APIs in PHP from basics to Advanced

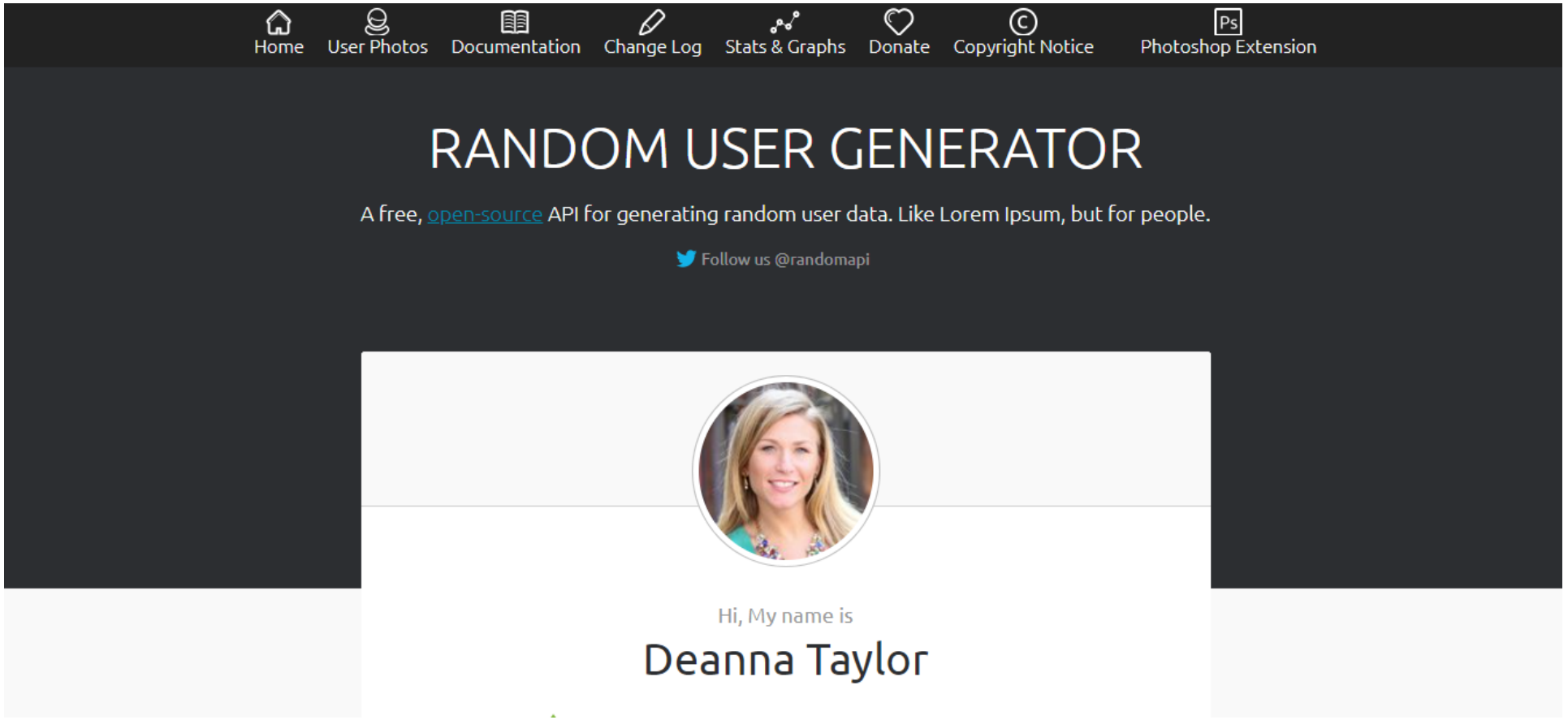
1. API basics: What APIs are and how to use them
2. HTTP basics: requests, and responses using CURL
3. REST and RESTful APIs: Using them from PHP
4. Create a RESTful API: build a framework for serving the API
5. Create a RESTful API: Create a database and retrieve data from it
6. Create a RESTful API: Create, update & delete individual resources
7. API key authentication
8. An Introduction to Authentication Using Access Tokens
9. An Introduction to Authentication Using JSON Web Tokens (JWTs)
10. Expiring and Refreshing Access Tokens

API basics: What APIs are and how to use them

What is an API?

An API is a means for CODE to retried data from a web server with no formatting whereas browsing a web will usually return HTML which is human readable code for display on the screen.

An example of a web interface that returns HTML is https://randomuser.me/



We want to access this data programmatically. We could build a web scraper that can parse the HTML and extract the data but this is error prone because if the underlying HTML elements change this the scraper code will not work.

This example website used for testing can also return just the raw data with no HTML for visual formatting. https://randomuser.me/api APIs just return data because they are for programs to use.


```
{"results":[{"gender":"female","name":{"title":"Ms","first":"Ava","last":"Murphy"},"location"
:{"street":{"number":7184,"name":"Kingsway"},"city":"Preston","state":"Lincolnshire","country
":"United Kingdom","postcode":"JP1H 9GU","coordinates":{"latitude":"-
9.9732","longitude":"97.6695"},"timezone":{"offset":"+8:00","description":"Beijing, Perth,
Singapore, Hong Kong"}},"email":"ava.murphy@example.com","login":{"uuid":"1aadb5b9-2553-4408-
8197-
5de5a5f055b7","username":"heavygoose855","password":"michigan","salt":"vupraIrh","md5":"97374
f6f0fda2b181850b0ea9fdaf9b9","sha1":"708b9ac1481c79055d1c48188f94d69ab8831d15","sha256":"f31d
b0c485169049c2680b2cc6a54ae9ac53f83f05a137b9582b9888427d24b6"},"dob":{"date":"1996-04-
07T15:19:57.915Z","age":26},"registered":{"date":"2008-09-
05T12:48:34.569Z","age":14},"phone":"01346 984632","cell":"07324
886423","id":{"name":"NINO","value":"NC 46 04 20
Z"},"picture":{"large":"https://randomuser.me/api/portraits/women/63.jpg","medium":"https://r
andomuser.me/api/portraits/med/women/63.jpg","thumbnail":"https://randomuser.me/api/portraits
/thumb/women/63.jpg"},"nat":"GB"}],"info":{"seed":"7dd130bbca8c5d09","results":1,"page":1,"ve
rsion":"1.4"}}
```


Users will access data using a browser and seeing visually formatted HTML code. Programs will access data using API's.

Make an API call to access an API from PHP

An API is a means for CODE to retried data from a web server with no formatting whereas browsing a web will usually return HTML which is human readable code for display on the screen.

```
<?php
$response = file_get_contents("https://example.com/");
echo htmlspecialchars($response);
?>
```

We can get the raw contents of a HTML webpage using php command file_get_contents.

```
<!doctype html> <html> <head> <title>Example Domain</title> <meta charset="utf-8" /> <meta http-equiv="Content-type" content="text/html; charset=utf-8" /> <meta name="viewport" content="width=device-width, initial-scale=1" /> <style type="text/css"> body { background-color: #f0f0f2; margin: 0; padding: 0; font-family: -apple-system, system-ui, BlinkMacSystemFont, "Segoe UI", "Open Sans", "Helvetica Neue", Helvetica, Arial, sans-serif; } div { width: 600px; margin: 5em auto; padding: 2em; background-color: #fdfdff; border-radius: 0.5em; box-shadow: 2px 3px 7px 2px rgba(0,0,0,0.02); } a:link, a:visited { color: #38488f; text-decoration: none; } @media (max-width: 700px) { div { margin: 0 auto; width: auto; } } </style> </head> <body> <div> <h1>Example Domain</h1> <p>This domain is for use in illustrative examples in documents. You may use this domain in literature without prior coordination or asking for permission.</p> <p><a href="https://www.iana.org/domains/example">More information...</a></p> </div> </body> </html>
```

We can also make an API call to the random user website using the same code. Note that a URL that returns code is commonly known as an API end point and usually has api at the end of the URL.

```
<?php
$response = file_get_contents("https://randomuser.me/api");
echo $response;
?>
```

Note how we get back from the API data only without any HTML.


```
{"results":[{"gender":"male","name":{"title":"Mr","first":"Claudio","last":"Hülsmann"},"locati
on":{"street":{"number":1424,"name":"Neue
Straße"},"city":"Stein","state":"Bayern","country":"Germany","postcode":56129,"coordinates":{"
latitude":"-47.9859","longitude":"-
104.6647"},"timezone":{"offset":"+9:30","description":"Adelaide,
Darwin"}},"email":"claudio.hulsmann@example.com","login":{"uuid":"26e2bd18-06bb-4c12-b23b-
a77876712db1","username":"bluepanda673","password":"bobbie","salt":"dASTXeiQ","md5":"e0e28315f
4d8a068a6dabc774bfef751","sha1":"6596d35f1e7a1cfda499d685262a4bcf57df47bc","sha256":"105f0ef3a
342dc95bac7a0bcb19cf0fc514f6b2fc3fa56c81aa3026ff06b662f"},"dob":{"date":"1969-12-
26T13:22:28.240Z","age":52},"registered":{"date":"2011-09-
12T08:31:37.055Z","age":11},"phone":"0444-0187005","cell":"0179-
1090187","id":{"name":"SVNR","value":"13 261269 H
064"},"picture":{"large":"https://randomuser.me/api/portraits/men/53.jpg","medium":"https://ra
ndomuser.me/api/portraits/med/men/53.jpg","thumbnail":"https://randomuser.me/api/portraits/thu
mb/men/53.jpg"},"nat":"DE"}],"info":{"seed":"62bdc63939d48ed0","results":1,"page":1,"version":
"1.4"}}
```

Decode API results reading JSON in PHP

APIs can return data in the XML, HTML or JSON data format with JSON (javascript notation) being the default. We can decode this data using php function of json_decode.

```
<?php
$response = file_get_contents("https://randomuser.me/api");
$data = json_decode($response);
var_dump($data);
?>
```

This returns a php object that contains all the data.


```
object(stdClass)#12 (2) { ["results"]=> array(1) { [0]=> object(stdClass)#1 (12) { ["gender"]=> string(6)
"female" ["name"]=> object(stdClass)#2 (3) { ["title"]=> string(3) "Mrs" ["first"]=> string(7) "Filippa"
["last"]=> string(9) "Mortensen" } ["location"]=> object(stdClass)#4 (7) { ["street"]=> object(stdClass)#3 (2)
{ ["number"]=> int(4008) ["name"]=> string(11) "Helsingevej" } ["city"]=> string(13) "Sundby/Erslev"
["state"]=> string(9) "Sjælland" ["country"]=> string(7) "Denmark" ["postcode"]=> int(69729) ["coordinates"]=>
object(stdClass)#5 (2) { ["latitude"]=> string(8) "-46.2715" ["longitude"]=> string(7) "67.3502" }
["timezone"]=> object(stdClass)#6 (2) { ["offset"]=> string(5) "-9:00" ["description"]=> string(6) "Alaska" }
} ["email"]=> string(29) "filippa.mortensen@example.com" ["login"]=> object(stdClass)#7 (7) { ["uuid"]=>
string(36) "488164db-ab05-407c-8257-e783201e49cb" ["username"]=> string(12) "heavyswan515" ["password"]=>
string(8) "earnhard" ["salt"]=> string(8) "TavbCUHj" ["md5"]=> string(32) "44dcdeaafb50f4cb352d4b13f2283e86"
["sha1"]=> string(40) "e81bd97fc857416d99db924fe6fc30ee50058aa1" ["sha256"]=> string(64)
"afe6ed63051e597b6c9e5e0367aca4e34fec915813b82d88809e43d4ca0b2633" } ["dob"]=> object(stdClass)#8 (2) {
["date"]=> string(24) "1974-03-09T07:37:39.663Z" ["age"]=> int(48) } ["registered"]=> object(stdClass)#9 (2) {
["date"]=> string(24) "2007-03-20T17:43:13.186Z" ["age"]=> int(15) } ["phone"]=> string(8) "01925765"
["cell"]=> string(8) "23909549" ["id"]=> object(stdClass)#10 (2) { ["name"]=> string(3) "CPR" ["value"]=>
string(11) "090374-9854" } ["picture"]=> object(stdClass)#11 (3) { ["large"]=> string(48)
"https://randomuser.me/api/portraits/women/14.jpg" ["medium"]=> string(52)
"https://randomuser.me/api/portraits/med/women/14.jpg" ["thumbnail"]=> string(54)
"https://randomuser.me/api/portraits/thumb/women/14.jpg" } ["nat"]=> string(2) "DK" } } ["info"]=>
object(stdClass)#13 (4) { ["seed"]=> string(16) "83fc42c1b256b43b" ["results"]=> int(1) ["page"]=> int(1)
["version"]=> string(3) "1.4" } }
```

We can convert the JSON data into associative arrays by passing in a second parameter of true.

```
<?php
$response = file_get_contents("https://randomuser.me/api");
$data = json_decode($response, true);
var_dump($data);
?>
```

Note how the output has changed from a PHP object to an array.

```
array(2) { ["results"]=> array(1) { [0]=> array(12) { ["gender"]=> string(6) "female" ["name"]=> array(3) {
["title"]=> string(3) "Mrs" ["first"]=> string(3) "Ava" ["last"]=> string(6) "Obrien" } ["location"]=>
array(7) { ["street"]=> array(2) { ["number"]=> int(321) ["name"]=> string(16) "Springfield Road" }
["city"]=> string(9) "Salisbury" ["state"]=> string(6) "Dorset" ["country"]=> string(14) "United Kingdom"
["postcode"]=> string(7) "Y21 7JS" ["coordinates"]=> array(2) { ["latitude"]=> string(7) "-5.7101"
["longitude"]=> string(8) "120.6678" } ["timezone"]=> array(2) { ["offset"]=> string(5) "-6:00"
["description"]=> string(39) "Central Time (US & Canada), Mexico City" } } ["email"]=> string(22)
"ava.obrien@example.com" ["login"]=> array(7) { ["uuid"]=> string(36) "2a630cb4-7626-4b89-b2b2-9d9b8b95ba22"
["username"]=> string(14) "bluemeercat858" ["password"]=> string(8) "pooppoop" ["salt"]=> string(8)
"xlBSFL8i" ["md5"]=> string(32) "ce24c8e45f004fde8f69bd293fc272ef" ["sha1"]=> string(40)
"1df90197e6f030c97bde51a253ddecc8a674dd38" ["sha256"]=> string(64)
"960806e84d2d947819e5d1c3a20bd95ad882eeaa2fac6ea4308c04507b04952b" } ["dob"]=> array(2) { ["date"]=>
string(24) "1977-06-22T07:05:17.707Z" ["age"]=> int(45) } ["registered"]=> array(2) { ["date"]=> string(24)
"2017-01-21T00:06:16.847Z" ["age"]=> int(5) } ["phone"]=> string(16) "0119189 563 6797" ["cell"]=> string(12)
"07752 572255" ["id"]=> array(2) { ["name"]=> string(4) "NINO" ["value"]=> string(13) "ZM 66 93 36 P" }
["picture"]=> array(3) { ["large"]=> string(48) "https://randomuser.me/api/portraits/women/48.jpg"
["medium"]=> string(52) "https://randomuser.me/api/portraits/med/women/48.jpg" ["thumbnail"]=> string(54)
"https://randomuser.me/api/portraits/thumb/women/48.jpg" } ["nat"]=> string(2) "GB" } } ["info"]=> array(4) {
["seed"]=> string(16) "8d7838ecbaf6cde6" ["results"]=> int(1) ["page"]=> int(1) ["version"]=> string(3) "1.4"
} }
```

We can access any of the data in the array and echo it out. In this case we are pulling the first name from the array.

```
<?php
$response = file_get_contents("https://randomuser.me/api");
$data = json_decode($response, true);
echo $data["results"][0]["name"]["first"], "\n";
?>
```

```
Siiri
```

Each time we refresh we get a different name because the PHP is doing an API call then converting the JSON data to an array and finally outputting the first name value.

Use API data in a web application

The below code block calls a free API that returns the age when we enter a given name.

```
<?php
    if ( ! empty($_GET["name"])) {
        $response = file_get_contents("https://api.agify.io?name={$_GET['name']}");
        $data = json_decode($response, true);
        $age = $data["age"];
    }
?>
<!DOCTYPE html>
<html>
    <head>
        <title>Example</title>
    </head>
    <body>
        <?php if (isset($age)): ?>
            Age: <?= $age ?>
        <?php endif; ?>

        <form>
            <label for="name">Name</label>
            <input name="name" id="name">
            <button>Guess age</button>
        </form>
    </body>
</html>
```

HTTP basics: requests, responses using CURL

Use cURL not file_get_contents to make API Request

PHP supports lib cURL and provides many functions for sending and receiving requests.
https://www.php.net/manual/en/book.curl.php

```
<?php
$curlHandle = curl_init();

curl_setopt($curlHandle, CURLOPT_URL, "https://randomuser.me/api");
curl_setopt($curlHandle, CURLOPT_RETURNTRANSFER, true);

$response = curl_exec($curlHandle);

curl_close($curlHandle);

echo $response, "\n";
?>
```

First we initialise the cURL connection.

We can use the curl_setopt function to define parameters such as the URL and that we want the returned results to be as a string using returntransfer set to true.

We execute the cURL request.

And close it afterwards to free up any system resources it has been using.


```
{"results":[{"gender":"female","name":{"title":"Madame","first":"Ornella","last":"Thomas"},"location":{"stree
t":{"number":6935,"name":"Rue des Cuirassiers"},"city":"La
Sarraz","state":"Schwyz","country":"Switzerland","postcode":9679,"coordinates":{"latitude":"-
58.5371","longitude":"117.6043"},"timezone":{"offset":"+6:00","description":"Almaty, Dhaka,
Colombo"}},"email":"ornella.thomas@example.com","login":{"uuid":"cf6cbb79-a944-46e9-b93d-
a9a0d8fb7f77","username":"ticklishmouse616","password":"choke","salt":"Kqpy4JDL","md5":"99c92a20dbf18e6a3beb5
239aa3462e4","sha1":"a624f9c255dec05b056911e126e51e2d1690f2fc","sha256":"20f699cdd3b6451601bfcbae9be355e5059f
87c1c75bdba819a542768dc1ac15"},"dob":{"date":"1982-08-01T20:08:00.005Z","age":40},"registered":{"date":"2002-
03-31T01:09:38.201Z","age":20},"phone":"078 494 41 10","cell":"075 969 69
51","id":{"name":"AVS","value":"756.3929.1619.19"},"picture":{"large":"https://randomuser.me/api/portraits/wo
men/84.jpg","medium":"https://randomuser.me/api/portraits/med/women/84.jpg","thumbnail":"https://randomuser.m
e/api/portraits/thumb/women/84.jpg"},"nat":"CH"}],"info":{"seed":"9a05a5fa5d57bd7d","results":1,"page":1,"ver
sion":"1.4"}}
```

```php
<?php
$curlHandle = curl_init();

// curl_setopt($curlHandle, CURLOPT_URL, "https://randomuser.me/api");
// curl_setopt($curlHandle, CURLOPT_RETURNTRANSFER, true);

curl_setopt_array($curlHandle, [
    CURLOPT_URL => "https://randomuser.me/api",
    CURLOPT_RETURNTRANSFER => true
]);

$response = curl_exec($curlHandle);

curl_close($curlHandle);

echo $response, "\n";
?>
```

When using multiple options with the cURL, we can set the options as an ARRAY.

This options array runs as before requestion data from the API


```
{"results":[{"gender":"female","name":{"title":"Madame","first":"Ornella","last":"Thomas"},"location":{"stree
t":{"number":6935,"name":"Rue des Cuirassiers"},"city":"La
Sarraz","state":"Schwyz","country":"Switzerland","postcode":9679,"coordinates":{"latitude":"-
58.5371","longitude":"117.6043"},"timezone":{"offset":"+6:00","description":"Almaty, Dhaka,
Colombo"}},"email":"ornella.thomas@example.com","login":{"uuid":"cf6cbb79-a944-46e9-b93d-
a9a0d8fb7f77","username":"ticklishmouse616","password":"choke","salt":"Kqpy4JDL","md5":"99c92a20dbf18e6a3beb5
239aa3462e4","sha1":"a624f9c255dec05b056911e126e51e2d1690f2fc","sha256":"20f699cdd3b6451601bfcbae9be355e5059f
87c1c75bdba819a542768dc1ac15"},"dob":{"date":"1982-08-01T20:08:00.005Z","age":40},"registered":{"date":"2002-
03-31T01:09:38.201Z","age":20},"phone":"078 494 41 10","cell":"075 969 69
51","id":{"name":"AVS","value":"756.3929.1619.19"},"picture":{"large":"https://randomuser.me/api/portraits/wo
men/84.jpg","medium":"https://randomuser.me/api/portraits/med/women/84.jpg","thumbnail":"https://randomuser.m
e/api/portraits/thumb/women/84.jpg"},"nat":"CH"}],"info":{"seed":"9a05a5fa5d57bd7d","results":1,"page":1,"ver
sion":"1.4"}}
```

Response codes: Get the HTTP status code

```php
<?php
$curlHandle = curl_init();

curl_setopt_array($curlHandle, [
    CURLOPT_URL => "https://randomuser.me/api",
    CURLOPT_RETURNTRANSFER => true
]);

$response = curl_exec($curlHandle);
$statusCode = curl_getinfo($curlHandle, CURLINFO_HTTP_CODE);
curl_close($curlHandle);

echo "<p>cURL status code: $statusCode</p><br>";
echo $response, "\n";
?>
```

We can get the cURL status code from the response using the PHP function of **curl_getinfo** and pass int the curl handle and CURLINFO_HTTP_CODE

```
cURL status code: 200

{"results":[{"gender":"male","name":{"title":"Mr","first":"Sivan","last":"Herder"},"location":{"street":{"number":1726,"
name":"Eerste Tuinsingel"},"city":"Westernieland","state":"Gelderland","country":"Netherlands","postcode":"4518
KT","coordinates":{"latitude":"37.2932","longitude":"-128.4955"},"timezone":{"offset":"-6:00","description":"Central
Time (US & Canada), Mexico City"}},"email":"sivan.herder@example.com","login":{"uuid":"d4327e40-868f-42cd-af5c-
5a80c60aafa4","username":"greenrabbit434","password":"ziggy","salt":"IyF0lW1l","md5":"0e62eceecea39dabb007165aee96f79b",
"sha1":"8f8578367f73bdbe6638bcc99e3a788955417485","sha256":"4c35e681b066310a22ebbdac5100fa8d24839cbb7343f6decb0435ac2d27
3fd1"},"dob":{"date":"1982-08-11T00:56:02.239Z","age":40},"registered":{"date":"2013-05-
10T21:38:41.358Z","age":9},"phone":"(0714) 925885","cell":"(06)
72259109","id":{"name":"BSN","value":"74459294"},"picture":{"large":"https://randomuser.me/api/portraits/men/4.jpg","med
ium":"https://randomuser.me/api/portraits/med/men/4.jpg","thumbnail":"https://randomuser.me/api/portraits/thumb/men/4.jp
g"},"nat":"NL"}],"info":{"seed":"27911c41754ef7d8","results":1,"page":1,"version":"1.4"}}
```

```php
<?php
$curlHandle = curl_init();

curl_setopt_array($curlHandle, [
    CURLOPT_URL =>
     "http://api.openweathermap.org/data/2.5/forecast?q=Paris&appid=a44c2366eada4d91b124e9a0
     1d82f381",
    CURLOPT_RETURNTRANSFER => true
]);

$response = curl_exec($curlHandle);

$statusCode = curl_getinfo($curlHandle, CURLINFO_HTTP_CODE);

curl_close($curlHandle);

echo "<p>cURL status code: $statusCode</p><br>";
echo $response, "\n";
?>
```

