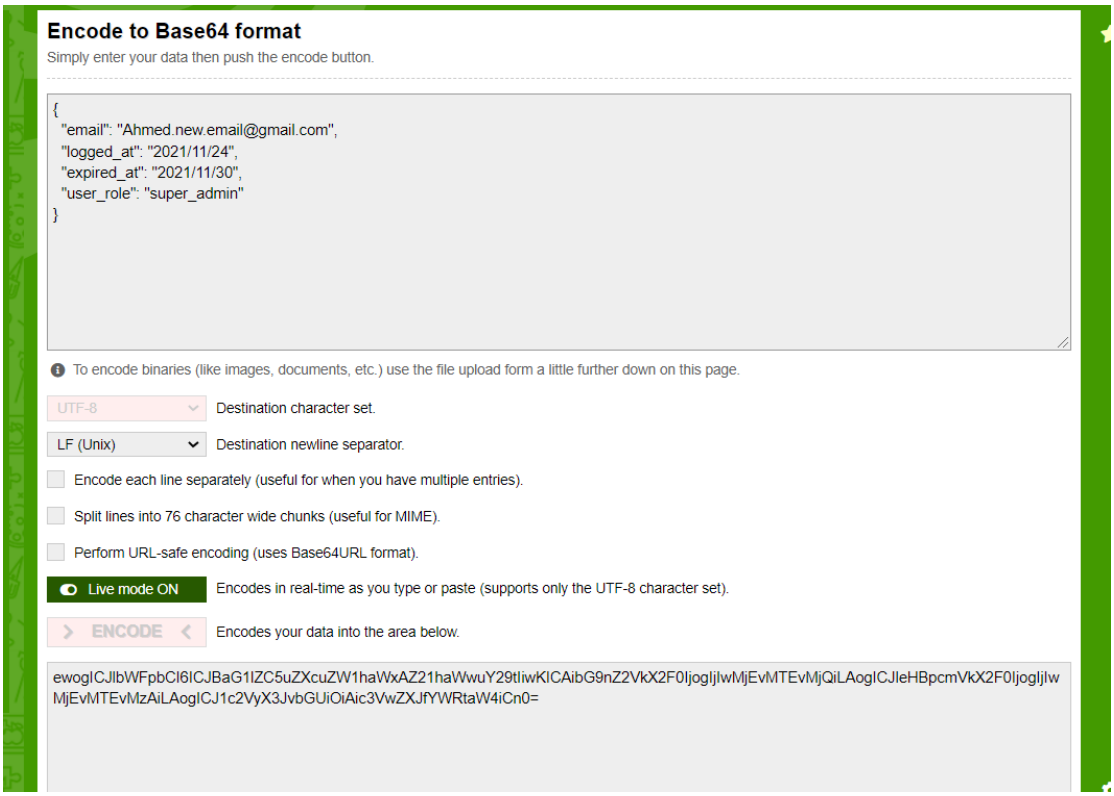
- Now, if we resent the same request from Postman, we will receive an error:

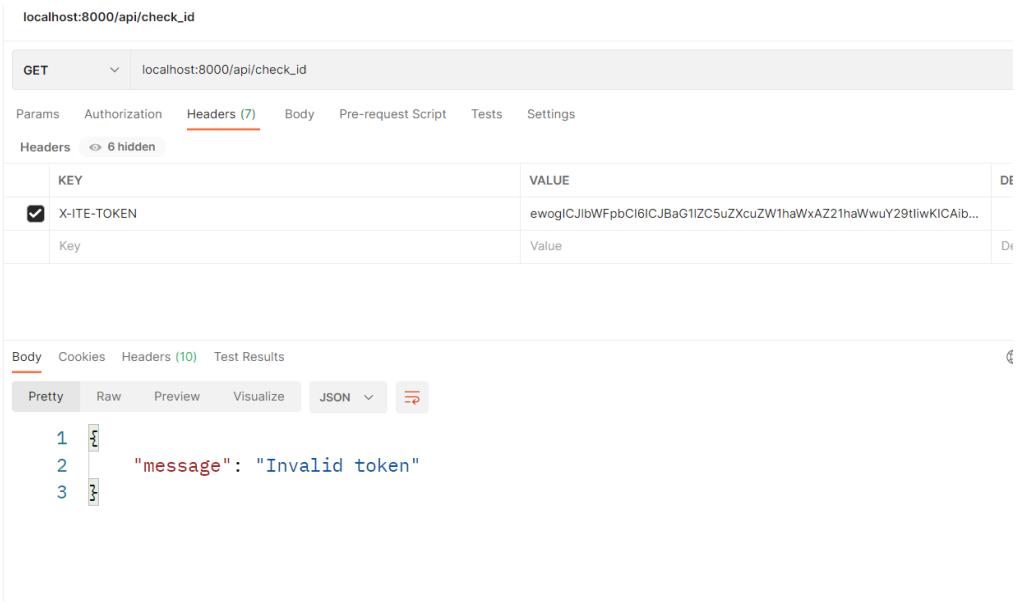


- Let's add the above token we got it and re-send the request:



- Let’s change the email in the token and re-send the request





- As we see, we could restrict the end-point to a specific set of emails.

4- Exercise

Implement the following scenario:

- We have a list of products, each one consists of: name, price, owner email. [JSON FILE]
- We have a list of user credentials [Email, Password] for many users in a new JSON FILE.
- We have 2 end-points:
  1. Login using a correct email & password, the response of success login should be a valid token for that user.
  2. Users with a valid token can delete products only if the user email matches the owner email of the requested product.