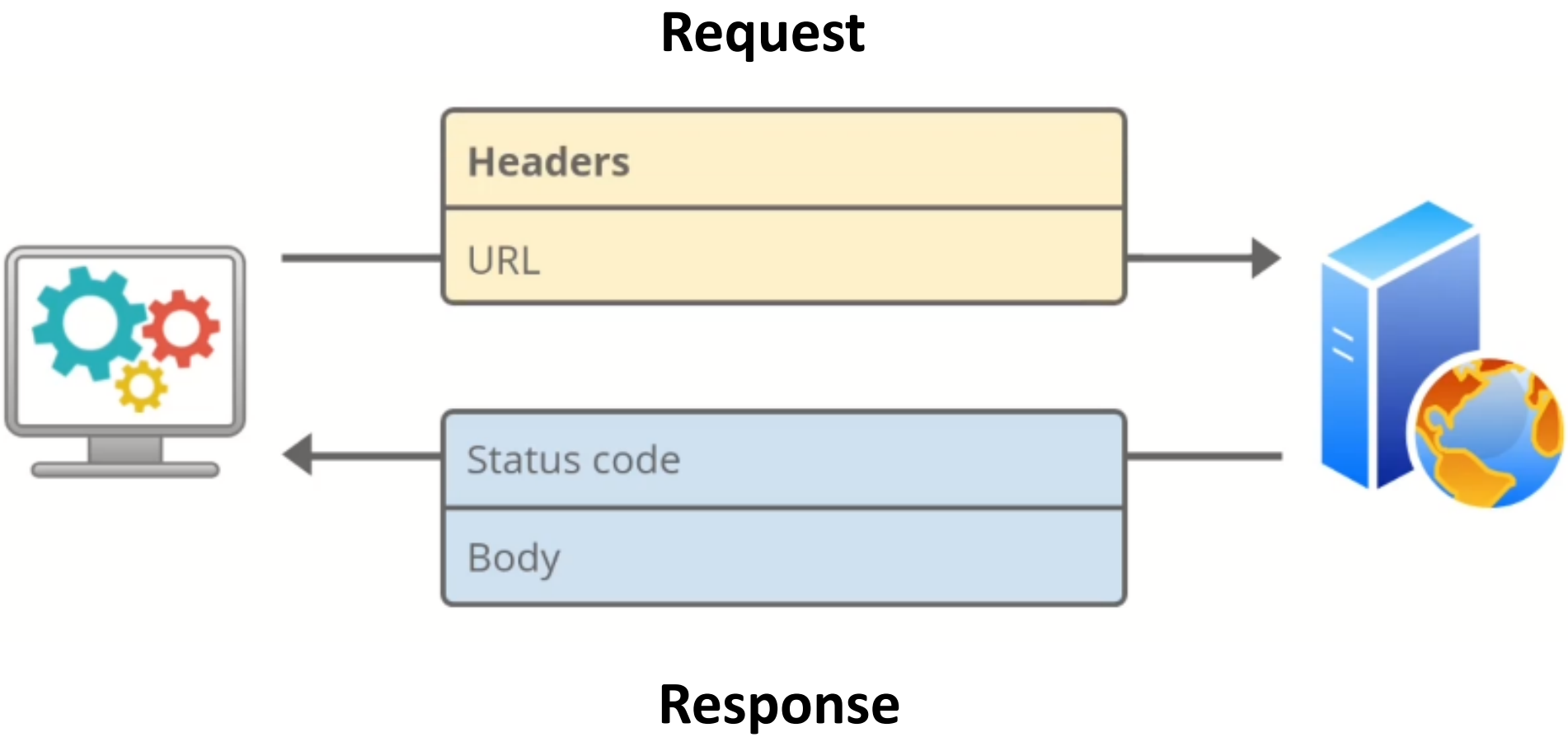
Another real world example is a free weather API

```
cURL status code: 200

{"cod":"200","message":0,"cnt":40,"list":[{"dt":1669431600,"main":{"temp":279.9,"feels_like":278.93,"temp_min
":279.9,"temp_max":280.82,"pressure":1028,"sea_level":1028,"grnd_level":1024,"humidity":91,"temp_kf":-
0.92},"weather":[{"id":802,"main":"Clouds","description":"scattered
clouds","icon":"03n"}],"clouds":{"all":42},"wind":{"speed":1.64,"deg":217,"gust":2.59},"visibility":10000,"po
p":0,"sys":{"pod":"n"},"dt_txt":"2022-11-26 03:00:00"},{"dt":1669442400,"main":

. . .
```

Request headers: Add metadata about the request

In addition to the URL we specify we can also add headers to the request such as details about the client, or request a response in a specific language or format or give an API key.

https://en.wikipedia.org/wiki/List_of_HTTP_header_fields#Request_fields



```php
<?php
$curlHandle = curl_init();

curl_setopt_array($curlHandle, [
    CURLOPT_URL => "https://api.unsplash.com/photos/random",
    CURLOPT_RETURNTRANSFER => true
]);

$response = curl_exec($curlHandle);

$statusCode = curl_getinfo($curlHandle, CURLINFO_HTTP_CODE);

curl_close($curlHandle);

echo "<p>cURL status code: $statusCode</p><br>";
echo $response, "\n";
?>
```

```
cURL status code: 401

{"errors":["OAuth error: The access token is invalid"]}
```

In this example we are using a different API that will return a random photo. This API requires an API key that we send in the header unlike attached to the query string in the previous example.

At the moment it fails with a 401 response because the API key to be sent in the header is not defined.

```php
<?php
$curlHandle = curl_init();

$headers = [
    "Authorization: Client-ID _ulE3RpFu6ontC309tETZNqE_fzwlLcwQL9o3m-gHYI"
];

curl_setopt_array($curlHandle, [
    CURLOPT_URL => "https://api.unsplash.com/photos/random",
    CURLOPT_RETURNTRANSFER => true,
    CURLOPT_HTTPHEADER => $headers
]);

$response = curl_exec($curlHandle);

$statusCode = curl_getinfo($curlHandle, CURLINFO_HTTP_CODE);

curl_close($curlHandle);

echo "<p>cURL status code: $statusCode</p><br>";
echo $response, "\n";
?>
```

Now we create a **headers array** which contains the api key.

And add the headers array to the **curl_set_opt array**

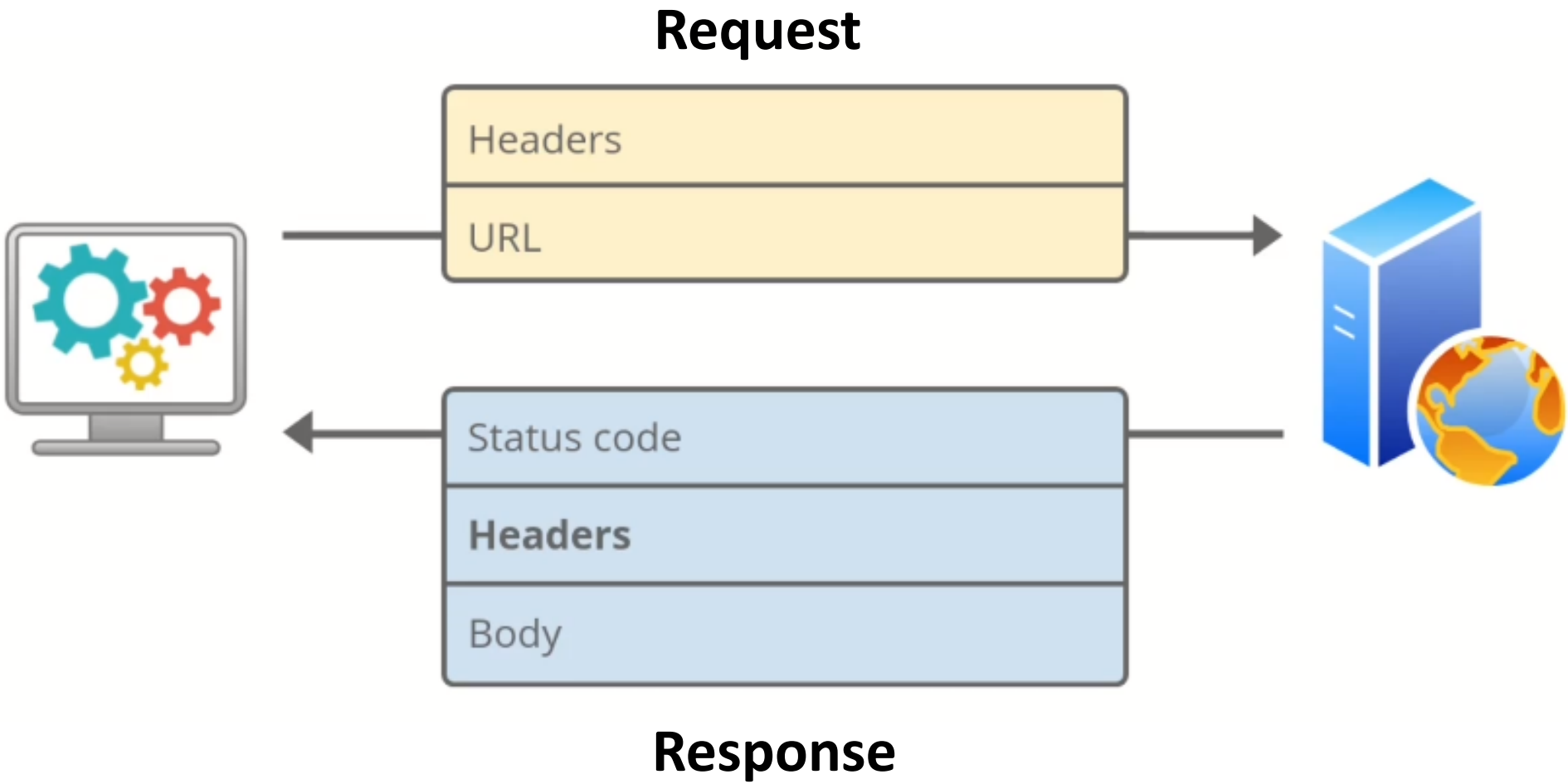
Now the response contains an object that contains an image path and associated image metadata.

```
cURL status code: 200

{"id":"cjS7ipT7tX0","created_at":"2022-10-31T18:48:17Z","updated_at":"2022-11-
26T22:32:48Z","promoted_at":"2022-11-
02T02:48:01Z","width":3456,"height":5184,"color":"#f3f3f3","blur_hash":"LNQJcc8_%Mxu_NW.RPoL?v%gM{M{","descri
ption":"If you like my work and want to support my passion for photography - any amount of donation is
. . .
```

Response headers: read meta data about the reponse

Common response headers include details about its length, its language and its type such as HTML, XML. JSNO etc.

https://en.wikipedia.org/wiki/List_of_HTTP_header_fields#Response_fields



```
curl_setopt_array($curlHandle, [
    CURLOPT_URL => "https://api.unsplash.com/photos/random",
    CURLOPT_RETURNTRANSFER => true,
    CURLOPT_HTTPHEADER => $headers,
    CURLOPT_HEADER => true
]);
```

```
cURL status code: 200

HTTP/2 200 server: Cowboy
content-type: application/json
access-control-allow-origin:
access-control-request-method:
* access-control-allow-headers:
* access-control-expose-headers: Link,X-Total,X-Per-Page,X-RateLimit-Limit,X-
RateLimit-Remaining
cache-control: private,max-age=0,stale-if-error=3600,stale-while-revalidate=0
x-unsplash-version: v1
content-language: en etag: W/"823482ba161c11c5194a6b41c2903427"
x-ratelimit-limit: 50 x-ratelimit-remaining: 47 x-request-id: 2295e680-4dfb-
40f8-ad72-821a57dd3b19 x-runtime: 0.029128
strict-transport-security: max-age=31536000;
includeSubDomains via: 1.1 vegur, 1.1 varnish,
1.1 varnish accept-ranges: bytes
date: Sun, 27 Nov 2022 00:30:51 GMT
x-served-by: cache-iad-kjyo7100023-IAD, cache-mad22083-MAD
x-cache: MISS, MISS x-cache-hits: 0, 0
x-timer: S1669509052.516197,VS0,VE137
vary: Accept-Encoding, Origin,Authorization,Accept-Language,Accept
content-length: 3943
{"id":"-LhLmNx9fTI","created_at":"2022-11-07T16:13:55Z","updated_at":"2022-11-
26T12:38:46Z","promoted_at":"2022-11-
08T13:40:02Z","width":4062,"height":6093,"color":"#404040. . .
```

To see the response header we can add the **CURLOPT_HEADER set to true**.

The response header shows first before the response body

content-type is json

Content-length in bytes is 3943

The header has some custom values denoted by an X. in this case the x-rate-limit is set to 50 images an hour and we have 47 remaining.

```php
<?php
$curlHandle = curl_init();

$headers = [
    "Authorization: Client-ID _ulE3RpFu6ontC309tETZNqE_fzwlLcwQL9o3m-gHYI"
];

curl_setopt_array($curlHandle, [
    CURLOPT_URL => "https://api.unsplash.com/photos/random",
    CURLOPT_RETURNTRANSFER => true,
    CURLOPT_HTTPHEADER => $headers,
    CURLOPT_HEADER => true
]);

$response = curl_exec($curlHandle);

$statusCode = curl_getinfo($curlHandle, CURLINFO_HTTP_CODE);
$contentType = curl_getinfo($curlHandle, CURLINFO_CONTENT_TYPE);
$contentLength = curl_getinfo($curlHandle, CURLINFO_CONTENT_LENGTH_DOWNLOAD);

curl_close($curlHandle);

echo "<p>cURL status code: $statusCode ::::: cURL content type: $contentType ::::: cURL
content length: $contentLength</p><br>";
echo $response, "\n";
?>
```

We can also get the content-type and content_length from the **curl_getinfo** function.

```
cURL status code: 200 ::::: cURL content type: application/json ::::: cURL content length: 3522

HTTP/2 200 server: Cowboy content-type: application/json access-control-allow-origin: * access-control-
request-method: * access-control-allow-headers: * access-c . . .
```

Get all individual response headers in an array

```
cURL status code: 200

Array (
[server] => Cowboy
[content-type] => application/json
[access-control-allow-origin] => *
[access-control-request-method] => *
[access-control-allow-headers] => *
[access-control-expose-headers] => Link,X-Total,X-Per-Page,X-RateLimit-Limit,X-RateLimit-Remaining
[cache-control] => private,max-age=0,stale-if-error=3600,stale-while-revalidate=0
[x-unsplash-version] => v1
[content-language] => en
[etag] => W/"3b299ea1219ec9b9d5a7355631932586"
[x-ratelimit-limit] => 50
[x-ratelimit-remaining] => 46
[x-request-id] => 0e60bd90-d18e-4129-a3d8-2ed97eb13f89
[x-runtime] => 0.033825
[strict-transport-security] => max-age=31536000; includeSubDomains
[via] => 1.1 vegur, 1.1 varnish, 1.1 varnish
[accept-ranges] => bytes
[date] => Sun, 27 Nov 2022 01:18:12 GMT
[x-served-by] => cache-iad-kiad7000128-IAD, cache-mad22030-MAD
[x-cache] => MISS, MISS
[x-cache-hits] => 0, 0
[x-timer] => S1669511893.558028,VS0,VE128
[vary] => Accept-Encoding, Origin,Authorization,Accept-Language,Accept
[content-length] => 18805
)
{"id":"7wntHja88xw","created_at":"2022-11-23T09:17:42Z","updated_at":"2022-11-26T12:39:07Z","pro. . .
```

We cannot get all the response headers from curl_get_info but we can put them into a userble format such as an array. See following slide for code.

```php
<?php
$curlHandle = curl_init();

$requestHeaders = [
    "Authorization: Client-ID _ulE3RpFu6ontC309tETZNqE_fzwlLcwQL9o3m-gHYI"
];

$responseHeaders = []; 1

$headerCallback = function($curlHandle, $header) use (&$responseHeaders) {
    $len = strlen($header);
    $parts = explode(":", $header, 2); 3
    if (count($parts) < 2) {
        return $len;
    }
    $responseHeaders[$parts[0]] = trim($parts[1]); 4
    return $len;
};

curl_setopt_array($curlHandle, [
    CURLOPT_URL => "https://api.unsplash.com/photos/random",
    CURLOPT_RETURNTRANSFER => true,
    CURLOPT_HTTPHEADER => $requestHeaders,
    CURLOPT_HEADERFUNCTION => $headerCallback 5
]);

$response = curl_exec($curlHandle);

$statusCode = curl_getinfo($curlHandle, CURLINFO_HTTP_CODE);

curl_close($curlHandle);

echo "<p>cURL status code: $statusCode</p><br>";
print_r($responseHeaders); 6
echo $response, "\n";
?>
```

1) Initialise an array to store each response header.

2) Create a closure which will be a callback function that will be called for each response header. We have use the response header array in the call back function passing it by reference using ampersand.

3) To format the array so that the header name is the key of the array and the header value is the array value we need to split the header into two parts by the colon. If it does not have two parts it is not a valid header so we can just return length.

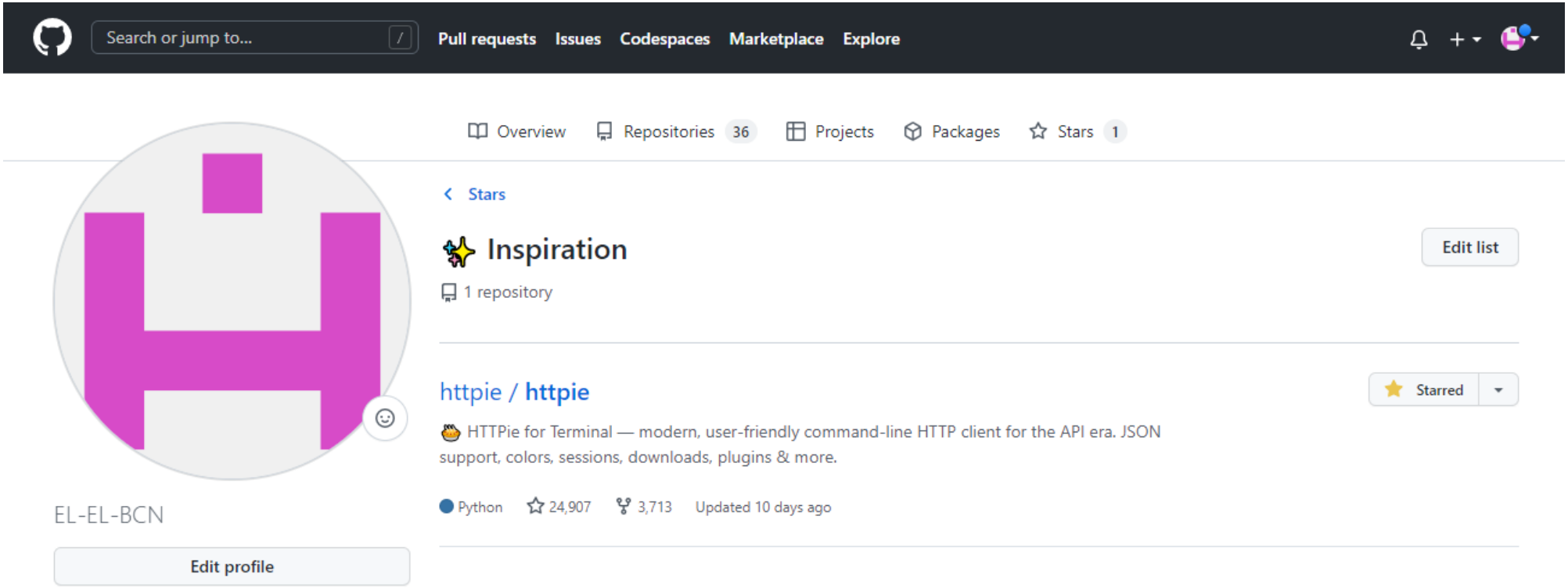
4) We can now add the header name and header value to the array triming any whitespace from the value

5) To use this function we set the curlopt headerfunction to the callback function.

6) print_r to see the response headers array on screen

Use an API that requres a specific request header

In this example we are using github repositories to see if a specific repo is starred by my personal user credentials with an API call from PHP and setting custom heading parameters as per the github documentation. https://docs.github.com/en/rest/activity/starring



For this exercise to work I have manually starred the repo using github from the browser.

```php
<?php
$curlHandle = curl_init();

$requestHeaders = [
    "Accept: application/vnd.github+json",
    "Authorization: Bearer ghp_xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
];

curl_setopt_array($curlHandle, [
    CURLOPT_URL => "https://api.github.com/user/starred/httpie/httpie",
    CURLOPT_RETURNTRANSFER => true,
    CURLOPT_HTTPHEADER => $requestHeaders,
    CURLOPT_USERAGENT => "el_el"
]);

$response = curl_exec($curlHandle);

$statusCode = curl_getinfo($curlHandle, CURLINFO_HTTP_CODE);

curl_close($curlHandle);

echo "<p>cURL status code: $statusCode</p><br>";
echo $response, "\n";
?>
```

My personal gitgub **PAT token**

My personal gitgub **username**

According to the github documentation if this repository is starred by me then it shoud return a 204 code.

```
cURL status code: 204
```

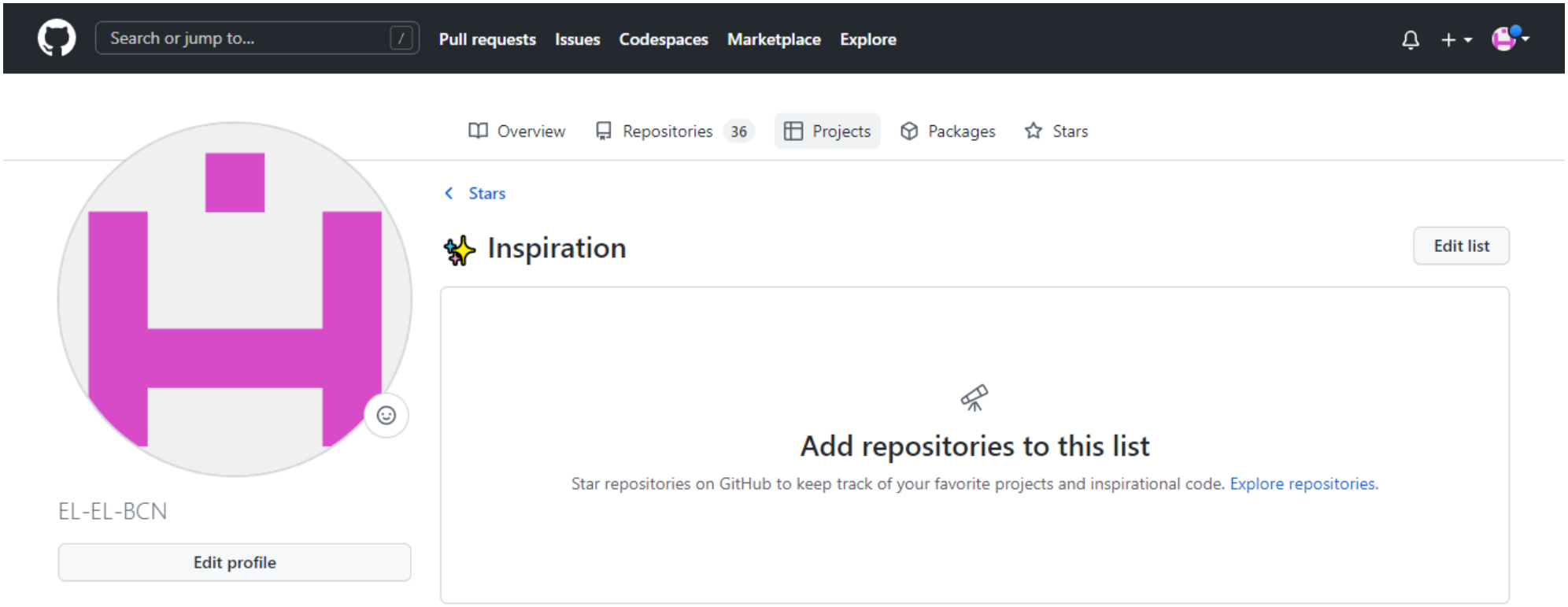
Request Method: change the method to get a differnet result with the same URL

https://docs.github.com/en/rest/activity/starring#unstar-a-repository-for-the-authenticated-user

```php
<?php
$curlHandle = curl_init();

$requestHeaders = [
    "Accept: application/vnd.github+json",
    "Authorization: Bearer ghp_xxxxxxxxxxxxxxxxxx"
];

curl_setopt_array($curlHandle, [
    CURLOPT_URL => "https://api.github.com/user/starred/httpie/httpie",
    CURLOPT_RETURNTRANSFER => true,
    CURLOPT_HTTPHEADER => $requestHeaders,
    CURLOPT_USERAGENT => "el_el",
    CURLOPT_CUSTOMREQUEST => "DELETE"
]);

$response = curl_exec($curlHandle);

$statusCode = curl_getinfo($curlHandle, CURLINFO_HTTP_CODE);

curl_close($curlHandle);

echo "<p>cURL status code: $statusCode</p><br>";
echo $response, "\n";
?>
```

The default for cURL is GET but we can use **DELETE** to modify. For example unstar my username from this repo.

According to the github documentation if I am unstarred from this repo then the API will return a 204 code. `cURL status code: 204` If I check my github account I can now see that this Repo is not in my starred list.

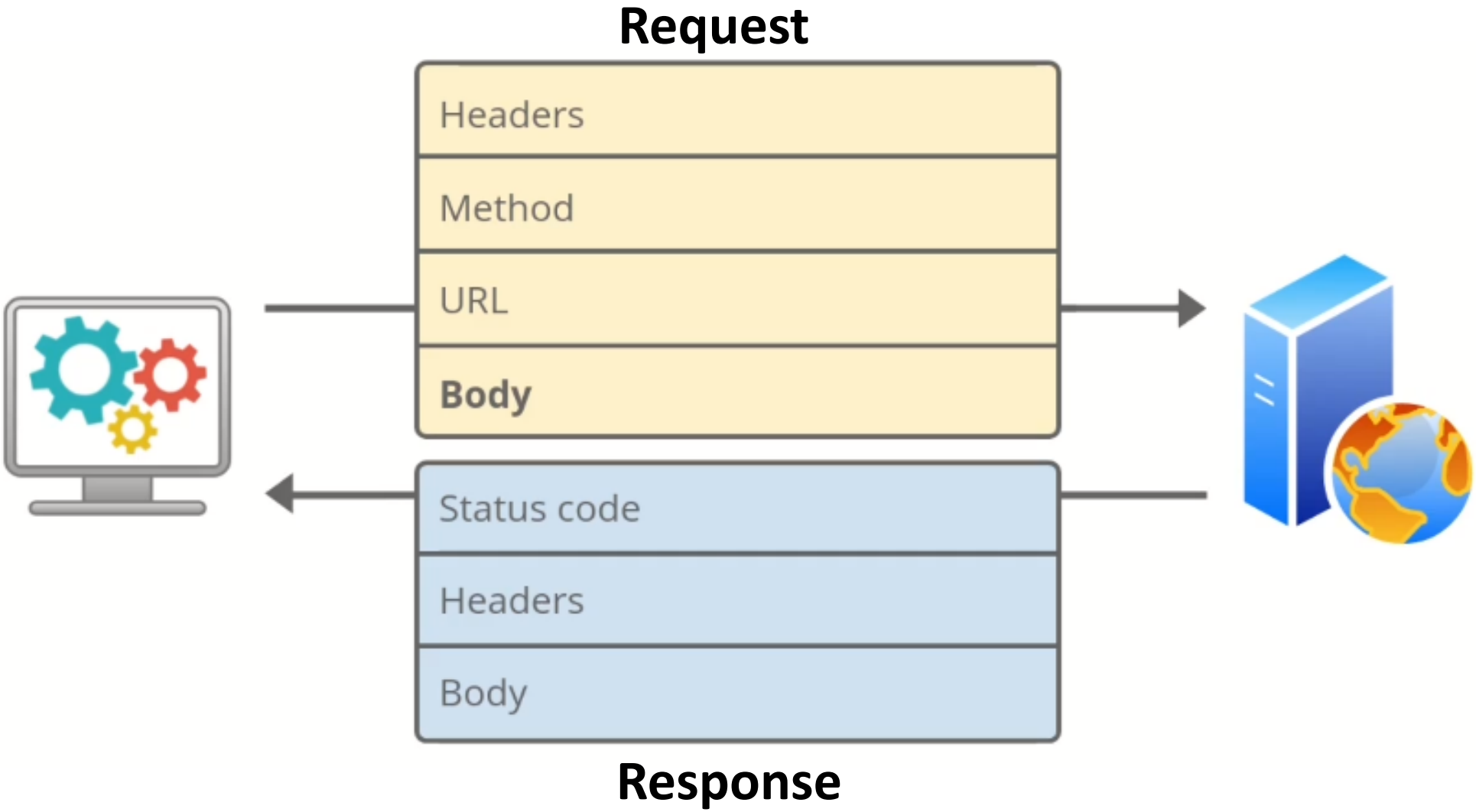


The CURLOPT_CUSTOMREQUEST => "PUT" can be changed to put to star this repository via API call.

Request body: add payload to send data along with the request

In addition to the response body we can also send a body with a request.
In this example we will create a reop using an cURL API call.

https://docs.github.com/en/rest/repos/repos



```php
<?php
$curlHandle = curl_init();

$requestHeaders = [
    "Accept: application/vnd.github+json",
    "Authorization: Bearer ghp_xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx"
];

$requestPayload = json_encode([
    "name" => "Created from API",
    "description" => "An example API-created repo"
]);

curl_setopt_array($curlHandle, [
    CURLOPT_URL => "https://api.github.com/user/repos",
    CURLOPT_RETURNTRANSFER => true,
    CURLOPT_HTTPHEADER => $requestHeaders,
    CURLOPT_USERAGENT => "el_el",
    CURLOPT_CUSTOMREQUEST => "POST",
    CURLOPT_POSTFIELDS => $requestPayload
]);

$response = curl_exec($curlHandle);

$statusCode = curl_getinfo($curlHandle, CURLINFO_HTTP_CODE);

curl_close($curlHandle);

echo "<p>cURL status code: $statusCode</p><br>";
echo $response, "\n";
?>
```

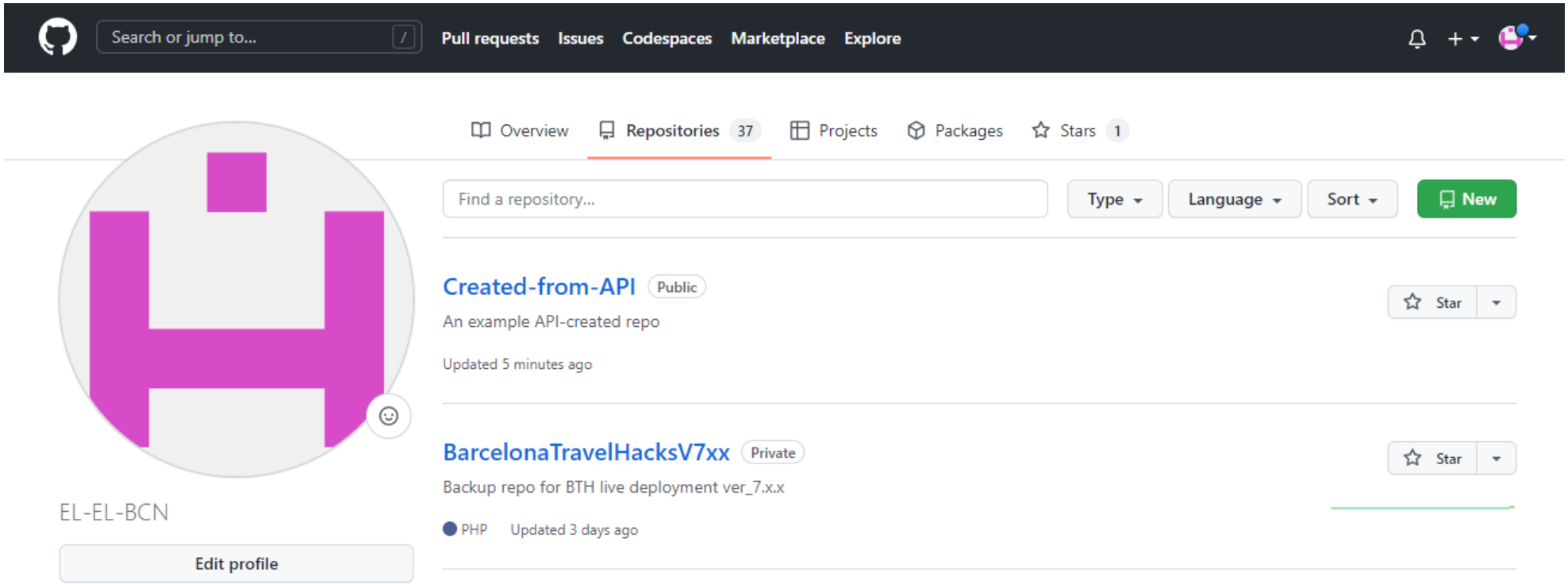
We create a new array for the request payload which has json encoded data of repo name and repo description in it.

The API call uses POST mthod to post data to the github server. We specify a post field set to the array of payload data.

```
cURL status code: 201

{"id":571068272,"node_id":"R_kgDOIgnPcA","name":"Created-from-
API","full_name":"EL-EL-BCN/Created-from-
API","private":false,"owner":{"login":"EL-EL-
BCN","id":101018935,"node_id":"U_kgDOBgVtNw","avatar_url":"https://avata
rs.githubusercontent.com/u/101018935?v=4","gravatar_id":"","url":"https:
//api.github.com/users/EL-EL-BCN","html_url":"https://github.com/EL-EL-
BCN","followers_url":"https://api.github.com/users/EL-EL-
BCN/followers","following_url":"https://api.github.com/users/EL-EL-
BCN/following{/other_user}","gists_url":"https://api.github.com/users/EL
-EL-BCN/gists{/gist_id}","starred_url":"https://api.github.com/users/
...
```

The API response call is code 201 as per the documentation and also contains an object of the new REPOs parameters.

We can also see the new repo in the web portal

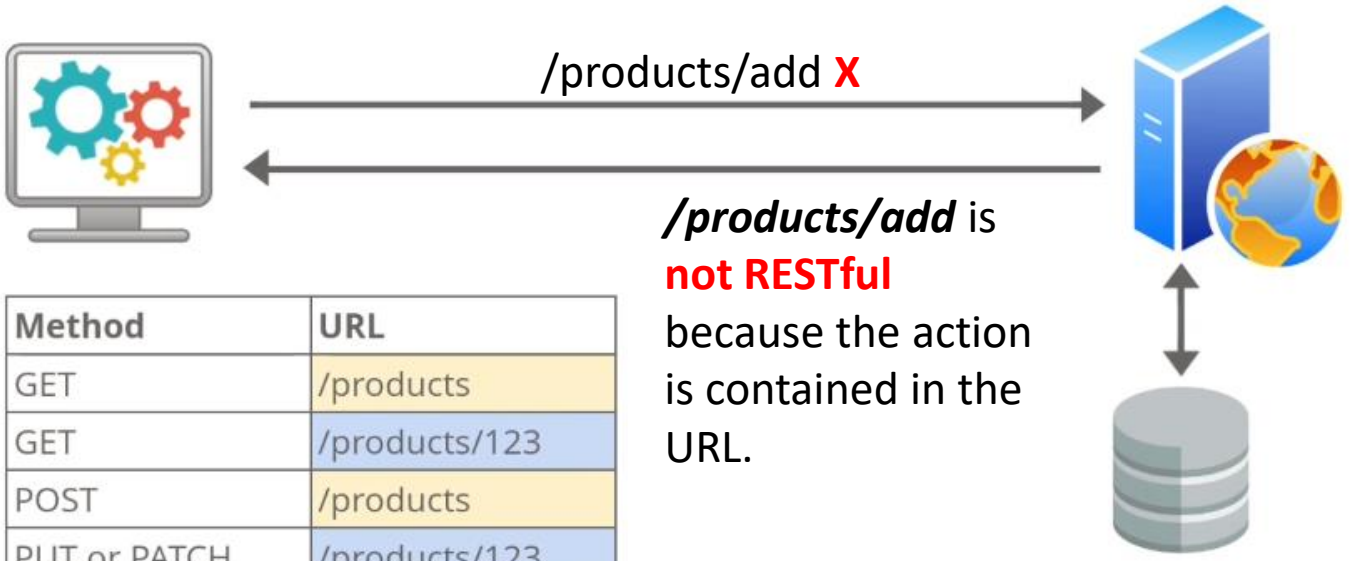


REST and RESTful APIs: Using them from PHP

REST and RESTful APIs: What are they?

REpresentational **S**tate **T**ransfer: is a software architectural style that describes a uniform interface between physically separate components, often across the Internet in a client-server architecture.
https://en.wikipedia.org/wiki/Representational_state_transfer
https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

Most APIs are RESTful in that they follow a set of rules known as representational state transfer or REST. REST is a set of rules about how to structure an application. The basic idea of REST is to treat objects on the server as resources that can be created, updated or destroyed, for example rows in a database table. Unlike a regular web application, these actions are carried out using specific URLs.



Method	URL
GET	/products
GET	/products/123
POST	/products
PUT or PATCH	/products/123
DELETE	/products/123

The example URLs in this diagram are RESTful because they have the same URL and the action is defined by the method. I.e. the same URL to get data, put data, post data and delete data.

POST method will create a new product. PUT or PATCH will update the product, DELETE will erase it.

Note that to do all this we only need two URLs. One for all products, and one for the product.

Access a RESTful API in PHP using cURL

```php
<?php
$curlHandle = curl_init();

curl_setopt_array($curlHandle, [
    CURLOPT_URL => "https://api.github.com/gists",
    CURLOPT_RETURNTRANSFER => true,
    CURLOPT_USERAGENT => "el_el",
]);

$response = curl_exec($curlHandle);

$statusCode = curl_getinfo($curlHandle, CURLINFO_HTTP_CODE);

curl_close($curlHandle);

echo "<p>cURL status code: $statusCode</p><br>";

$data = json_decode($response, true);

foreach ($data as $gist) {
    echo "<span>".$gist['id']." - ". $gist['description']."</span><br>";
}
?>
```

The Json response is decoded into an associative array and we loop around it using foreach to get a list of gist IDs and their descriptions.

Note that we only get 30 results because this is the default but we can change it with the query string.

cURL status code: 200

e9a70cff0c2ac3cfbf66046ab7c0fba1 -
326b4c2ceb4fe0cf1bd03c256b81708f -
7e5286602eb864e23dccfd0ee340f48b -
5770ef75a14596a6fa0bbf541cae41eb -
a8d185bb234722b4950c9a8431d0e069 -
7754f1bfe686175c1583702a7cb1b264 -
9ef0bfdd00863a85fc7a23e02edef42f -
256dbde1c90b2572deaed657e0bf6767 -
de644843ae2a570903de2ff126404258 - daruhanBloggerCustom.gist
13cbe22dda9309b64c1cc5e45b8cba92 -
69c3ffa37b781b51ccfabf129c2759a1 - Rimworld output log published using HugsLib
6eb55599652283d0d896cab0d4a912a7 - Andorid - Kotlin - RecyclerView View binding adapter with memory leak fix
805ec58d3702710699bdd083373f7740 -
e02a0a1e71d22b67a8e6267d109893df - Regex tutorial on how to match a URL
e91b9afbd5ad6048aa7205bbda9af7ad -
e42c0443bf77d34887d99f4d00517135 -
90a1717b39632eb0c27e55b39ca2faf9 - ☁ 一些免费的云资源
574286350c60bdbcf4a0dd33dae83da2 -
594028c89c394cc1a13b23468f57d572 -
843e93774d07940ef12cc20f02bda9bd - Swtich case code generator for enum
a3fe9825581937ae1f7ba0e815e7a5c3 -
96819050016d351917f267e8bd39cc11 -
de43f11b967821cf810201964adac1eb -
f9c66300360de1b5c0f3b62223bf0f6d - Gentoo - /usr/lib64/ruby/2.7.0/rubygems.rb:16:in `require': cannot load such file -- rubygems/compatibility (LoadError)
acce505caceae2ba02cd719a77048f73 -
162e9f07cc0aa49aa18a260c282aac72 - Angular Data Table Component
910679689a36ce1ddd6dc90e9472b9c4 -
7546b1ed6dc596ce0050fba9fe0fae20 -
7466bd9bc7fda031007040fb19bf0131 - Patch for https://aur.archlinux.org/packages/ttf-koruri#comment-891415
cbe3f9e62424fe8e7fa3a442df2e95ac -

```php
<?php
$curlHandle = curl_init();

curl_setopt_array($curlHandle, [
    CURLOPT_URL => "https://api.github.com/gists/e02a0a1e71d22b67a8e6267d109893df",
    CURLOPT_RETURNTRANSFER => true,
    CURLOPT_USERAGENT => "el_el",
]);

$response = curl_exec($curlHandle);

$statusCode = curl_getinfo($curlHandle, CURLINFO_HTTP_CODE);

curl_close($curlHandle);

echo "<p>cURL status code: $statusCode</p><br>";

print_r($data);
?>
```

Note how the URL changes with the gist ID added to get data about a specific GIST.

```
Array ( [0] => Array ( [url] => https://api.github.com/gists/80d38caffb2068fac1a03b04b124aefd [forks_url] =>
https://api.github.com/gists/80d38caffb2068fac1a03b04b124aefd/forks [commits_url] =>
https://api.github.com/gists/80d38caffb2068fac1a03b04b124aefd/commits [id] => 80d38caffb2068fac1a03b04b124aefd
[node_id] => G_kwDOAWatx9oAIDgwZDM4Y2FmZmIyMDY4ZmFjMWEwM2IwNGIxMjRhZWZk [git_pull_url] =>
https://gist.github.com/80d38caffb2068fac1a03b04b124aefd.git [git_push_url] =>
https://gist.github.com/80d38caffb2068fac1a03b04b124aefd.git [html_url] =>
https://gist.github.com/80d38caffb2068fac1a03b04b124aefd [files] => Array ( [output_log.txt] => Array (
[filename] => output_log.txt [type] => text/plain [language] => Text [raw_url] =>
. . .
```

Use the Guzzle HTTP client for object orientated PHP code

I used the below tutorials to install Guzzle in my xampp environment and perform the composer update from within the guzzle folder.

https://github.com/guzzle/guzzle
https://www.phpflow.com/php/php-guzzle-http-client-example-with-php-7-laravel5/
https://www.coderchamp.com/update-php-in-xampp-and-composer/

```php
<?php

require "guzzle_test/vendor/autoload.php";

$client = new GuzzleHttp\Client;

$response = $client->request("GET", "https://api.github.com/user/repos", [
    "headers" => [
        "Authorization" => "token ghp_xxxxxxxxxxxxxxxxxxxx",
        "User-Agent" => "el_el"
    ]
]);

echo $response->getStatusCode(), "\n";

echo $response->getHeader("content-type")[0], "\n";

echo substr($response->getBody(), 0, 200), "...\n";
```

We need to install the **autoload.php** file from guzzle folder so that we have all the guzzle classes.

Create a **guzzle client object**

To use guzzle we specify the **method** then the **URL**

This method returns a response object. We can use specific Guzzle methods & classes.

getStatusCode() **getHeader()** **getBody()**

In this case we are getting the first 200 characters of the first item in the body of the response.

```
200 application/json; charset=utf-8 [{"id":466599203,"node_id":"R_kgDOG8-9Iw","name":"-my-first-git-hub-repository","full_name":"EL-EL-BCN/-my-first-git-hub-repository","private":false,"owner":{"login":"EL-EL-BCN","id":101018935,"node_i...
```

In this case we are seeing the first 200 characters of the array for the first item from response which is the first github repo that I ever created.

Guzzle has a comprehensive documentation guide https://docs.guzzlephp.org/en/stable/

Use an SDK: Compare the stripe API to its SDK

Curl and guzzle use HTTP to access the API. The next alternative is the use an SDK, software development kit. An SDK is a pre-written package set or library to include in our code. Not all APIs will have an SDK and there is no standard structure for deploying an SDK so the provider documentation must be consulted.

Examples of SDKs are: https://aws.amazon.com/sdk-for-php/ or https://stripe.com/en-gb-es

In this example we will use the stripe online web payment platform as an example of API and SDK https://stripe.com/docs/api/customers?lang=php

For testing purposes I have an account with stripe and from here I can get my api_key https://dashboard.stripe.com/test/apikeys

```php
<?php

$api_key = "sk_test_xxxxxxxxxxxxxxxxxxxx";

$data = [
    "name" => "Bob",
    "email" => "bob@example.com"
];

$curlhandler = curl_init();

curl_setopt_array($curlhandler, [
    CURLOPT_URL => 'https://api.stripe.com/v1/customers',
    CURLOPT_RETURNTRANSFER => true,
    CURLOPT_USERPWD => $api_key,
    CURLOPT_POSTFIELDS => http_build_query($data)
]);

$response = curl_exec($curlhandler);

curl_close($curlhandler);

echo $response;
?>
```

To use cURL we start with the standard code and define our API key.

We can have an array of data we want to send via the API in the body using post.

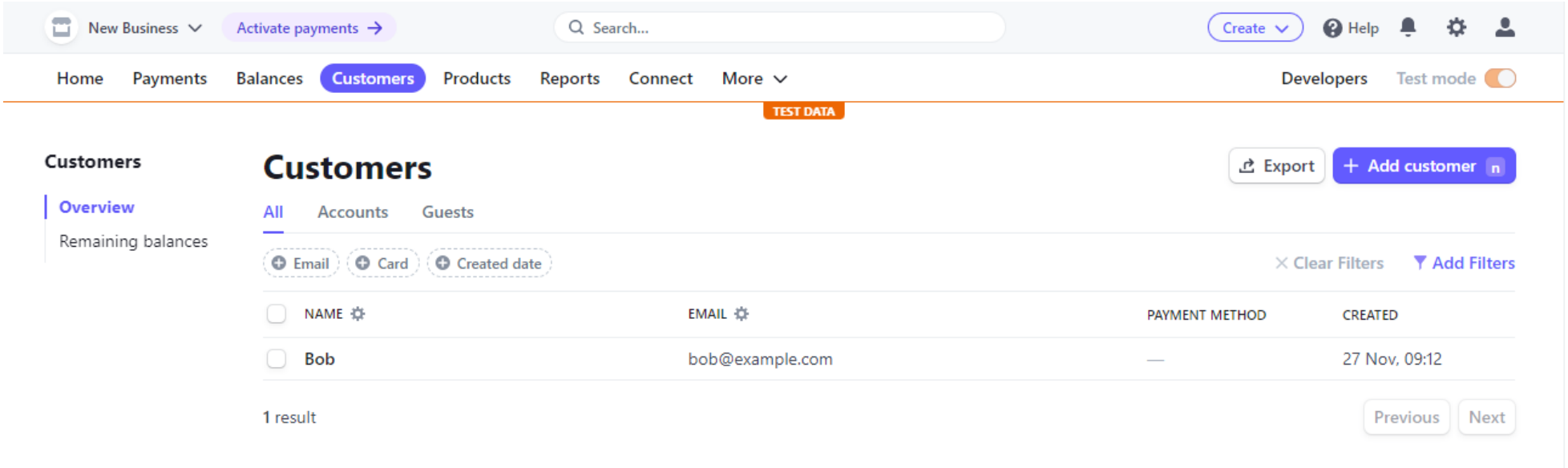
The Stripe platform requires basic HTTP authentication which we can do using the **CURLOPT_USERPWD** function.

The Stripe platform requires the data in the URL as a query string so we can use the PHP function of **http_build_query**, passing in the data.

And echo out the **response object**.


```
{ "id": "cus_MsQpkqgvkb6kuF", "object": "customer", "address": null, "balance": 0, "created": 1669536728,
"currency": null, "default_source": null, "delinquent": false, "description": null, "discount": null, "email":
"bob@example.com", "invoice_prefix": "13DA4769", "invoice_settings": { "custom_fields": null,
"default_payment_method": null, "footer": null, "rendering_options": null }, "livemode": false, "metadata": {},
"name": "Bob", "phone": null, "preferred_locales": [], "shipping": null, "tax_exempt": "none", "test_clock":
null }
```

In the customers tab of the stripe portal we can now see the customer that we just added via the cURL API code.



Now I use composer from the terminal to install the stripe SDK files in the directory where I have this code located –

`c:\xampp3\htdocs\hollingworthAPIs\09-api-SDK>`**`composer require stripe/stripe-php`**

```
Info from https://repo.packagist.org: #StandWithUkraine
Using version ^10.0 for stripe/stripe-php
./composer.json has been created
Running composer update stripe/stripe-php
Loading composer repositories with package information
Updating dependencies
Lock file operations: 1 install, 0 updates, 0 removals
  - Locking stripe/stripe-php (v10.0.0)
Writing lock file
Installing dependencies from lock file (including require-dev)
Package operations: 1 install, 0 updates, 0 removals
  - Downloading stripe/stripe-php (v10.0.0)
  - Installing stripe/stripe-php (v10.0.0): Extracting archive
Generating autoload files
No security vulnerability advisories found
```

The SDK packages are installed in the folder specified.

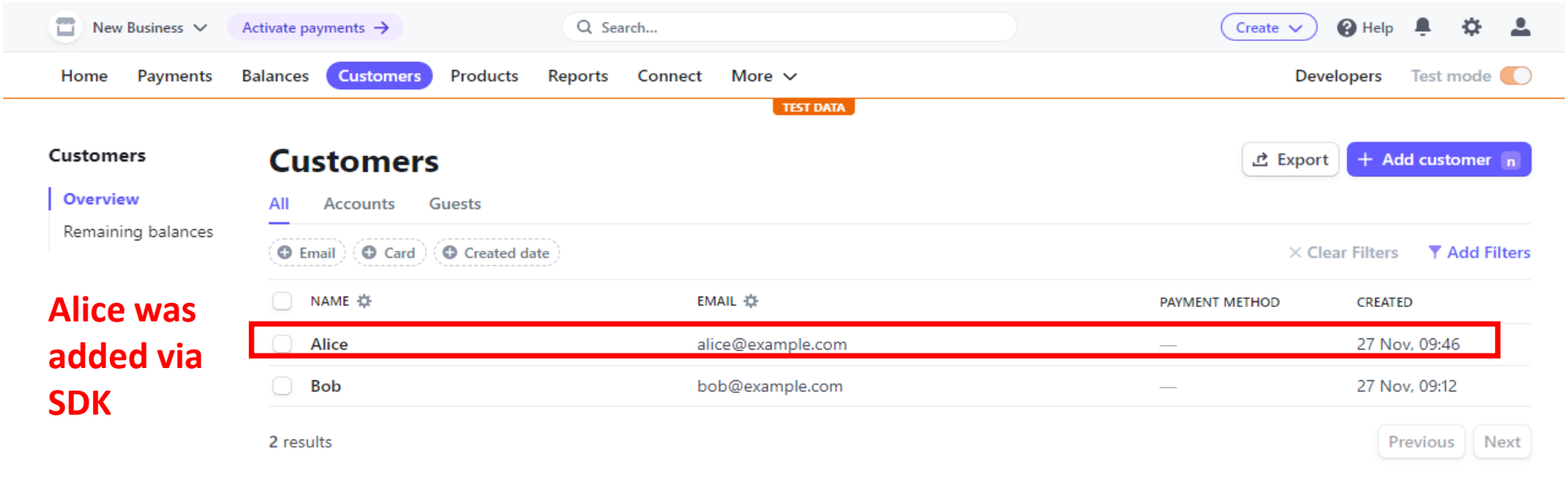
```php
<?php

$api_key = "sk_test_xxxxxxxxxx";

$data = [
    "name" => "Alice",
    "email" => "alice@example.com"
];

require "stripeSDK/vendor/autoload.php";

$stripe = new \Stripe\StripeClient($api_key);

$customer = $stripe->customers->create($data);

echo $customer;
?>
```

This time we require the path to the stripe autoload.php file.

We substantiate a new client passing in the api key.

And use the create method from the customers class


```
Stripe\Customer JSON: { "id": "cus_MsRQKEEgpF7pil",
"object": "customer", "address": null, "balance":
0, "created": 1669538931, "currency": null,
"default_source": null, "delinquent": false,
"description": null, "discount": null, "email":
"alice@example.com", "invoice_prefix": "652683D7",
"invoice_settings": { "custom_fields": null,
"default_payment_method": null, "footer": null,
"rendering_options": null }, "livemode": false,
"metadata": [], "name": "Alice", "phone": null,
"preferred_locales": [], "shipping": null,
"tax_exempt": "none", "test_clock": null }
```

Alice was added via SDK

Create a RESTful API: build a framework for serving the API

Initial setup of API code

The project starts with the directory structre shown on the right, where any URL after the api directory routes to index.php using a simple htaccess file. Index.php is now a front controller because all URLs will route through this file.

```
xampp3/htdocs/hollingworthAPIs/10-create-RESTful-api/api/index.php

RewriteEngine On
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d
RewriteCond %{REQUEST_FILENAME} !-l
RewriteRule . index.php [L]
```

```
<?php
$path = parse_url($_SERVER["REQUEST_URI"], PHP_URL_PATH);
$parts = explode("/", $path);
$resource = $parts[4];
$id = $parts[5] ?? null;
$method = $_SERVER["REQUEST_METHOD"];

echo "<p>Resource: <b>".$resource."</b> ::: id: <b>".$id."</b> ::: method: <b>".$method."</b></p>";
?>
```

In the front end controller (index.php) We have URL handelling code that extracts the resource and id if there is one.

URL/path	Resource	id	Method
http://localhost/hollingworthAPIs/10-create-RESTful-api/api/product	product		GET
http://localhost/hollingworthAPIs/10-create-RESTful-api/api/product/123	product	123	GET
http://localhost/hollingworthAPIs/10-create-RESTful-api/api/product/123/page=1	products	123	GET
http://localhost/hollingworthAPIs/10-create-RESTful-api/api/tasks	tasks		GET
http://localhost/hollingworthAPIs/10-create-RESTful-api/api/tasks/387	tasks	387	GET

Use a Client for API development: cURL, Postman or HTTPie

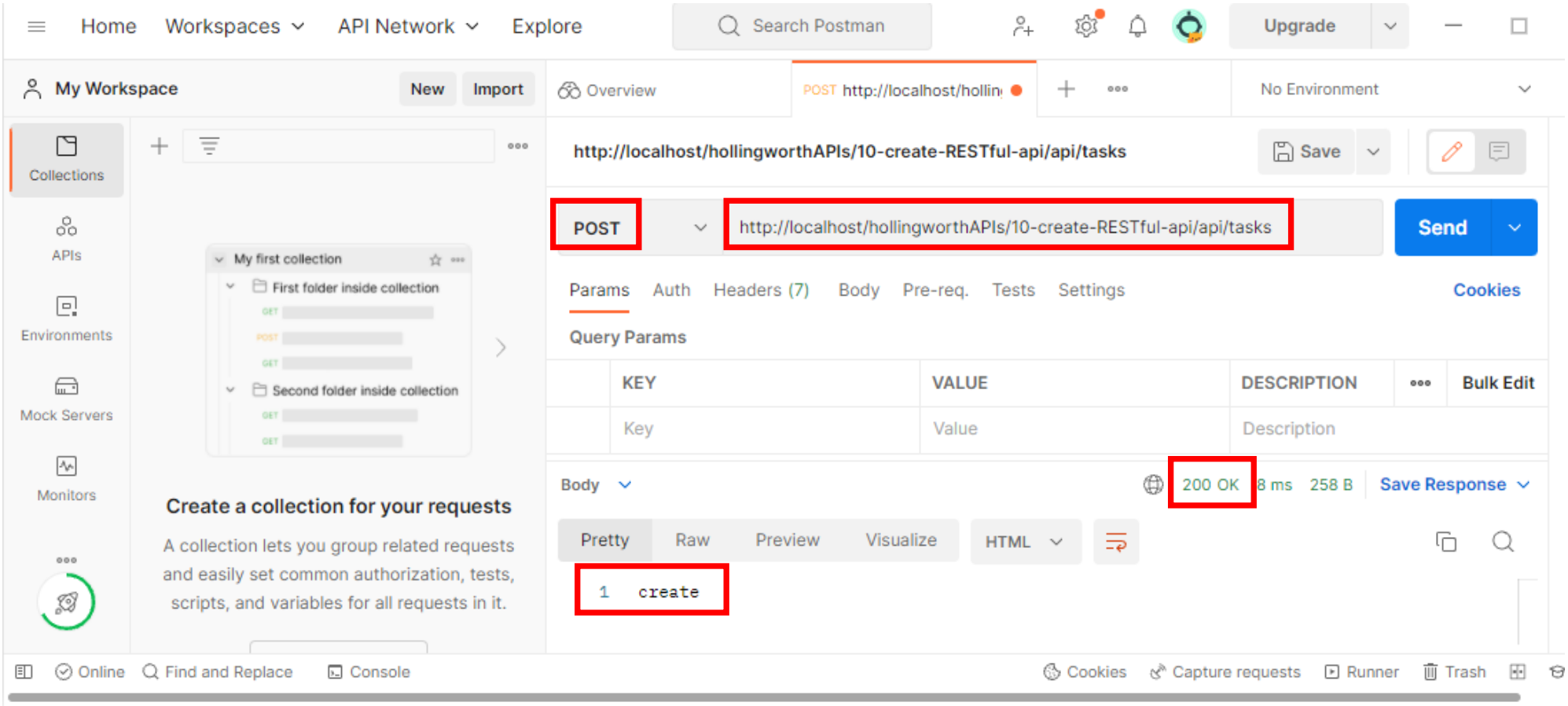
cURL	Postman	HTTPie
Is preinstalled on most modern computers but can also be downloaded.	A GUI based application that can be used from the browser or downloadable app to generate URLs.	HTTPie is a comad line tool that is more user friendly than cURL.
https://curl.se/download.html	https://www.postman.com	https://httpie.io

```php
<?php

class TaskController
{
    public function processRequest($method, $id) {
        if ($id === null) {
            if ($method == "GET") {
                echo "index";
            } elseif ($method == "POST") {
                echo "create";
            }
        } else {

            switch ($method) {
                case "GET":
                    echo "show $id";
                    break;
                case "PATCH":
                    echo "update $id";
                    break;
                case "DELETE":
                    echo "delete $id";
                    break;
            }
        }
    }
}
```

In a new folder called src we create a TaskController.php file and define some if else and switch statements in a class to handle the possible scenarios for the methods being passed into the URL.

In the index.php file we substantiate the class and call the processRequest function.

```php
<?php
$path = parse_url($_SERVER["REQUEST_URI"], PHP_URL_PATH);
$parts = explode("/", $path);
$resource = $parts[4];
$id = $parts[5] ?? null;
$method = $_SERVER["REQUEST_METHOD"];

if ($resource != "tasks") {
    http_response_code(404);
    exit;
}

require dirname(__DIR__) . "/src/TaskController.php";
$controller = new TaskController;
$controller->processRequest($_SERVER['REQUEST_METHOD'], $id);
?>
```



In the postman app that I have installed on my computer I can now experiment by sending API requests, changing the method from POST, GET, PATCH, DELETE, etc and the URL observing the response code, the header and body of the http response.

Use composers autoloader to load classes automatically

Step 1: in the root folder of the project create a composer.json file

```
xampp3/htdocs/hollingworthAPIs/10-create-RESTful-api/
```

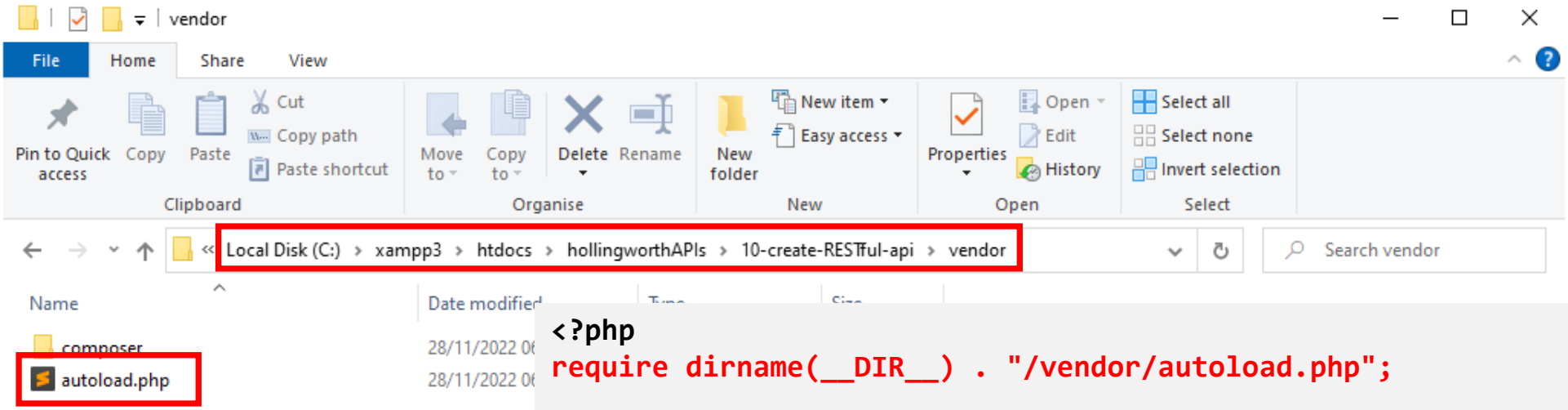
Step 2: input the following code. What it does is autoload all files and classes. Psr-4 means where the classname and filename are the same, from the folder src.

```
{
  "autoload": {
    "psr-4": {
      "": "src/"
    }
  }
}
```

Step 3: launch composer from the command line. And CD into the root directory of our project. Run a composer dump-autoload command.

```
C:\Windows\system32>composer
   ______
  / ____/___  ____ ___  ____  ____  ________  _____
 / /   / __ \/ __ `__ \/ __ \/ __ \/ ___/ _ \/ ___/
/ /___/ /_/ / / / / / / /_/ / /_/ (__  )  __/ /
\____/\____/_/ /_/ /_/ .___/\____/____/\___/_/
                    /_/

C:\Windows\system32>cd c:\xampp3\htdocs\hollingworthAPIs\10-create-RESTful-api
C:\xampp3\htdocs\hollingworthAPIs\10-create-RESTful-api>composer dump-autoload

Generating autoload files
Generated autoload files

c:\xampp3\htdocs\hollingworthAPIs\10-create-RESTful-api>
```

Step 4: Composer will create a folder called vendor in the root that contains an autoload.php file. We do not need to manually modify this file because it is controlled by composer.



Step 5: Now we can include the autoload.php file at the top of our document and remove the manual loading of class files.

```php
<?php
require dirname(__DIR__) . "/vendor/autoload.php";

$path = parse_url($_SERVER["REQUEST_URI"], PHP_URL_PATH);
$parts = explode("/", $path);
$resource = $parts[4];
$id = $parts[5] ?? null;
$method = $_SERVER["REQUEST_METHOD"];

if ($resource != "tasks") {
    http_response_code(404);
    exit;
}

// require dirname(__DIR__) . "/src/TaskController.php";
$controller = new TaskController;
$controller->processRequest($_SERVER['REQUEST_METHOD'], $id);

?>
```

Make Debugging Easier: add type declarations & enable strict type checking

The task controller processRequest function takes two parameters which are coming from the URL and so should always be a string, even if it is a number. We can enforce this rule.

```
<?php

class TaskController
{
    public function processRequest(string $method, string $id): void  {
        if ($id === null) {

            if ($method == "GET") {
 . . .
```

```
<?php
declare(strict_types=1);

require dirname(__DIR__) . "/vendor/autoload.php";
. . .
```

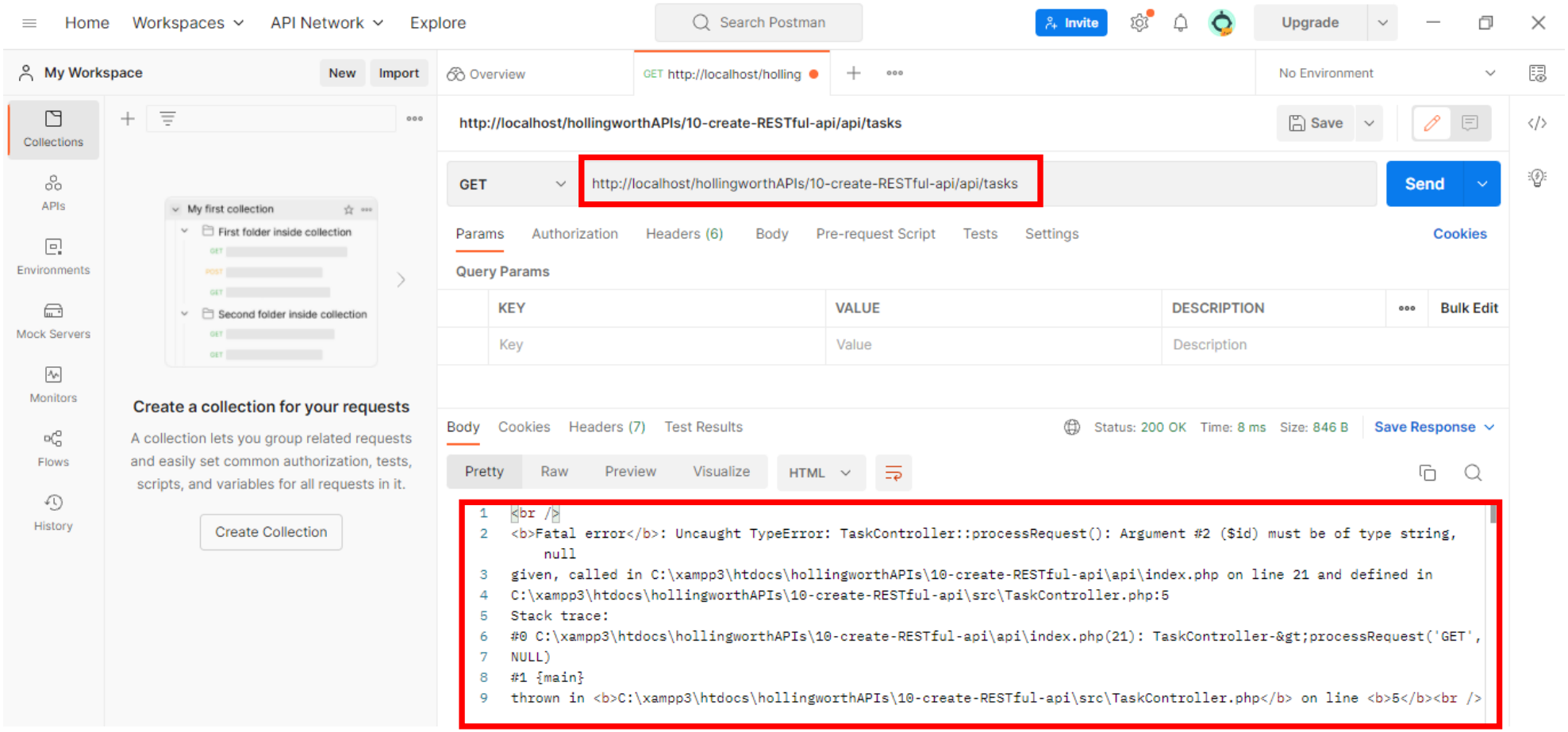
In the index.php file, we declare strict_types=1. As all requests are going through this script the setting will be applied globally.

Now in postman if I try to get a task with an id number it works as before.
http://localhost/hollingworthAPIs/10-create-RESTful-api/api/tasks/123

But if I try to get the index of tasks it thorws an error.

`http://localhost/hollingworthAPIs/10-create-RESTful-api/api/tasks/123`

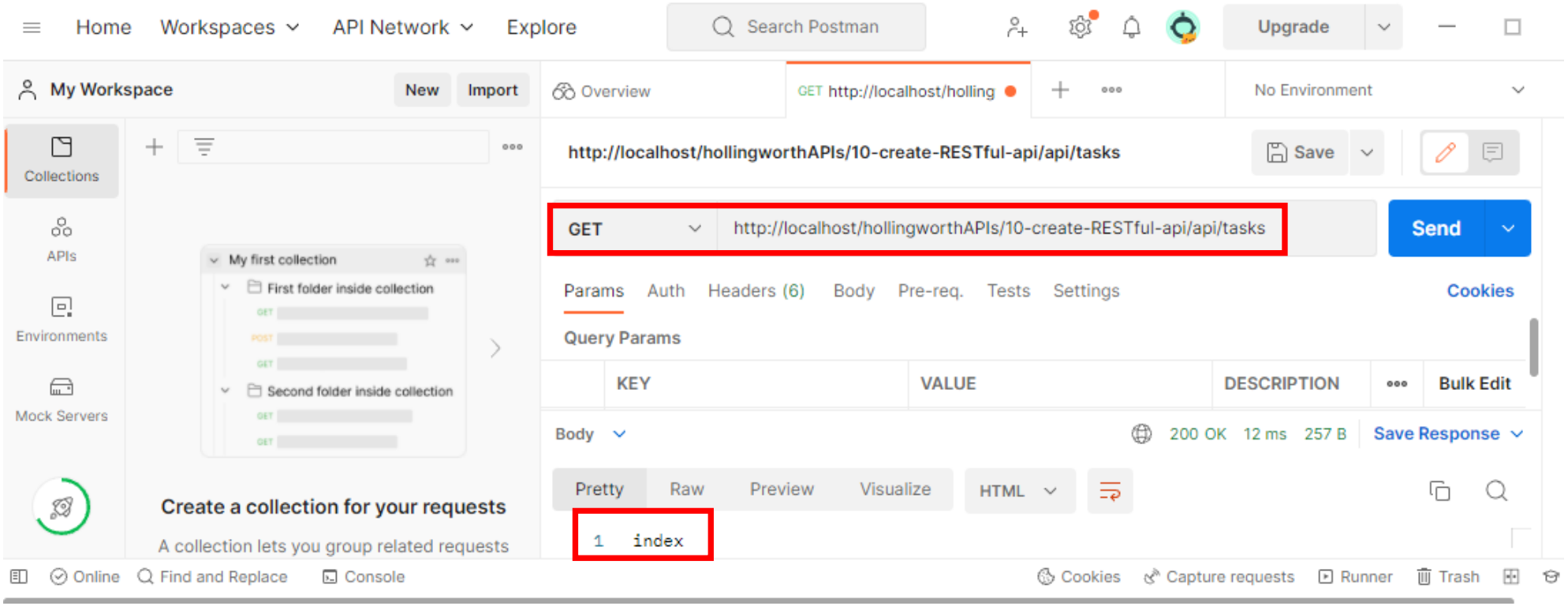
The reason for the error: ***`Argument #2 ($id) must be of type string, null`***. the second value of ID is expected to be a string but null was given.



```php
<?php

class TaskController
{
    public function processRequest(string $method, ?string $id): void  {
        if ($id === null) {

            if ($method == "GET") {
 . . .
```

We can make the id value nullable by prefixing it with a **question mark**.

Now an empty string value for ID will be accepted without errors.



Always return JSON: add a generic exception handler and json content-type header

In the previous lesson, the error was returned formatted as HTML because it is assumed that it would be viewed in a browser but as this is an API and will be called by another machine we want everything in JSON format.

```php
<?php
class ErrorHandler
{
    public static function handleException(Throwable $exception): void {
        http_response_code(500);

        echo json_encode([
            "code" => $exception->getCode(),
            "message" => $exception->getMessage(),
            "file" => $exception->getFile(),
            "line" => $exception->getLine()
        ]);
    }
}
```

In the src folder we create a seperate class called ErrorHandler to handle the exceptions and response code. It json_encodes an array of the throwable errors.

```php
<?php
declare(strict_types=1);

require dirname(__DIR__) . "/vendor/autoload.php";

set_exception_handler("Errorhandler::handleException");
. . .
```

In the index.php file we use the set_exception_handler and set it to the class and function. Note that we no not manually require this class because the autoloader is taking care of that.

Now If we recreate the error scenario from before then we see that the error is returned as JSON and with a status code of 500 Internal Server Error. However the content-type is still text/html; charset=UTF-8

```
{
"code":0,
"message":"TaskController::processRequest(): Argument #2 ($id) must be of type string, null given, called in
C:\\xampp3\\htdocs\\hollingworthAPIs\\10-create-RESTful-api\\api\\index.php on line 20",
"file":"C:\\xampp3\\htdocs\\hollingworthAPIs\\10-create-RESTful-api\\src\\TaskController.php",
"line":5
}
```

```php
<?php
declare(strict_types=1);

require dirname(__DIR__) . "/vendor/autoload.php";

set_exception_handler("ErrorHandler::handleException");

$path = parse_url($_SERVER["REQUEST_URI"], PHP_URL_PATH);
$parts = explode("/", $path);
$resource = $parts[4];
$id = $parts[5] ?? null;

if ($resource != "tasks") {
    http_response_code(404);
    exit;
}

Header("Content-Type: application/JSON charset=UTF-8");

$controller = new TaskController;
$controller->processRequest($_SERVER['REQUEST_METHOD'], $id);
?>
```

As all the responses from this API program will be in JSON format we can use the Header() function to manually define the content-type and charset.

Return a 405 status code and allow header for invalid methods

In the task controller we are outputting the response based on the action and the URL.

For example a GET request to tasks will return a list of tasks and a POST will create a new task but what happens if we try another method? Well it is an invalid request and should send a 405 method not allowed and contain a header with a list of allowed methods.

```php
<?php
class TaskController {

    public function processRequest(string $method, ?string $id): void  {
        if ($id === null) {
            if ($method == "GET") {
            echo "index";
            } elseif ($method == "POST") {
                echo "create";
            } else {
                http_response_code(405);
                Header("Allow: GET, POST");
            }
        } else {

            switch ($method) {
                case "GET":
                    echo "show $id";
                    break;
                case "PATCH":
                    echo "update $id";
                    break;
                case "DELETE":
                    echo "delete $id";
                    break;
            }
        }
    }
}
```

We can handle this with an else statement setting the response code and header.

2) We can call this function passing a string of the allowed methods.

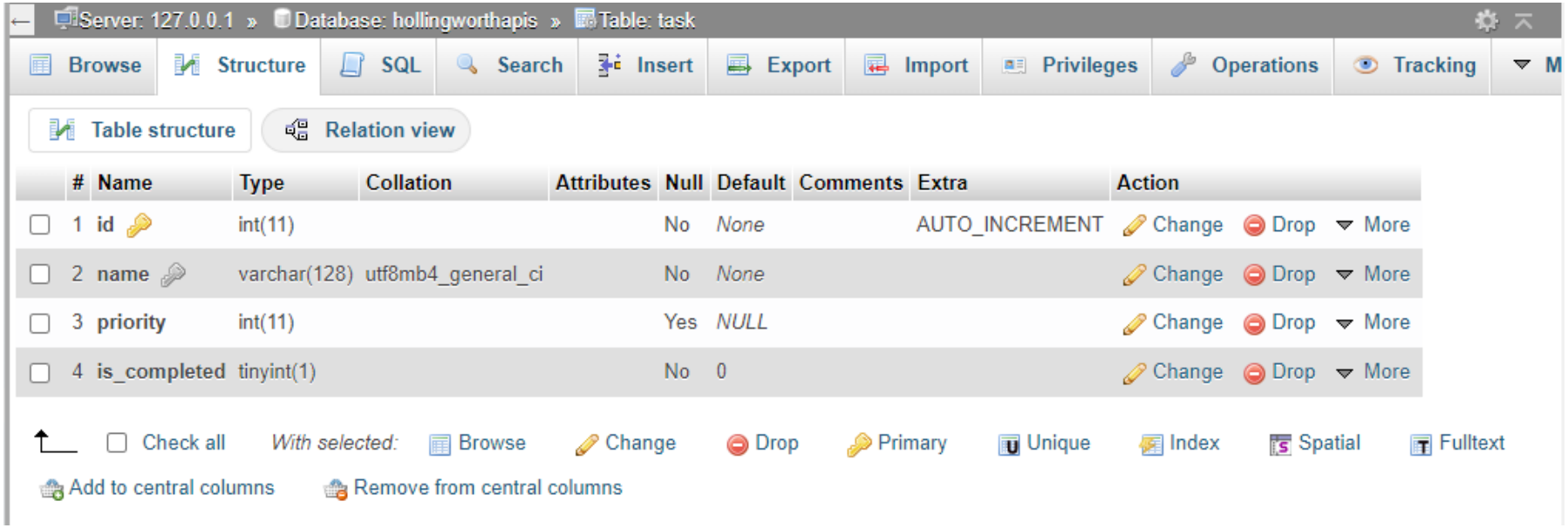
3) We can call this function passing a string of the allowed methods.

1) A separate function is created to handle not allowed methods and it takes a string variable of allowed methods. It sets the https response code and header.

```php
<?php

class TaskController {
    public function processRequest(string $method, ?string $id): void  {
        if ($id === null) {

            if ($method == "GET") {

                echo "index";
            } elseif ($method == "POST") {

                echo "create";
            } else {
                $this->respondMethodNotAllowed("GET, POST");
            }
        } else {

            switch ($method) {

                case "GET":
                    echo "show $id";
                    break;

                case "PATCH":
                    echo "update $id";
                    break;

                case "DELETE":
                    echo "delete $id";
                    break;

                default:
                    $this->respondMethodNotAllowed("GET, PATCH, DELETE");
            }
        }
    }

    private function respondMethodNotAllowed(string $allowed_methods): void {
        http_response_code(405);
        Header("Allow: $allowed_methods");
    }
}
```

Create a RESTful API: create a database and retrieve data from it

Create database and table




```
CREATE TABLE task (
    id INT NOT NULL AUTO_INCREMENT,
    name VARCHAR(128) NOT NULL,
    priority INT DEFAULT NULL,
    is_completed BOOLEAN NOT NULL DEFAULT FALSE,
    PRIMARY KEY (id),
    INDEX (name)
);
```

Connect to the database from PHP: add a Database class

```php
<?php

class Database
{
    public function __construct(
        private string $host,
        private string $name,
        private string $user,
        private string $password
    ) {
    }

    public function getConnection(): PDO {
        $dsn = "mysql:host={$this->host};dbname={$this->name};charset=utf8";

        return new PDO($dsn, $this->user, $this->password, [
            PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION
        ]);
    }
}
```

A standard database class that thakes four parameters to make a PDO connection to the database.

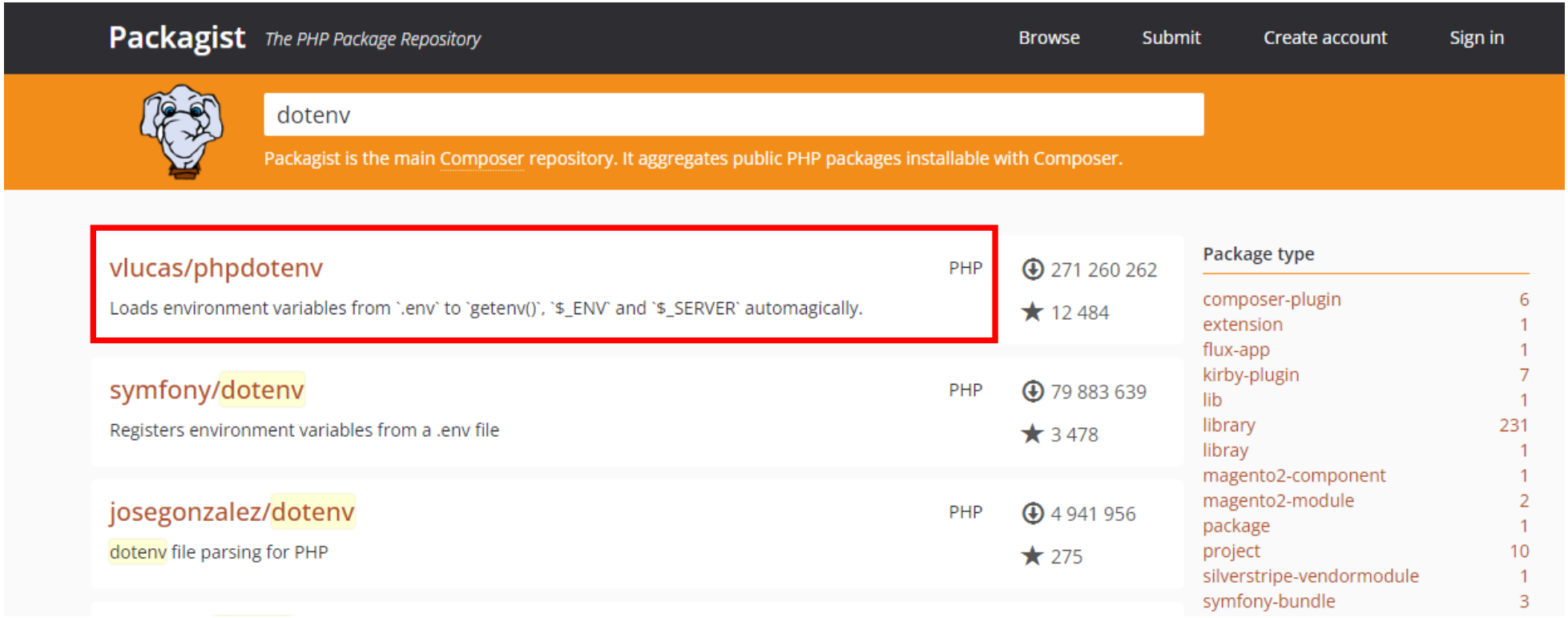
```php
<?php
declare(strict_types=1);

require dirname(__DIR__) . "/vendor/autoload.php";

set_exception_handler("ErrorHandler::handleException");

$path = parse_url($_SERVER["REQUEST_URI"], PHP_URL_PATH);
$parts = explode("/", $path);
$resource = $parts[4];
$id = $parts[5] ?? null;

if ($resource != "tasks") {
    http_response_code(404);
    exit;
}

Header("Content-Type: application/JSON charset=UTF-8");

$database = new Database("localhost", "hollingworthapis", "root", "");
$database->getConnection();

$controller = new TaskController;
$controller->processRequest($_SERVER['REQUEST_METHOD'], $id);
?>
```

From the controller we can call the instantiate the database class with the four credentials parameters and call the getConnection function.

Move the database connection to a seperate .env file

```
$database = new Database("localhost", "hollingworthapis", "root", "");
```

Rather than have our database credentials in the php file which might pose a security risk we can use a third party package from https://packagist.org/?query=dotenv to create an .env file.



```
c:\xampp3\htdocs\hollingworthAPIs\10-create-RESTful-api>composer require vlucas/phpdotenv
Using version ^5.5 for vlucas/phpdotenv
./composer.json has been updated
Running composer update vlucas/phpdotenv
Loading composer repositories with package information
Updating dependencies
Lock file operations: 6 installs, 0 updates, 0 removals
  - Locking graham-campbell/result-type (v1.1.0)
  - Locking phpoption/phpoption (1.9.0)
  - Locking symfony/polyfill-ctype (v1.27.0)
  - Locking symfony/polyfill-mbstring (v1.27.0)
  - Locking symfony/polyfill-php80 (v1.27.0)
  - Locking vlucas/phpdotenv (v5.5.0)
Writing lock file
Installing dependencies from lock file (including require-dev)
Package operations: 6 installs, 0 updates, 0 removals
  - Downloading symfony/polyfill-php80 (v1.27.0)
  - Downloading symfony/polyfill-mbstring (v1.27.0)
  - Downloading symfony/polyfill-ctype (v1.27.0)
  - Downloading phpoption/phpoption (1.9.0)
  - Downloading graham-campbell/result-type (v1.1.0)
  - Downloading vlucas/phpdotenv (v5.5.0)
  - Installing symfony/polyfill-php80 (v1.27.0): Extracting archive
  - Installing symfony/polyfill-mbstring (v1.27.0): Extracting archive
  - Installing symfony/polyfill-ctype (v1.27.0): Extracting archive
  - Installing phpoption/phpoption (1.9.0): Extracting archive
  - Installing graham-campbell/result-type (v1.1.0): Extracting archive
  - Installing vlucas/phpdotenv (v5.5.0): Extracting archive
Generating autoload files
6 packages you are using are looking for funding.
Use the `composer fund` command to find out more!
No security vulnerability advisories found

c:\xampp3\htdocs\hollingworthAPIs\10-create-RESTful-api>
```

From the ROOT folder of the project run
composer require vlucas/phpdotenv

```
DB_HOST="localhost"
DB_NAME="hollingworthapis"
DB_USER="api_db_user"
DB_PASS="pa55word"
```

In the ROOT directory of our project we create a file called .env which must start with a dot and does not have a file type extension such as php. In this file we store the key value pairs for our database credentials.

```php
<?php
declare(strict_types=1);

require dirname(__DIR__) . "/vendor/autoload.php";

$dotenv = Dotenv\Dotenv::createImmutable(dirname(__DIR__));
$dotenv->load();

set_exception_handler("ErrorHandler::handleException");

$path = parse_url($_SERVER["REQUEST_URI"], PHP_URL_PATH);
$parts = explode("/", $path);
$resource = $parts[4];
$id = $parts[5] ?? null;

if ($resource != "tasks") {
    http_response_code(404);
    exit;
}

Header("Content-Type: application/JSON charset=UTF-8");

$database = new Database($_ENV["DB_HOST"], $_ENV["DB_NAME"], $_ENV["DB_USER"], $_ENV["DB_PASS"]);

$controller = new TaskController;
$controller->processRequest($_SERVER['REQUEST_METHOD'], $id);
```

We call the dotenv package and class called createImmutable and load it.

We now reference the 4 database connection credientials in the .env file rather than having them directly in our php file.

Create a table data gateway class for the resource table

```php
<?php

class TaskGateway {
    private PDO $conn;

    public function __construct(Database $database) {
        $this->conn = $database->getConnection();
    }
}
```

In the src folder we create another file called TaskGateway.php that is an object that will serve as a gateway to the tasks table.

First we will declare a private variable to store the connection.

The tasks gatway requires a database connection so we can pass in this dependancy in the constructor. In the constuctor we will call the getConnection method from the database.

```php
<?php

class TaskController {
    public function __construct(private TaskGateway $gateway) {}
. . .
```

In the task controller we will pass in the task gateway in the constructor.

```php
$taskGateway = new TaskGateway($database);
$controller = new TaskController($taskGateway);
```

In the front controller (index.php) When we create an object of the controller class we need to pass in an object of the gateway class.

The gatway object requires a database connection wich we can pass in. Now we can pass the gateway into the controller.

Show a list of all records

```php
<?php

class TaskGateway {
    private PDO $conn;

    public function __construct(Database $database) {
        $this->conn = $database->getConnection();
    }

    public function getAll(): array {
        $sql = "SELECT * FROM task ORDER BY name";
        $stmt = $this->conn->query($sql);
        return $stmt->fetchAll(PDO::FETCH_ASSOC);
    }
}
```

In TaskGateway.php file we can add an SQL query to get all rows from the table and use PDO to return the data.

Because the task table is currently empty we can inser some test data.

```sql
INSERT INTO task
    (name, priority, is_completed)
VALUES
    ('Buy new shoes', 1, true),
    ('Renew passport', 2, false),
    ('Paint wall', NULL, true)
;
```

Now a GET request to the below URL returns a JSON object with the three rows of data.

`http://localhost/hollingworthAPIs/10-create-RESTful-api/api/tasks`


```json
[
    {
        "id": 1,
        "name": "Buy new shoes",
        "priority": 1,
        "is_completed": 1
    },
    {
        "id": 3,
        "name": "Paint wall",
        "priority": null,
        "is_completed": 1
    },
    {
        "id": 2,
        "name": "Renew passport",
        "priority": 2,
        "is_completed": 0
    }
]
```

Configure PDO to prevent numeric values from being converted to strings

https://www.php.net/manual/en/pdo.setattribute.php

PDO can convert non string values to string but we can change some attributes to prevent this.

PDO::ATTR_STRINGIFY_FETCHES: This will stringify all fields returned from the database and is usually set to false but we can verify it.

PDO::ATTR_EMULATE_PREPARES: If the database does not support prepared statements PDO will emulate it. We can set this attribute to false because our database does hanndle prepared statements.

```php
<?php
class Database
{
    public function __construct(
        private string $host,
        private string $name,
        private string $user,
        private string $password
    ) {
    }

    public function getConnection(): PDO {
        $dsn = "mysql:host={$this->host};dbname={$this->name};charset=utf8";

        return new PDO($dsn, $this->user, $this->password, [
            PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION,
            PDO::ATTR_STRINGIFY_FETCHES => false,
            PDO::ATTR_EMULATE_PREPARES => false
        ]);
    }
}
```

Convert database booleans to boolean literals in JSON

In phpmyadmin database boolean values are stored as a tiny integer, that is true is 1 and false is 0. we can change it so that in the JSON response we have true or false as a word rather than a single digit binary.

There is no easy way to change this setting globally so it will have to be done on the output of the SQL query.

```php
<?php
class TaskGateway {
    private PDO $conn;

    public function __construct(Database $database)
{
        $this->conn = $database->getConnection();
    }

    public function getAll(): array {
        $sql = "SELECT * FROM task ORDER BY name";
        $stmt = $this->conn->query($sql);
        // return $stmt->fetchAll(PDO::FETCH_ASSOC);
        $data = [];
        while ($row = $stmt-
>fetch(PDO::FETCH_ASSOC)) {
            $row['is_completed'] = (bool)
['is_completed'];
            $data[] = $row;
        }
        return $data;
    }
}
```

We initialise an empty array called data then iterate over the sql results in a while loop changing the is completed value to bollean.

```json
[
  {
    "id": 1,
    "name": "Buy new shoes",
    "priority": 1,
    "is_completed": true
  },
  {
    "id": 3,
    "name": "Paint wall",
    "priority": null,
    "is_completed": true
  },
  {
    "id": 2,
    "name": "Renew passport",
    "priority": 2,
    "is_completed": true
  }
]
```

Show an individual record

```php
<?php
class TaskGateway {
    private PDO $conn;

    public function __construct(Database $database) {
        $this->conn = $database->getConnection();
    }

    public function getAll(): array {
        $sql = "SELECT * FROM task ORDER BY name";
        $stmt = $this->conn->query($sql);
        // return $stmt->fetchAll(PDO::FETCH_ASSOC);
        $data = [];
        while ($row = $stmt->fetch(PDO::FETCH_ASSOC)) {
            $row['is_completed'] = (bool) ['is_completed'];
            $data[] = $row;
        }
        return $data;
    }

    public function get(string $id): array | false {
        $sql = "SELECT * FROM task WHERE id=:id";
        $stmt = $this->conn->prepare($sql);
        $stmt->bindValue(":id", $id, PDO::PARAM_INT);
        $stmt->Execute();
        $data = $stmt->fetch(PDO::FETCH_ASSOC);
        if ($data !== false) {
            $data['is_completed'] = (bool) ['is_completed'];
        }
        return $data;
    }
}
```

In TaskGateway.php file we can add an SQL query to get one rows from the table based on an id. Note that this function will return an array or false if the row is not found.

```php
<?php
class TaskController {
    public function __construct(private TaskGateway $gateway) {}

    public function processRequest(string $method, ?string $id): void  {
        if ($id === null) {
            if ($method == "GET") {
                echo json_encode($this->gateway->getAll());
                // echo "index";
            } elseif ($method == "POST") {

                echo "create";
            } else {
                $this->respondMethodNotAllowed("GET, POST");
            }
        } else {
            switch ($method) {
                case "GET":
                    echo json_encode($this->gateway->get($id));
                    break;
                case "PATCH":
                    echo "update $id";
                    break;
                case "DELETE":
                    echo "delete $id";
                    break;
                default:
                    $this->respondMethodNotAllowed("GET, PATCH, DELETE");
            }
        }
    }
    private function respondMethodNotAllowed(string $allowed_methods): void {
        http_response_code(405);
        Header("Allow: $allowed_methods");
    }
}
```

In TaskController.php file we can call the previously created method and json_encode it.

Now when we call the API with the correct URL from postman we get one row encoded as a JSON object.

`http://localhost/hollingworthAPIs/10-create-RESTful-api/api/tasks/1`


```json
{
  "id": 1,
  "name": "Buy new shoes",
  "priority": 1,
  "is_completed": true
}
```

```php
<?php
class TaskController {
    public function __construct(private TaskGateway $gateway) {}

    public function processRequest(string $method, ?string $id): void  {
        if ($id === null) {
            if ($method == "GET") {
                echo json_encode($this->gateway->getAll());
                // echo "index";
            } elseif ($method == "POST") {
                echo "create";
            } else {
                $this->respondMethodNotAllowed("GET, POST");
            }
        } else {
            $task = $this->gateway->get($id);
            if ($task === false) {
                $this->respondNotFound($id);
                return;
            }

            switch ($method) {
                case "GET":
                    echo json_encode($task);
                    break;
                case "PATCH":
                    echo "update $id";
                    break;
                case "DELETE":
                    echo "delete $id";
                    break;
                default:
                    $this->respondMethodNotAllowed("GET, PATCH, DELETE");
            }
        }
    }
```

When we call the get method in task gateway and it returns false this means that there is no row. We can use an if statement to respond with the correct 404 status code and a custom message. This is done in a separate unction called respondNotFound.

Respond with a 404 if the resourse with the specified ID is not found

If we call a specific resource that is not in the database then it will return false in the body but a status 200 OK in the header.

`http://localhost/hollingworthAPIs/10-create-RESTful-api/api/tasks/4` false

```
    private function respondMethodNotAllowed(string $allowed_methods): void {
        http_response_code(405);
        Header("Allow: $allowed_methods");
    }

    private function respondNotFound(string $id): void {
        http_response_code(404);
        echo json_encode(["message" => "Task with ID $id not found"]);
    }
}
```

http://localhost/hollingworthAPIs/10-create-RESTful-api/api/tasks/4 Status 404 not found


```
{
  "message": "Task with ID 4 not found"
}
```

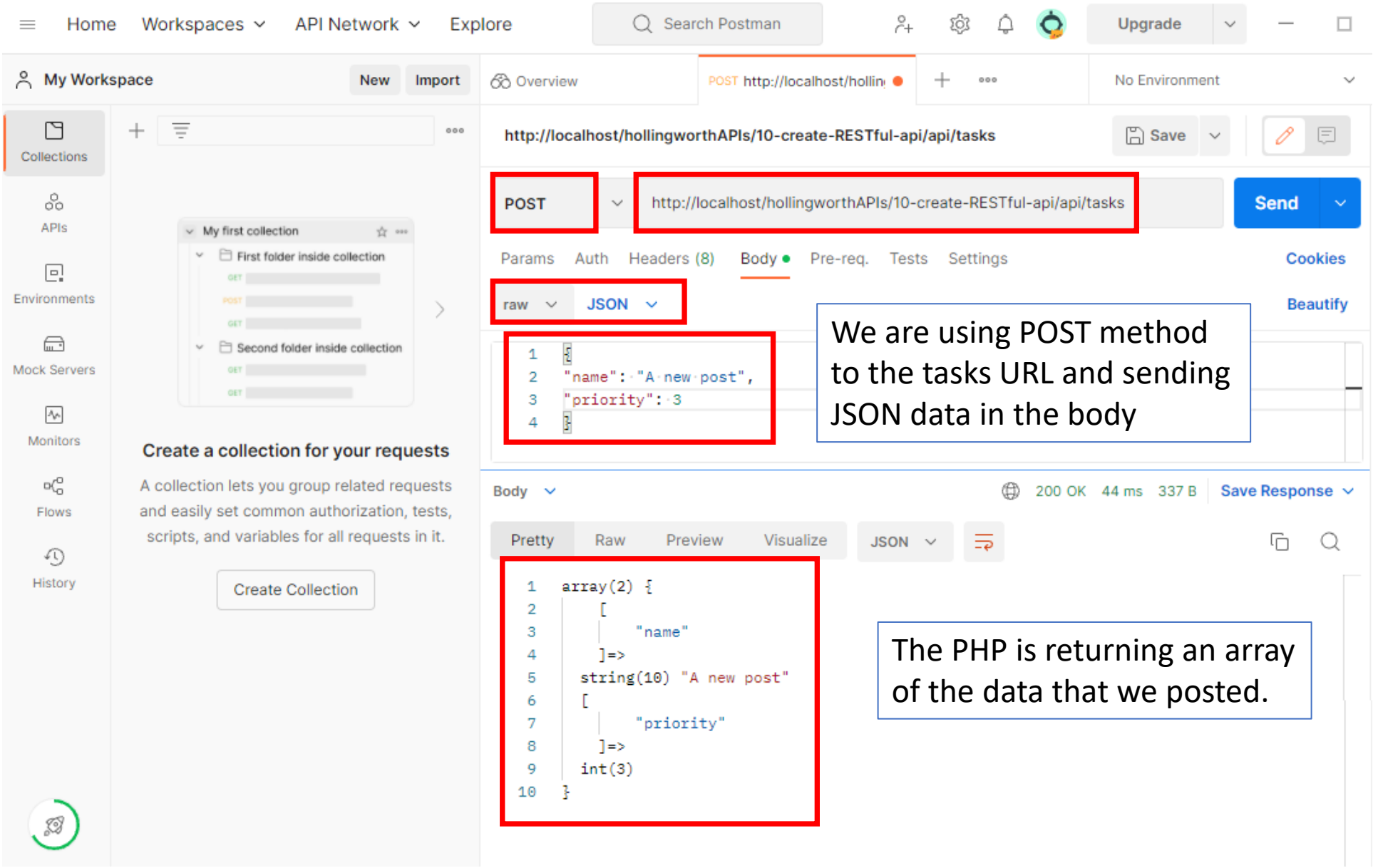
Create a RESTful API: create, update and delete individual resources

Get the data from the request as JSON

```
<?php

class TaskController {
    public function __construct(private TaskGateway $gateway) {}

    public function processRequest(string $method, ?string $id): void  {
        if ($id === null) {

            if ($method == "GET") {
                echo json_encode($this->gateway->getAll());
                // echo "index";
            } elseif ($method == "POST") {
                $data = (array) json_decode(file_get_contents("php://input"), true);
                var_dump($data);
                // echo "create";
            } else {
                $this->respondMethodNotAllowed("GET, POST");
. . .
```

We can post data to the API using the POST method and passing the data in the body of the HTTP request as JSON.

In the PHP we need to decode the JSON input and dump it in an array.


Home
Workspaces
API Network
Explore
Search Postman
Upgrade
My Workspace
New
Import
Overview
POST http://localhost/hollin
No Environment
Collections
APIs
Environments
Mock Servers
Monitors
Flows
History
My first collection
First folder inside collection
Second folder inside collection
Create a collection for your requests
A collection lets you group related requests and easily set common authorization, tests, scripts, and variables for all requests in it.
Create Collection
http://localhost/hollingworthAPIs/10-create-RESTful-api/api/tasks
Save
POST
http://localhost/hollingworthAPIs/10-create-RESTful-api/api/tasks
Send
Params
Auth
Headers (8)
Body
Pre-req.
Tests
Settings
Cookies
raw
JSON
Beautify
{
"name": "A new post",
"priority": 3
}
We are using POST method to the tasks URL and sending JSON data in the body
Body
200 OK
44 ms
337 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
array(2) {
[
"name"
]=>
string(10) "A new post"
[
"priority"
]=>
int(3)
}
The PHP is returning an array of the data that we posted.

Insert a record into the database and respond with a 201 status code

```php
<?php

class TaskGateway {
    private PDO $conn;

    public function __construct(Database $database) {
        $this->conn = $database->getConnection();
    }

    . . .

    public function create(array $data): string {
        $sql = "INSERT INTO task (name, priority, is_completed)
                VALUES (:name, :priority, :is_completed)";
        $stmt = $this->conn->prepare($sql);
        $stmt->bindValue(":name", $data["name"], PDO::PARAM_STR);
        if (empty($data["priority"])) {
            $stmt->bindValue(":priority", null, PDO::PARAM_NULL);
        } else {
            $stmt->bindValue(":priority", $data["priority"], PDO::PARAM_INT);
        }
        $stmt->bindValue(":is_completed", $data["is_completed"] ?? false, PDO::PARAM_BOOL);
        $stmt->execute();
        return $this->conn->lastInsertId();
    }
}
```

We create a function that will insert the data into the database, passing in the data array and type casting the returned value as a string.

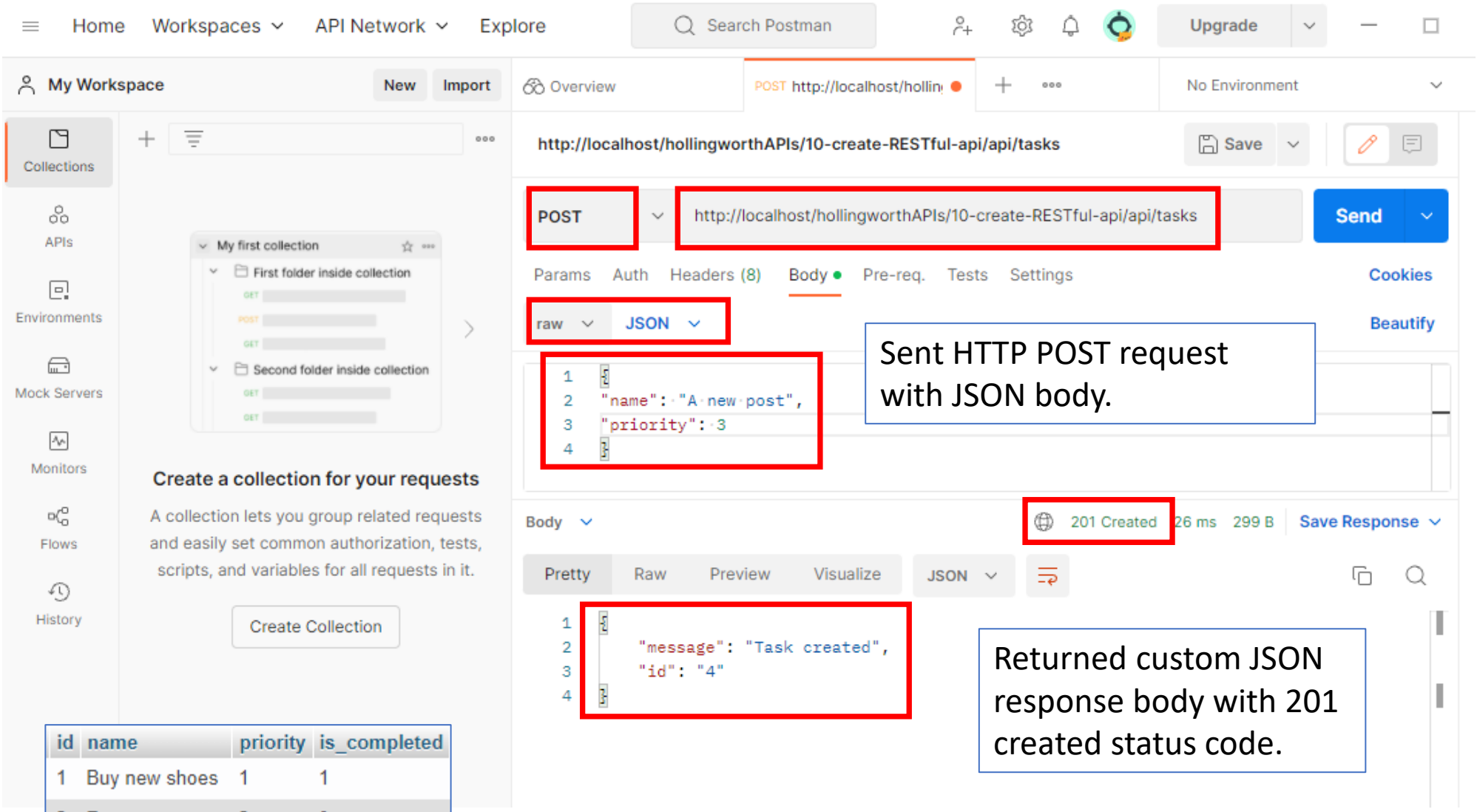
The priority column can take a null or an integer so we need to handle this with an if else statement for each scenario.

We use last ID function to return the last ID inserted into the database after executing the SQL.

```php
<?php

class TaskController {
    public function __construct(private TaskGateway $gateway) {}

    public function processRequest(string $method, ?string $id): void  {
        if ($id === null) {

            if ($method == "GET") {
                echo json_encode($this->gateway->getAll());
                // echo "index";
            } elseif ($method == "POST") {
                // print_r($_POST);
                $data = (array) json_decode(file_get_contents("php://input"), true);
                $id = $this->gateway->create($data);
                $this->respondCreated($id);
                // echo "create";
            }
            else {
                $this->respondMethodNotAllowed("GET, POST");
            }
        }
        . . .

    private function respondCreated($id): void {
        http_response_code(201);
        echo json_encode(["message" => "Task created", "id" => $id]);
    }
}
```

We call the create method on the data from the POST request.

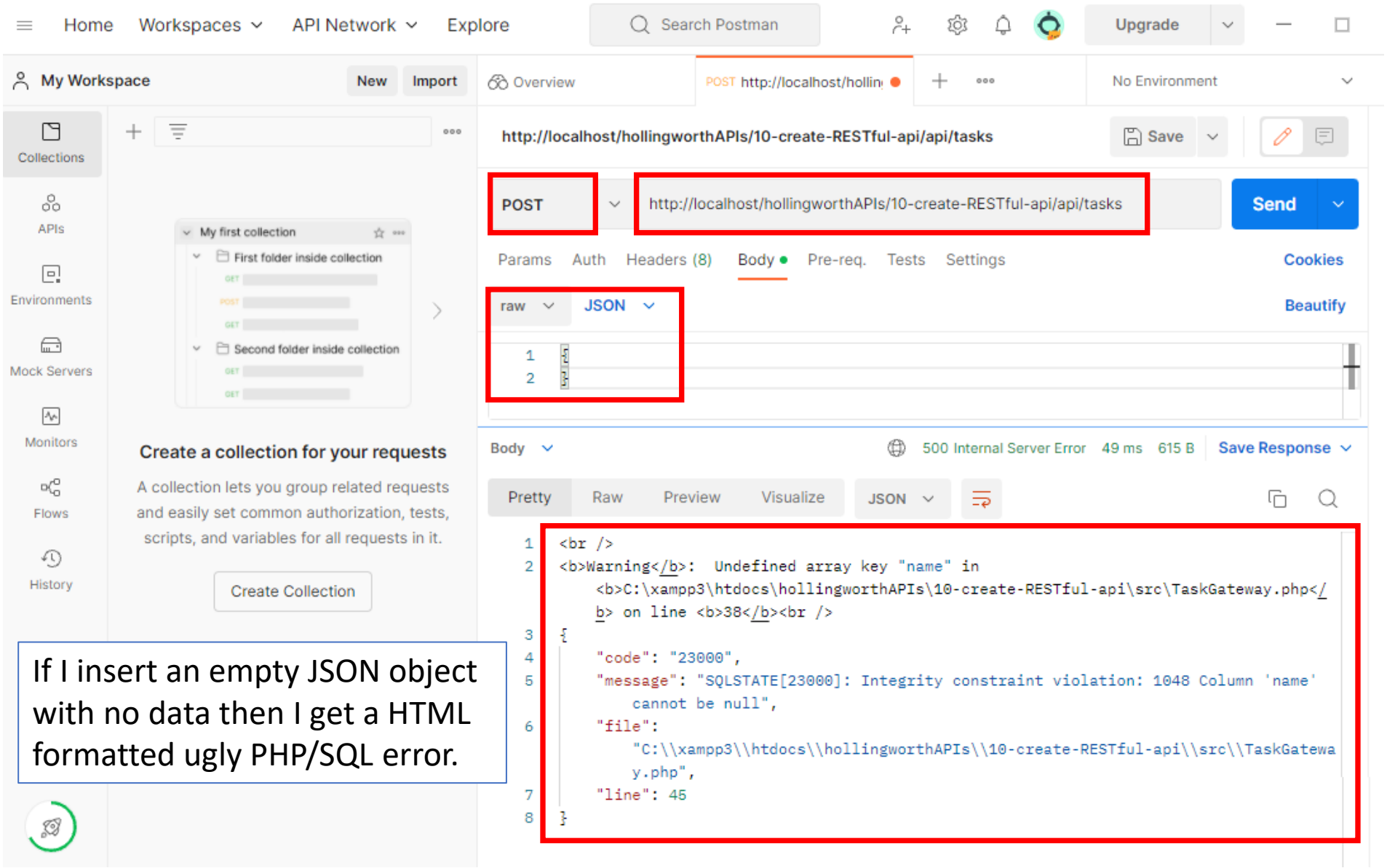
And respond with a custom JSON message and a 201 created status code.



id	name	priority	is_completed
1	Buy new shoes	1	1
2	Renew passport	2	0
3	Paint wall	*NULL*	1
4	A new post	3	0
5	task 2	6	0
6	task 3	*NULL*	0

I inserted some more rows with POST HTTP requests. Note that for task 3 I did not set a priority in the JSON object of values so it was given the default value of NULL.

Add a Generic error handler to output warnings as JSON



If I insert an empty JSON object with no data then I get a HTML formatted ugly PHP/SQL error.

The handleError method takes four perameters and throws an error exception which is delt with by the handleException function.
https://www.php.net/manual/en/function.set-error-handler.php

We just have to call the handleError function before he exception handler in the index.php file.

```php
<?php

class ErrorHandler {

    public static function handleError(
        int $errno,
        string $errstr,
        string $errfile,
        int $errline): void
    {
        throw new ErrorException($errstr, 0, $errno, $errfile, $errline);
    }

    public static function handleException(Throwable $exception): void {
        http_response_code(500);

        echo json_encode([
            "code" => $exception->getCode(),
            "message" => $exception->getMessage(),
            "file" => $exception->getFile(),
            "line" => $exception->getLine()
        ]);
    }
}
```

This will now return a pretty JSON formatted error object.

```
. . .
set_error_handler("ErrorHandler::handleError");
set_exception_handler("ErrorHandler::handleException");
. . .
```


```
{
  "code": 0,
  "message": "Undefined array key \"name\"",
  "file": "C:\\xampp3\\htdocs\\hollingworthAPIs\
\10-create-RESTful-api\\src\\TaskGateway.php",
  "line": 38
}
```

Validate the data and respond with a 422 status code if invalid

```
{
"name": ""
}
```

```
{
  "message": "Task created",
  "id": "7"
}
```

If I POST to the tasks URL with a JSON body where the name is empty then it still gets inserted into the database with the name column empty. This is undesirable and we need to validate the input before inserting it into the database.

```
private function getValidationErrors(array $data): array {
    $errors = [];
    if (empty($data["name"])) {
        $errors[] = "name is required";
    }
    if ( ! empty($data["priority"])) {
        if (filter_var($data["priority"], FILTER_VALIDATE_INT) === false) {
        $errors[] = "priority must be an integer";
        }
    }
    return $errors;
}
```

At the bottom of the task controller we will have a method for filtering the data.

First an empty array called errors is initialised. Then name value is checked and if empty it will insert a custom error message into the errors array.

Next if the priority value is not empty is is validated to check if it is an integer, returning a custom error if it is not.

This method returns the error array.

```
private function respondUnprocessableEntity(array $errors): void {
    http_response_code(422);
    echo json_encode(["errors" => $errors]);
}
```

This function will set the HTTP code to 422 unprocessable entity if the data fails validation and echo a custom error message.

```php
<?php

class TaskController {
    public function __construct(private TaskGateway $gateway) {}

    public function processRequest(string $method, ?string $id): void  {
        if ($id === null) {

            if ($method == "GET") {
                echo json_encode($this->gateway->getAll());
            } elseif ($method == "POST") {
                $data = (array) json_decode(file_get_contents("php://input"), true);
                $errors = $this->getValidationErrors($data);
                if (!empty($errors)) {
                    $this->respondUnprocessableEntity($errors);
                    return;
                }
                $id = $this->gateway->create($data);
                $this->respondCreated($id);
                // echo "create";
            }
. . .
```

Firstly an error variable is defined which passes the data into the validation method.

If the errors variable is not empty, i.e. the validation method returned an array with erors in it, then the respondUnprocessable Entity finction is invoked and we return after this to stop the code progressing onto the insert SQL section.

```
{
"name": ""
}
```

```
{
  "errors": [
    "name is required"
  ]
}
```

```
{
"name": "",
"priority": "abc"
}
```

```
{
  "errors": [
    "name is required",
    "priority must be an integer"
  ]
}
```

It returns JSON when one or both fields do not pass validation and 422 Unprossesible entity status code.

Conditionally Validate the data when updating an existing record

```
case "PATCH":
    $data = (array) json_decode(file_get_contents("php://input"), true);
    $errors = $this->getValidationErrors($data);
    if (!empty($errors)) {
        $this->respondUnprocessableEntity($errors);
        return;
    }
    echo "update $id";
    break;
```

We can reuse the validation code block from POST and copy it into the PATCH case.

```
PATCH   http://localhost/hollingworthAPIs/10-create-RESTful-api/api/tasks/8
```

```
{
"priority": 3
}
```

```
{
  "errors": [
    "name is required"
  ]
}
```

However if we only want to update a task priority and not a name and priority then this code needs to be modified to handle POST and PATCH.

```php
private function getValidationErrors(array $data, bool $is_new=true): array {
    $errors = [];
    if ($is_new && empty($data["name"])) {
        $errors[] = "name is required";
    }
    if ( ! empty($data["priority"])) {
        if (filter_var($data["priority"], FILTER_VALIDATE_INT) === false) {
            $errors[] = "priority must be an integer";
        }
    }
    return $errors;
}
```

We can modift the validation method to take a second parameter of Boolean called is_true and set the default value to true. We only validate the name if is_new is true.

```php
case "PATCH":
    $data = (array) json_decode(file_get_contents("php://input"), true);
    $errors = $this->getValidationErrors($data, false);
    if (!empty($errors)) {
        $this->respondUnprocessableEntity($errors);
        return;
    }
    echo "update $id";
    break;
```

Now when the data is passed into the validation function, we also pass in an additional parameter of false. This will cause the validation to skip the name field validation.

```
PATCH   http://localhost/hollingworthAPIs/10-create-RESTful-api/api/tasks/8
```

```
update 8
```

Validation works and the code progesses to the update $id line.

```
{
"priority": 3
}
```

Get the data from the request to update an existing record

```
public function update(string $id, Array $data) {
        $fields = [];
        if (!empty($data["name"])) {
            $fields["name"] =
            [$data["name"], PDO::PARAM_STR];
        }
        if (array_key_exists("priority", $data)) {
            $fields["priority"] =
            [$data["priority"], $data["priority"] === null ? PDO::PARAM_NULL : PDO::PARAM_INT];
        }
        if (array_key_exists("is_completed", $data)) {
            $fields["is_completed"] =
            [$data["is_completed"], PDO::PARAM_BOOL];
        }
        print_r($fields);
        exit;
    }
```

We need to dynamically create our SQL query specifying what fields to SET. We do this by creating a fields array and populating it with the fields and data type that are present in the data array.

Note that priority can be either NULL or an integer so we can use a ternary statement to handle this.

We call the update method from the PATCH case.

```
case "PATCH":
    $data = (array) json_decode(file_get_contents("php://input"), true);
    $errors = $this->getValidationErrors($data, false);
    if (!empty($errors)) {
        $this->respondUnprocessableEntity($errors);
        return;
    }
    $this->gateway->update($id, $data);
    break;
```

```
PATCH   http://localhost/hollingworthAPIs/10-create-RESTful-api/api/tasks/8
```

```
{
"priority": 3
}
```

```
Array(
[priority] => Array([0] => 3 [1] => 1)
)
```

```
PATCH   http://localhost/hollingworthAPIs/10-create-RESTful-api/api/tasks/8
```

```
{
"name": "Cook dinner at 5",
"priority": 3
}
```

```
Array(
    [name] => Array([0] => Cook dinner at 5  [1] => 2)
    [priority] => Array([0] => 3 [1] => 1)
)
```

```
PATCH   http://localhost/hollingworthAPIs/10-create-RESTful-api/api/tasks/8
```

```
{
"name": "Cook dinner at 5",
"priority": 3,
"is_completed": 1
}
```

```
Array(
    [name] => Array([0] => Cook dinner at 5 [1] => 2)
    [priority] => Array([0] => 3 [1] => 1)
    [is_completed] => Array([0] => 1 [1] => 5)
)
```

```
PATCH   http://localhost/hollingworthAPIs/10-create-RESTful-api/api/tasks/8
```

```
{
"name": "Cook dinner at 5",
"priority": null,
"is_completed": false
}
```

```
Array(
    [name] => Array([0] => Cook dinner at 5 [1] => 2)
    [priority] => Array([0] => [1] => 0)
    [is_completed] => Array([0] => [1] => 5)
)
```

The fields array contains sub arrays for each of the fields where key 0 is the value and key 1 is a numeric representation of the data type (string, Integer, Boolean)

Note that null and false are treated as empty so do not show in the array using print_r.

Update the record in the database and return a 200 status

```
public function update(string $id, array $data) {
    $fields = [];
    if ( ! empty($data["name"])) {
            $fields["name"] = [$data["name"], PDO::PARAM_STR];
    }
    if (array_key_exists("priority", $data)) {
        $fields["priority"] = [$data["priority"], $data["priority"] === null ? PDO::PARAM_NULL : PDO::PARAM_INT];
    }
    if (array_key_exists("is_completed", $data)) {
       $fields["is_completed"] = [$data["is_completed"], PDO::PARAM_BOOL];
    }
    if (empty($fields)) {
       return 0;
    } else {
        $sets = array_map(function($value) {
           return "$value = :$value";
        }, array_keys($fields));

        $sql = "UPDATE task"
             . " SET " . implode(", ", $sets)
             . " WHERE id = :id";

       $stmt = $this->conn->prepare($sql);
       $stmt->bindValue(":id", $id, PDO::PARAM_INT);
       foreach ($fields as $name => $values) {
            $stmt->bindValue(":$name", $values[0], $values[1]);
       }
       $stmt->execute();
       return $stmt->rowCount();
       }
    }
}
```

We use the array_map function to convert the values from $fields array into binding parameters for the SQL fields to be updated.

A foreach loop is used to iterate over the fields to create the bind values statements.

We return row count which will be 1 if the query executed successfully.

```
case "PATCH":
    $data = json_decode(file_get_contents("php://input"), true);
    $errors = $this->getValidationErrors($data, false);
    if (!empty($errors)) {
        $this->respondUnprocessableEntity($errors);
        return;
    }
    $rows = $this->gateway->update($id, $data);
    echo json_encode(["message" => "Task updated", "rows" => $rows]);
break;
```

The controller is updated to handle the response from the gateway and return a JSON custom message.

Delete the record in the database and return a 200 code

```
    public function delete(string $id) {
        $sql = "DELETE from task WHERE id=:id";
        $stmt = $this->conn->prepare($sql);
        $stmt->bindValue(":id", $id, PDO::PARAM_INT);
        $stmt->execute();
        return $stmt->rowCount();
    }
```

A delete SQL code block in the gateway php file.

```
case "DELETE":
    $rows = $this->gateway->delete($id);
    echo json_encode(["message" => "Task deleted", "rows" => $rows]);
    break;
```

Calling the delete method and returning a custom JSON success message.

API key authentication

Create a table to store user account data

```
CREATE TABLE user (
    id INT NOT NULL AUTO_INCREMENT,
    name VARCHAR(128) NOT NULL,
    username VARCHAR(128) NOT NULL,
    password_hash VARCHAR(255) NOT NULL,
    api_key VARCHAR(32) NOT NULL,
    PRIMARY KEY (id),
    UNIQUE (username),
    UNIQUE (api_key)
);
```

Create a user table that stores user passwords as hashed. The PHP documentation recommends 255 character length for hashed paswords.
https://www.php.net/manual/en/function.password-hash.php

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
1	id	int(11)			No	*None*		AUTO_INCREMENT
2	name	varchar(128)	utf8mb4_general_ci		No	*None*		
3	username	varchar(128)	utf8mb4_general_ci		No	*None*		
4	password_hash	varchar(255)	utf8mb4_general_ci		No	*None*		
5	api_key	varchar(32)	utf8mb4_general_ci		No	*None*		

Add a register page to insert a new user and generate an API key

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Register</title>
    <link rel="stylesheet" href="https://unpkg.com/@picocss/pico@latest/css/pico.min.css">
</head>
<body>
    <main class="container">
        <h1>Register</h1>
        <form method="post">
            <label for="name">Name
                <input name="name" id="name">
            </label>
            <label for="username">Username
                <input name="username" id="username">
            </label>
            <label for="password">Password
                <input type="password" name="password" id="password">
            </label>
            <button>Register</button>
        </form>
    </main>
</body>
</html>
```

To keep things simple a form to enter credentials that has no validation. Styling comes from a generic CSS template obtained via CDN. https://picocss.com

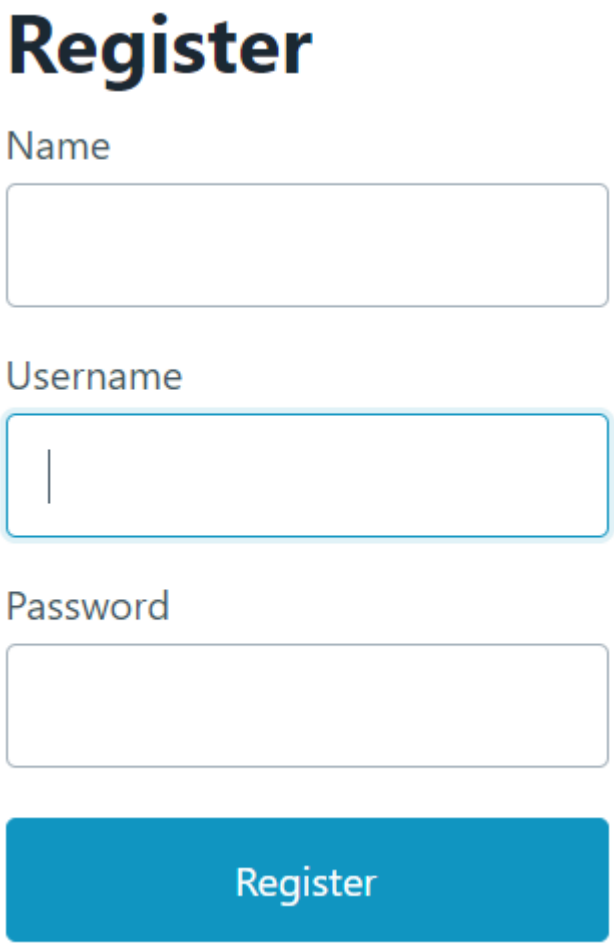
```php
<?php

require __DIR__ . "/vendor/autoload.php";

if ($_SERVER["REQUEST_METHOD"] === "POST") {

    $dotenv = Dotenv\Dotenv::createImmutable(__DIR__);
    $dotenv->load();

    $database = new Database($_ENV["DB_HOST"],
                             $_ENV["DB_NAME"],
                             $_ENV["DB_USER"],
                             $_ENV["DB_PASS"]);

    $conn = $database->getConnection();

    $sql = "INSERT INTO user (name, username, password_hash, api_key)
            VALUES (:name, :username, :password_hash, :api_key)";

    $stmt = $conn->prepare($sql);

    $password_hash = password_hash($_POST["password"], PASSWORD_DEFAULT);
    $api_key = bin2hex(random_bytes(16));

    $stmt->bindValue(":name", $_POST["name"], PDO::PARAM_STR);
    $stmt->bindValue(":username", $_POST["username"], PDO::PARAM_STR);
    $stmt->bindValue(":password_hash", $password_hash, PDO::PARAM_STR);
    $stmt->bindValue(":api_key", $api_key, PDO::PARAM_STR);

    $stmt->execute();

    echo "Thank you for registering. Your API key is ", $api_key;
    exit;
}

?>
```

At the top of the register.php file we add PHP code to make it work.

We can copy the database connection code from before and write an SQL query to insert the user credentials into the database.

The password is hashed before insertion into the database.

The API key is a random 16 byte number that we convert from binary to hey using the nin2hex php function. https://www.php.net/manual/en/function.bin2hex.php



Now when we enter user credentials and press the register button we get an api key.

Thank you for registering. Your API key is 22fe8690113a0dc42edc9a4e59f0d9b3

The user credentials get inserted into the user table with the password being hashed and a unique API key being generated and stored in the table.

id	name	username	password_hash	api_key
1	John Doe	johnDoe	$2y$10$/oYhMdqEBn1xp8oQUCNXm.CSrwKW5g609T.lioVhbim...	22fe8690113a0dc42edc9a4e59f0d9b3

Send the API key with the request: query string or request header

```php
<?php
declare(strict_types=1);

require dirname(__DIR__) . "/vendor/autoload.php";

$dotenv = Dotenv\Dotenv::createImmutable(dirname(__DIR__));
$dotenv->load();

set_error_handler("ErrorHandler::handleError");
set_exception_handler("ErrorHandler::handleException");

$path = parse_url($_SERVER["REQUEST_URI"], PHP_URL_PATH);
$parts = explode("/", $path);
$resource = $parts[4];
$id = $parts[5] ?? null;
if ($resource != "tasks") {
    http_response_code(404);
    exit;
}

$api_hey = $_GET["api-key"];
echo $api_key;
exit;
```

The API key can be sent in the query string of the URL or as a Header in the HTTP request.

In the front end controller (index.php) we can retrieve the API key from the query string using the get super global.

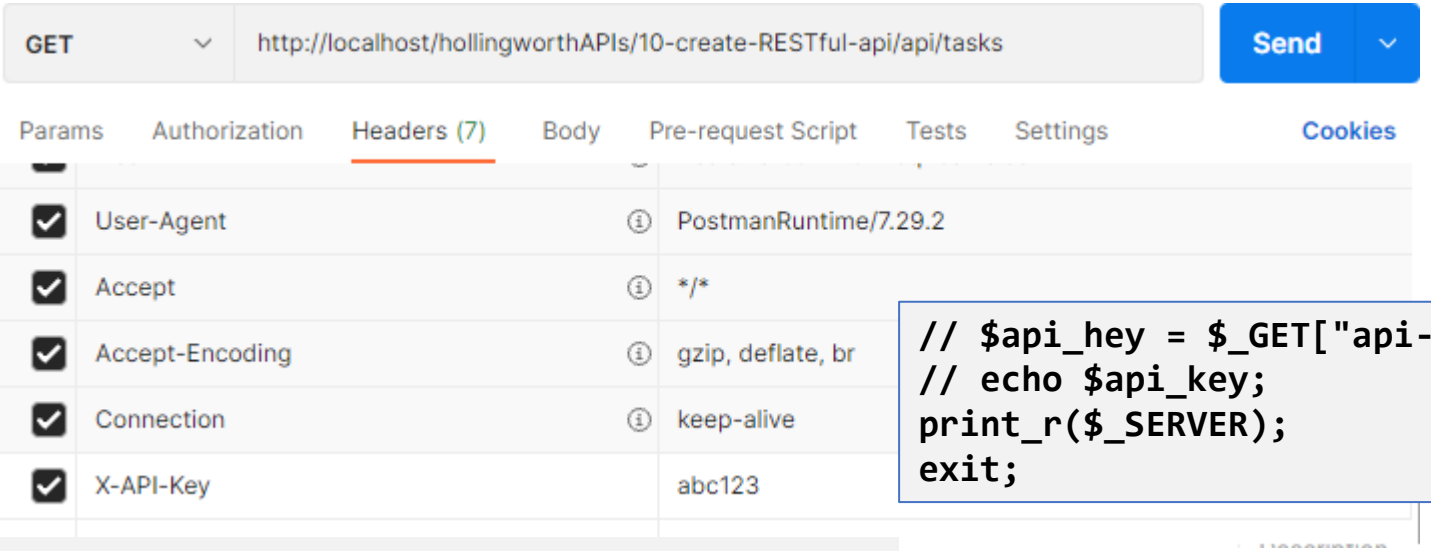
```
http://localhost/hollingworthAPIs/10-create-RESTful-api/api/tasks?api-key=abc123
```

```
abc123
```

Although this is sometimes used it is more common to send the API key in the header.

```
GET /something HTTP/1.1
X-API-Key: abcdef12345
```

Although there is no common api key header it is common to use X-API-Key. https://en.wikipedia.org/wiki/API_key



To send a custom header from postman we just ass it in under the header tab.

```
// $api_hey = $_GET["api-key"];
// echo $api_key;
print_r($_SERVER);
exit;
```

HTTP headers in PHP can be found in the superglobal of $_SERVER.

```
Array(
[REDIRECT_MIBDIRS] => C:/xampp3/php/extras/mibs
[REDIRECT_MYSQL_HOME] => \xampp\mysql\bin
[REDIRECT_OPENSSL_CONF] => C:/xampp3/apache/bin/openssl.cnf
[REDIRECT_PHP_PEAR_SYSCONF_DIR] => \xampp\php
[REDIRECT_PHPRC] => \xampp\php
[REDIRECT_TMP] => \xampp\tmp
[REDIRECT_STATUS] => 200
[MIBDIRS] => C:/xampp3/php/extras/mibs
[MYSQL_HOME] => \xampp\mysql\bin
[OPENSSL_CONF] => C:/xampp3/apache/bin/openssl.cnf
[PHP_PEAR_SYSCONF_DIR] => \xampp\php
[PHPRC] => \xampp\php
[TMP] => \xampp\tmp
[HTTP_X_API_KEY] => abc123
[HTTP_USER_AGENT] => PostmanRuntime/7.29.2
[HTTP_ACCEPT] => */*
[HTTP_POSTMAN_TOKEN] => 483f47ca-d9ed-4143-abba-6396a776fa83
[HTTP_HOST] => localhost
. . .
)
```

Note that in the $_SERVER superglobal array, values that pertain to the HTTP headers are prefixed with HTTP. Note also that hyphens are converted to underscore and all letters are capitalised.

X-API-Key = [HTTP_X_API_KEY]

Check the API Key is present in the request and return a 400 if not

```
$api_hey = $_SERVER["HTTP_X_API_KEY"];
echo $api_key;
exit;
```

If the API key is sent in the header this code will return a 200 status code. If the API key is not sent in the header then it will return a status code of 500 and the below error message.

```
{"code":0,"message":"Undefined array key
\"HTTP_X_API_KEY\"","file":"C:\\xampp3\\htdocs\\hollingworthAPIs\\10-create-RESTful-
api\\api\\index.php","line":21}
```

```
if(empty($_SERVER["HTTP_X_API_KEY"])) {
    http_response_code(400);
    echo json_encode(["message" => "missing API key"]);
    exit;
}
$api_key = $_SERVER["HTTP_X_API_KEY"];
echo $api_key;
```

This code can handle a missing API key from the header and return a custom JSON response.

```
{"message":"missing API key"}
```

Create a table data gateway class for the user table

```php
<?php

class UserGateway {
    private PDO $conn;

    public function __construct(Database $database) {
        $this->conn = $database->getConnection();
    }

    public function getByAPIKey(string $key) : array | false
{
        $sql = "SELECT * FROM user WHERE api_key=:api_key";
        $stmt = $this->conn->prepare($sql);
        $stmt->bindValue(":api_key", $key, PDO::PARAM_STR);
        $stmt->execute();
        return $stmt->fetch(PDO::FETCH_ASSOC);
    }
}
```

A UserGateway.php class is created in the src folder that will take a $key and return user data that match the $key.

```php
<?php
declare(strict_types=1);
require dirname(__DIR__) . "/vendor/autoload.php";

$dotenv = Dotenv\Dotenv::createImmutable(dirname(__DIR__));
$dotenv->load();

set_error_handler("ErrorHandler::handleError");
set_exception_handler("ErrorHandler::handleException");

$path = parse_url($_SERVER["REQUEST_URI"], PHP_URL_PATH);
$parts = explode("/", $path);
$resource = $parts[4];
$id = $parts[5] ?? null;
if ($resource != "tasks") {
    http_response_code(404);
    exit;
}

if(empty($_SERVER["HTTP_X_API_KEY"])) {
    http_response_code(400);
    echo json_encode(["message" => "missing API key"]);
    exit;
}
$api_key = $_SERVER["HTTP_X_API_KEY"];

$database = new Database($_ENV["DB_HOST"], $_ENV["DB_NAME"], $_ENV["DB_USER"], $_ENV["DB_PASS"]);
$user_gateway = new UserGateway($database);

Header("Content-Type: application/JSON charset=UTF-8");

$taskGateway = new TaskGateway($database);
$controller = new TaskController($taskGateway);
$controller->processRequest($_SERVER['REQUEST_METHOD'], $id);
?>
```

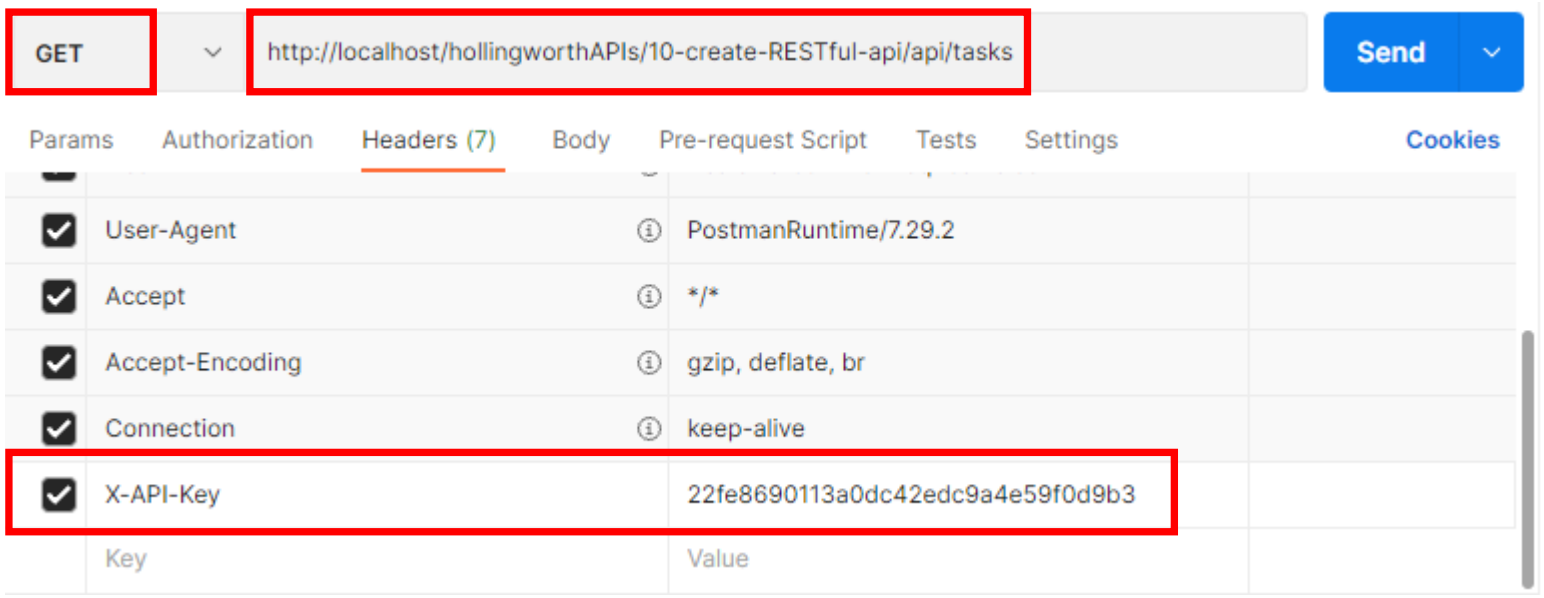
We can use the userGateway class by instantiating the database class then after, the userGateway class.

Authenticate the API key and return a 401 status code if invalid

```
$database = new Database($_ENV["DB_HOST"], $_ENV["DB_NAME"], $_ENV["DB_USER"], $_ENV["DB_PASS"]);
$user_gateway = new UserGateway($database);
if ($user_gateway->getByAPIkey($api_key) === false) {
    http_response_code(401);
    echo json_encode(["message" => "invalid API key"]);
    exit;
};
```

Now if I send an ALI key that is not in the database I get a 401 unothorised status code and the custom JSON message and the script exits here.

```
{"message":"invalid API key"}
```



If I do send an API key that is in the database then the API returns a 200 status and completes the API request.

Refactor front controller to a bootstrap file and auth class

At the moment all requests are going through the index.php front controller. If we add anymore files that serve as endpoints we will want to do a lot of the things that we are doing here so we can extract this code out to a common bootstrap file.

```php
<?php

require dirname(__DIR__) . "/vendor/autoload.php";

set_error_handler("ErrorHandler::handleError");
set_exception_handler("ErrorHandler::handleException");

$dotenv = Dotenv\Dotenv::createImmutable(dirname(__DIR__));
$dotenv->load();

Header("Content-Type: application/JSON charset=UTF-8");
```

These lines of code can be extracted from the index.php file and placed in a bootstrap.php file in the api folder.

```php
<?php
declare(strict_types=1);
require dirname(__DIR__) . "/api/bootstrap.php";
```

We need to require the bootstrap file in the index.php file.

```php
<?php

class Auth {

    public function __construct(private UserGateway $user_gateway) {}

    public function authenticateAPIKey(): bool {
        if(empty($_SERVER["HTTP_X_API_KEY"])) {
            http_response_code(400);
            echo json_encode(["message" => "missing API key"]);
            return false;
        }
        $api_key = $_SERVER["HTTP_X_API_KEY"];

        if ($this->user_gateway->getByAPIkey($api_key) === false) {
            http_response_code(401);
            echo json_encode(["message" => "invalid API key"]);
            return false;
        };
        return true;
    }
}
```

In the src folder a new class is created to specifically handle authentication of the API key. It has a constructor that instantiates the user gateway class.

The auth class handles a missing API key scenario or an invalid API key and this code is cut and copied directly from the index.php file.

Note that when we call the userGateway class we now use $this keyword because we are instantiating the class in the constructor.

The authenticate API key method returns Boolean false if authentication falis and Boolean true if authentication passes.

```php
<?php
declare(strict_types=1);
require dirname(__DIR__) . "/api/bootstrap.php";

$path = parse_url($_SERVER["REQUEST_URI"], PHP_URL_PATH);
$parts = explode("/", $path);
$resource = $parts[4];
$id = $parts[5] ?? null;
if ($resource != "tasks") {
    http_response_code(404);
    exit;
}

$database = new Database($_ENV["DB_HOST"], $_ENV["DB_NAME"],
$_ENV["DB_USER"], $_ENV["DB_PASS"]);
$user_gateway = new UserGateway($database);
$auth = new Auth($user_gateway);
if (!$auth->authenticateAPIKey()) {
    exit;
}

$taskGateway = new TaskGateway($database);
$controller = new TaskController($taskGateway);
$controller->processRequest($_SERVER['REQUEST_METHOD'], $id);
?>
```

In the refactored index.php front end controller we just call the auth class authenticateAPI key method and if it returns false, exit.

Add a foreign key relationship to link task records to user records

The tasks table is modified to add another column called user_id which is indexed.

```
ALTER TABLE task
ADD user_id INT NOT NULL
ADD INDEX (user_id);
```

id	name	username	password_hash	api_key
1	John Doe	johnDoe	$2y$10$/oYhMdqEBn1xp8oQUCNXm.CSrwKW5g609T.lioVhbim...	22fe8690113a0dc42edc9a4e59f0d9b3

Currently there is one user in the users table with an ID of 1 so in the tasks I have set the user_id column to 1 for all rows. This means that these tasks were created by user with id of 1.

id	name	priority	is_completed	user_id
1	Buy new shoes	1	1	1
2	Renew passport	2	0	1
3	Paint wall	*NULL*	1	1
4	A new post	3	0	1
5	task 2	6	0	1
6	fix code bug	3	1	1

```
ALTER TABLE task
ADD FOREIGN KEY (user_id)
REFERENCES user(id)
ON DELETE CASCADE ON UPDATE CASCADE;
```

The tasks table is updated to add a foreign key so that user_id references ID of users table and that any changes or deletions will cascade down. i.e. if a user is deleted then all their tasks will also be deleted.

Retrieve the ID of the authenticated user when authenticating

In the Auth class we define a user_id variable as private.

A variable of user contains the user credentials array and now if this is empty then we can return an invalid API key custom message.

We set the value of the private variable for user_id.

We have a public method that returns the User_id.

```php
<?php

class Auth {

    private int $user_id;
    public function __construct(private UserGateway $user_gateway) {}

    public function authenticateAPIKey(): bool {
        if(empty($_SERVER["HTTP_X_API_KEY"])) {
            http_response_code(400);
            echo json_encode(["message" => "missing API key"]);
            return false;
        }
        $api_key = $_SERVER["HTTP_X_API_KEY"];
        $user = $this->user_gateway->getByAPIkey($api_key);
        if ($user === false) {
            http_response_code(401);
            echo json_encode(["message" => "invalid API key"]);
            return false;
        };
        $this->user_id = $user["id"];
        return true;
    }

    public function getUserId(): int {
        return $this->user_id;
    }
}
```

```php
<?php
declare(strict_types=1);
require dirname(__DIR__) . "/api/bootstrap.php";

$path = parse_url($_SERVER["REQUEST_URI"], PHP_URL_PATH);
$parts = explode("/", $path);
$resource = $parts[4];
$id = $parts[5] ?? null;
if ($resource != "tasks") {
    http_response_code(404);
    exit;
}

$database = new Database($_ENV["DB_HOST"],
$_ENV["DB_NAME"], $_ENV["DB_USER"], $_ENV["DB_PASS"]);
$user_gateway = new UserGateway($database);
$auth = new Auth($user_gateway);
if (!$auth->authenticateAPIKey()) {
    exit;
}

$user_id = $auth->getUserId();
var_dump($user_id);
exit;
```

In the front end controller (index.php) after authenticating the user we can get the user_id and temporarily output it.

When I send an API request to ***http://localhost/hollingworthAPIs/10-create-RESTful-api/api/tasks*** with the correct API key defined in the header I get back the User_id.

Now using the register page in the browser I can add another user.
http://localhost/hollingworthAPIs/10-create-RESTful-api/register.php

Thank you for registering.
Your API key is
0a37d180f691bc45600e4a1266986293

I can test again sending the API key for Mary Smith and I get her user ID back.

id	name	username	password_hash	api_key
1	John Doe	johnDoe	$2y$10$/oYhMdqEBn1xp8oQUCNXm.CSrwKW5g609T.lioVhbim...	22fe8690113a0dc42edc9a4e59f0d9b3
3	Mary Smith	MarySmith	$2y$10$GVE5QbpyicypG4nlm6MgJumtkmiwOZ2lyfhxKlnJORI...	0a37d180f691bc45600e4a1266986293

Restrict tasks index endpoint to only show the authenticated user's tasks

```php
<?php
declare(strict_types=1);
require dirname(__DIR__) . "/api/bootstrap.php";

$path = parse_url($_SERVER["REQUEST_URI"], PHP_URL_PATH);
$parts = explode("/", $path);
$resource = $parts[4];
$id = $parts[5] ?? null;
if ($resource != "tasks") {
    http_response_code(404);
    exit;
}

$database = new Database($_ENV["DB_HOST"], $_ENV["DB_NAME"],
$_ENV["DB_USER"], $_ENV["DB_PASS"]);
$user_gateway = new UserGateway($database);
$auth = new Auth($user_gateway);
if (!$auth->authenticateAPIKey()) {
    exit;
}

$user_id = $auth->getUserId();
$taskGateway = new TaskGateway($database);
$controller = new TaskController($taskGateway, $user_id);
$controller->processRequest($_SERVER['REQUEST_METHOD'], $id);
?>
```

In the front end controller (index.php) we pass in the $user_id to the tasks controller.

```php
<?php

class TaskController {
    public function __construct(
        private TaskGateway $gateway,
        private int $user_id
    ) {}
```

In the task controller we declare the user_id variable in the constructor.

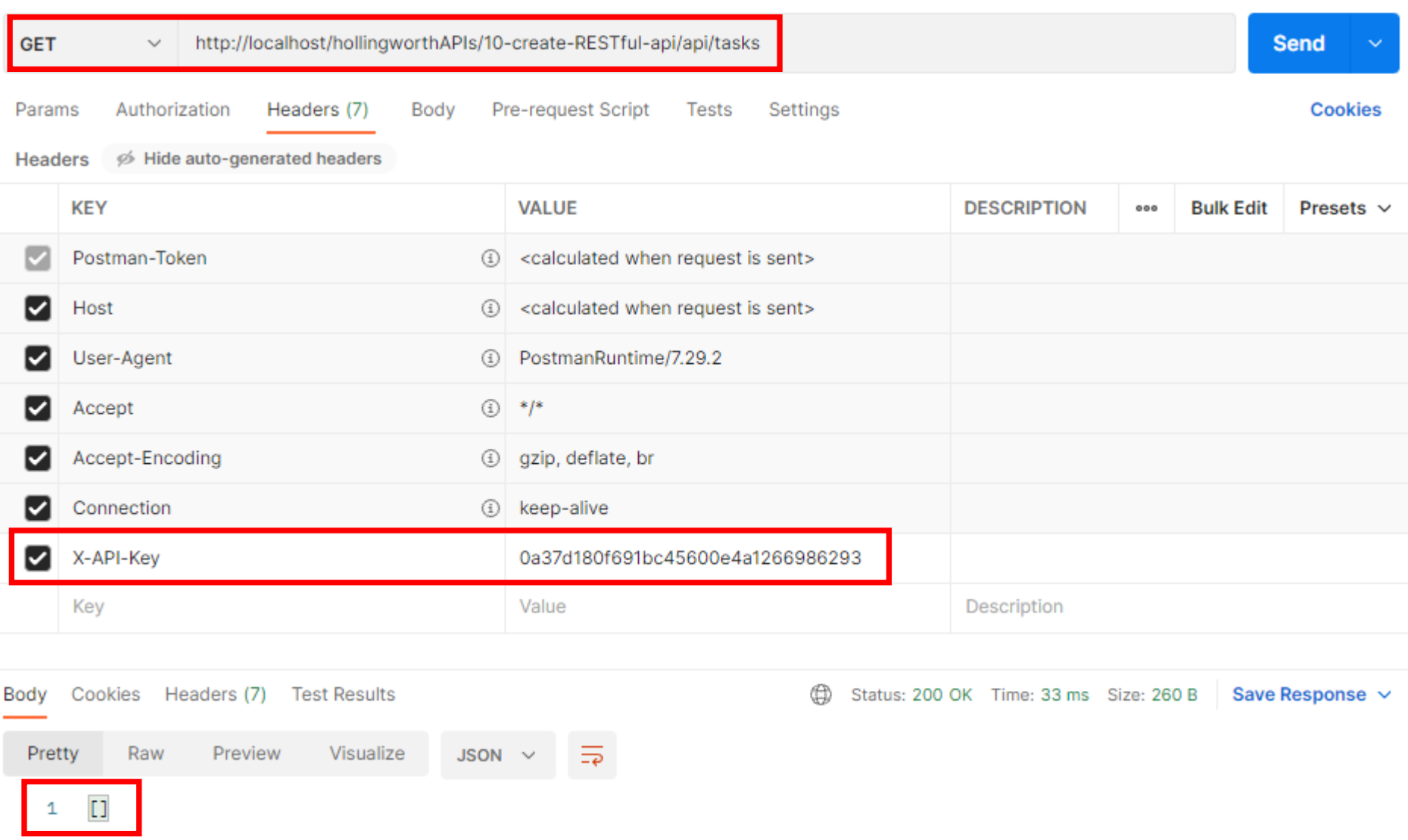
```
public function getAllForUser(int $user_id): array {
    $sql = "SELECT * FROM task WHERE user_id=:user_id ORDER BY name";
    $stmt = $this->conn->prepare($sql);
    $stmt->bindValue(":user_id", $user_id, PDO::PARAM_INT);
    $stmt->execute();
    $data = [];
    while ($row = $stmt->fetch(PDO::FETCH_ASSOC)) {
        $row['is_completed'] = (bool) ['is_completed'];
        $data[] = $row;
    }
    return $data;
}
```

In the tasks gateway the method for getAll is adjusted to be getAllForUser so that it finds all rows of tasks created by the user_id passed in.

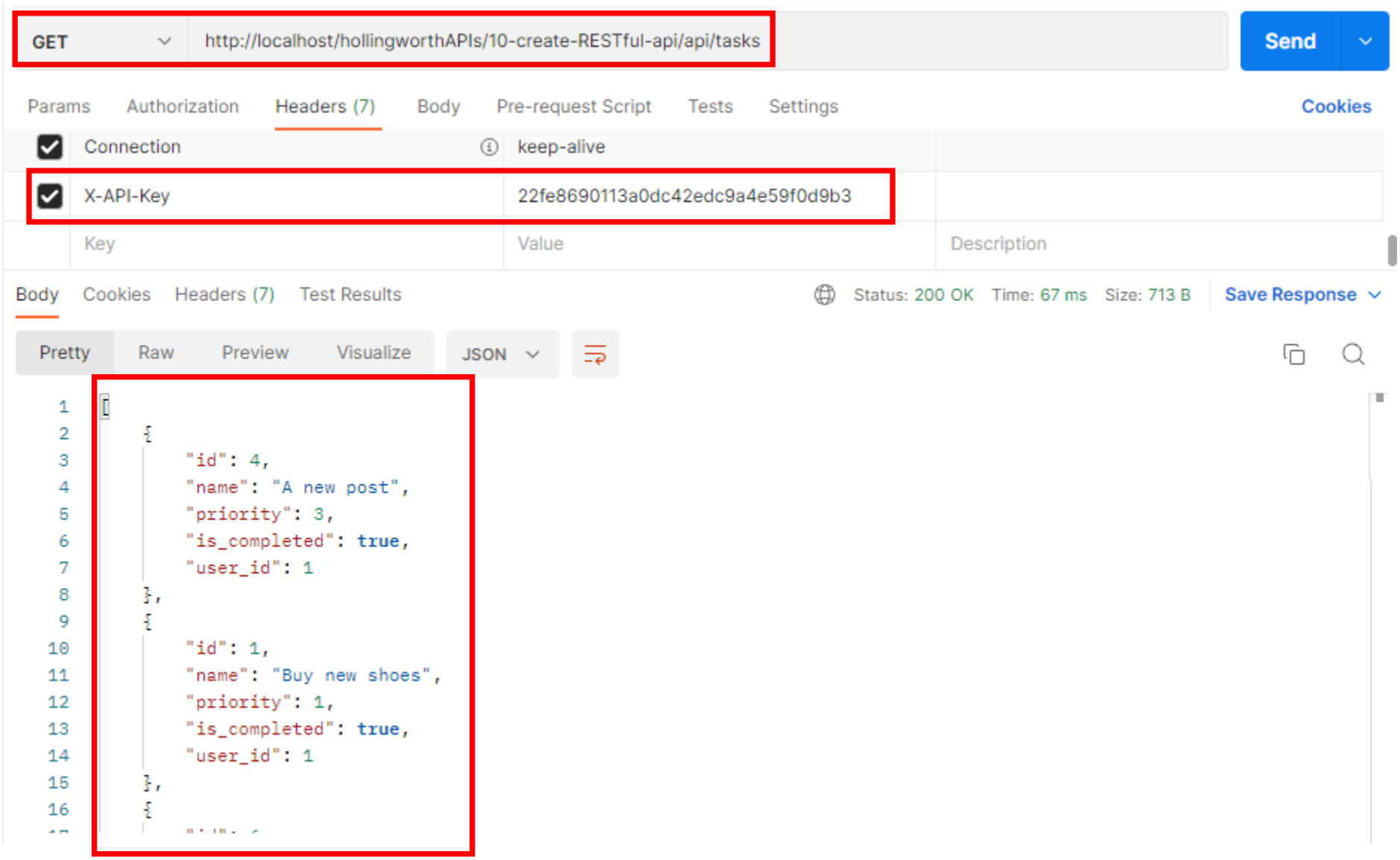
```
<?php

class TaskController {
    public function __construct(private TaskGateway $gateway, private int $user_id) {}

    public function processRequest(string $method, ?string $id): void  {
        if ($id === null) {

            if ($method == "GET") {
                echo json_encode($this->gateway->getAllForUser($this->user_id));
                // echo "index";
            } elseif ($method == "POST") {
. . .
```

In the task controller I cnage the call to getAllForUser and pass in the User_id.

In postman if I send an API request with Mary's API key in the header I get back an empty array of tasks because she has not tasks in the task table.



In postman if I send an API request with John's API key in the header I get back a JSON object with all of his tasks.



Restrict the rest of the task Endpoints to the authenticated user's tasks

```
public function getForUser(int $user_id, string $id): array | false {
    $sql = "SELECT * FROM task WHERE id=:id AND user_id=:user_id";
    $stmt = $this->conn->prepare($sql);
    $stmt->bindValue(":id", $id, PDO::PARAM_INT);
    $stmt->bindValue(":user_id", $user_id, PDO::PARAM_INT);
    $stmt->Execute();
    $data = $stmt->fetch(PDO::FETCH_ASSOC);
    if ($data !== false) {
        $data['is_completed'] = (bool) ['is_completed'];
    }
    return $data;
}
```

The rest of the methods in the tasks gateway are modified to take a user_id parameter and execute the SQL for that user_id

```
public function createForUser(int $user_id, array $data): string {
     $sql = "INSERT INTO task (name, priority, is_completed, user_id)
                VALUES (:name, :priority, :is_completed, :user_id)";
     $stmt = $this->conn->prepare($sql);
     $stmt->bindValue(":name", $data["name"], PDO::PARAM_STR);
     $stmt->bindValue(":user_id", $user_id, PDO::PARAM_INT);
     if (empty($data["priority"])) {
        $stmt->bindValue(":priority", null, PDO::PARAM_NULL);
     } else {
         $stmt->bindValue(":priority", $data["priority"], PDO::PARAM_INT);
     }
     $stmt->bindValue(":is_completed", $data["is_completed"] ?? false, PDO::PARAM_BOOL);
     $stmt->execute();
     return $this->conn->lastInsertId();
    }
```

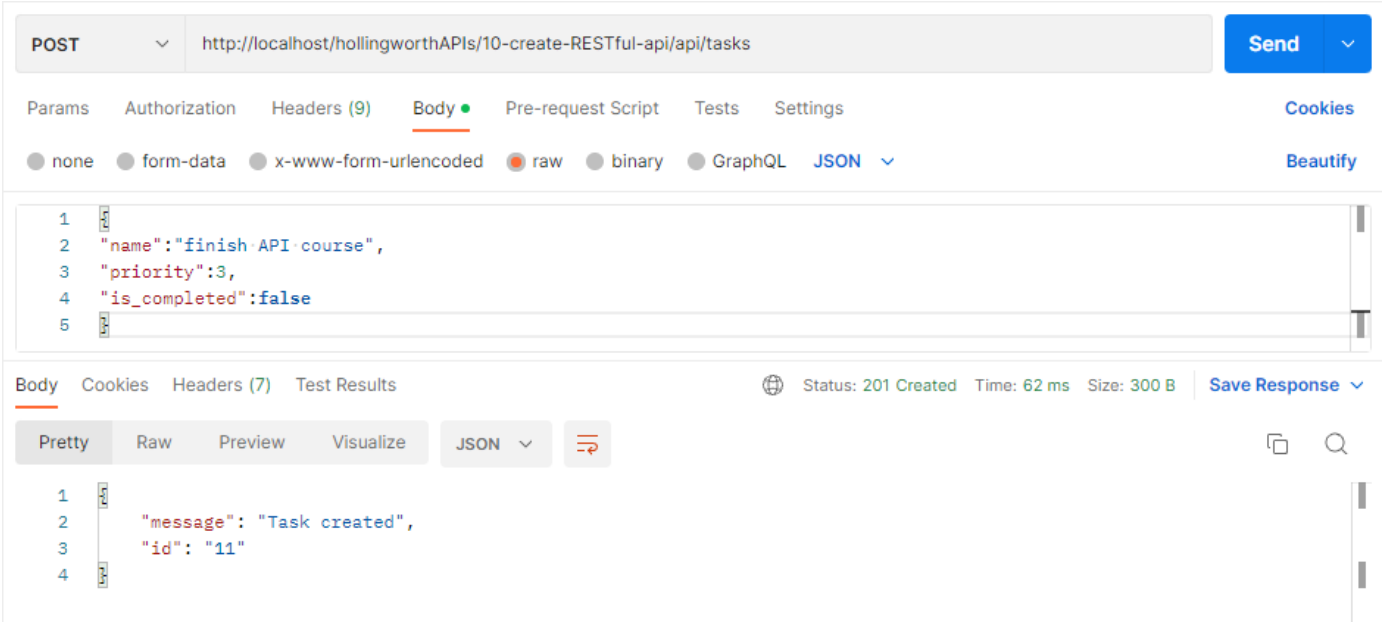
```php
public function updateForUser(int $user_id,string $id, array $data) {
    $fields = [];
    if ( ! empty($data["name"])) {
           $fields["name"] = [$data["name"], PDO::PARAM_STR];
    }
    if (array_key_exists("priority", $data)) {
        $fields["priority"] = [$data["priority"], $data["priority"] === null ? PDO::PARAM_NULL : PDO::PARAM_INT];
    }
    if (array_key_exists("is_completed", $data)) {
        $fields["is_completed"] = [$data["is_completed"], PDO::PARAM_BOOL];
    }

    if (empty($fields)) {
        return 0;
    } else {
        $sets = array_map(function($value) {
            return "$value = :$value";
        }, array_keys($fields));

        $sql = "UPDATE task"
             . " SET " . implode(", ", $sets)
             . " WHERE id = :id AND user_id=:user_id";

        $stmt = $this->conn->prepare($sql);
        $stmt->bindValue(":id", $id, PDO::PARAM_INT);
        $stmt->bindValue(":user_id", $user_id, PDO::PARAM_INT);
        foreach ($fields as $name => $values) {
            $stmt->bindValue(":$name", $values[0], $values[1]);
        }
        $stmt->execute();
        return $stmt->rowCount();
    }
}
```

```php
public function deleteForUser(int $user_id, string $id): int {
    $sql = "DELETE from task WHERE id=:id AND user_id=:user_id";
    $stmt = $this->conn->prepare($sql);
    $stmt->bindValue(":id", $id, PDO::PARAM_INT);
    $stmt->bindValue(":user_id", $user_id, PDO::PARAM_INT);
    $stmt->execute();
    return $stmt->rowCount();
}
```

```php
<?php

class TaskController {
    public function __construct(private TaskGateway $gateway, private int $user_id) {}

    public function processRequest(string $method, ?string $id): void  {
        if ($id === null) {

            if ($method == "GET") {
                echo json_encode($this->gateway->getAllForUser($this->user_id));
                // echo "index";
            } elseif ($method == "POST") {
                // print_r($_POST);
                $data = (array) json_decode(file_get_contents("php://input"), true);
                $errors = $this->getValidationErrors($data);
                if (!empty($errors)) {
                    $this->respondUnprocessableEntity($errors);
                    return;
                }
                $id = $this->gateway->createForUser($this->user_id, $data);
                $this->respondCreated($id);
                // echo "create";
            }
            else {
                $this->respondMethodNotAllowed("GET, POST");
            }
        }
        else {
            $task = $this->gateway->getForUser($this->user_id, $id);
            if ($task === false) {
                $this->respondNotFound($id);
                return;
            }
// . . . Continued on next page
```

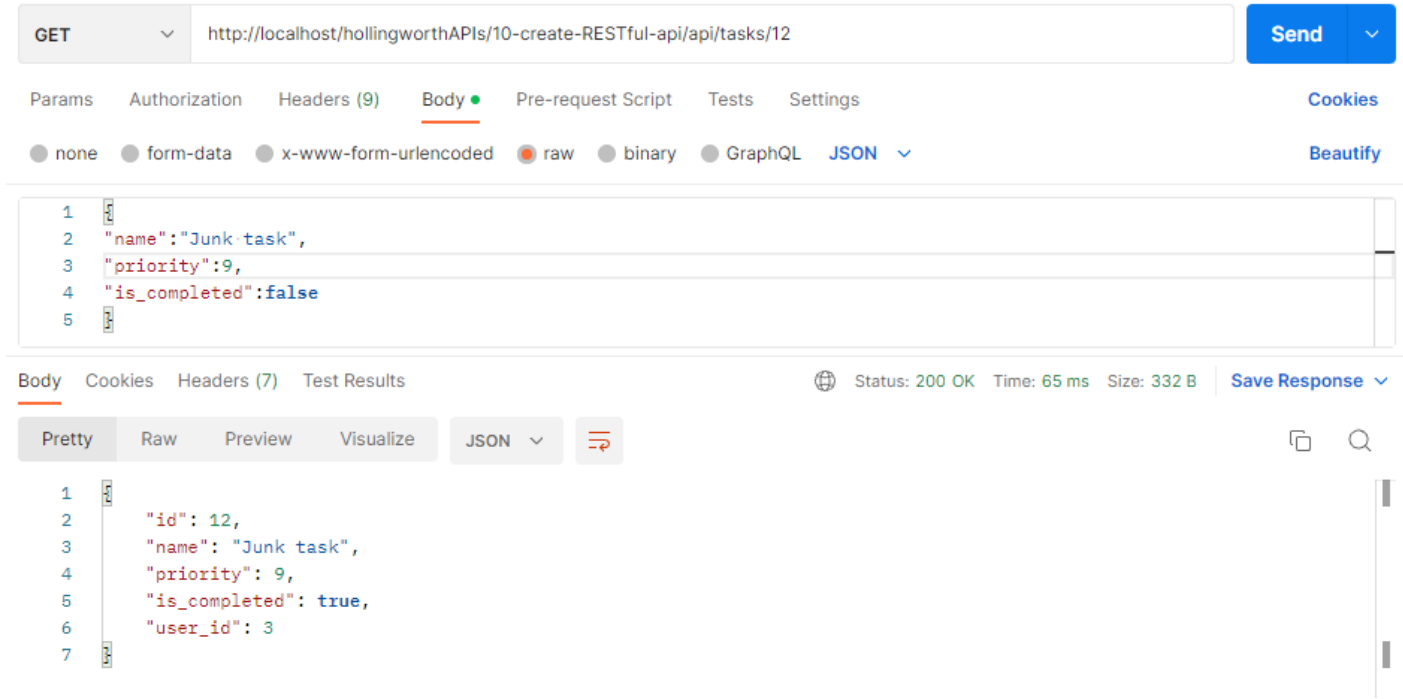
The method calls in the tasks controller are modified to the new names and pass in the user_id argument.

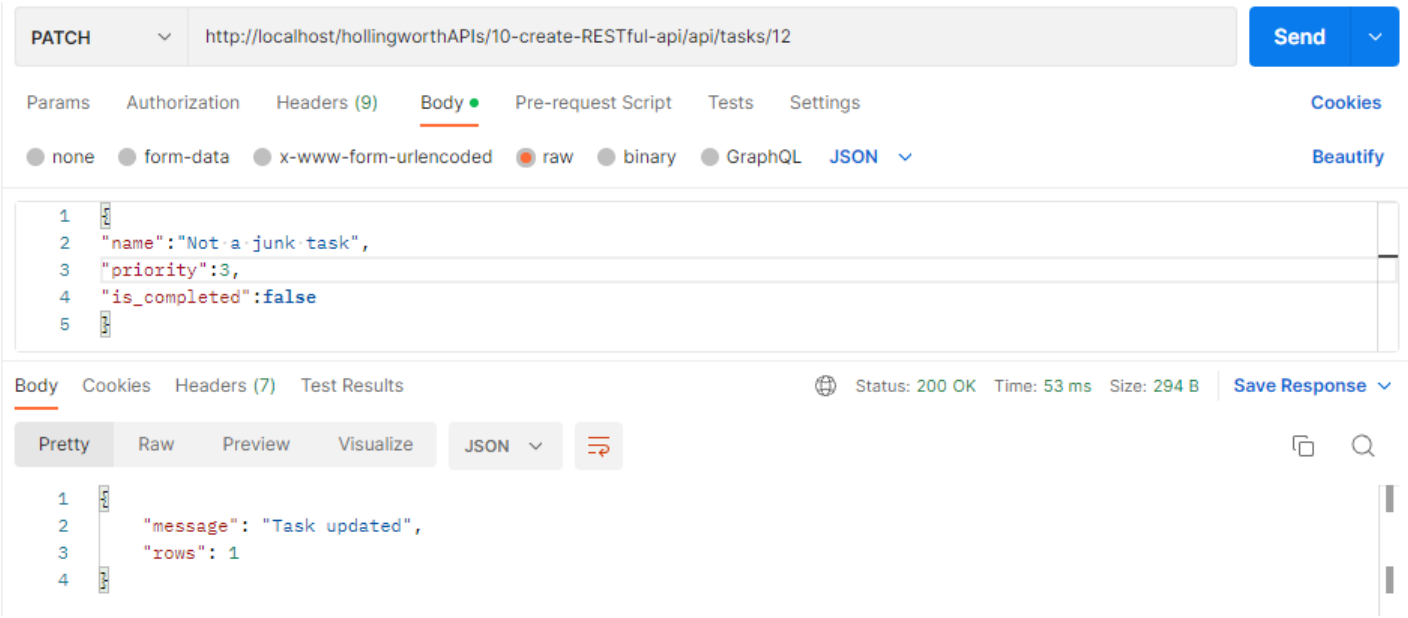
```
// . . . Continuation from previous page
            switch ($method) {
                case "GET":
                    echo json_encode($task);
                    // echo "show $id";
                break;
                case "PATCH":
                    $data = json_decode(file_get_contents("php://input"), true);
                    $errors = $this->getValidationErrors($data, false);
                    if (!empty($errors)) {
                        $this->respondUnprocessableEntity($errors);
                        return;
                    }
                    $rows = $this->gateway->updateForUser($this->user_id, $id, $data);
                    echo json_encode(["message" => "Task updated", "rows" => $rows]);
                    // echo "update $id";
                break;

                case "DELETE":
                    $rows = $this->gateway->deleteForUser($this->user_id, $id);
                    echo json_encode(["message" => "Task deleted", "rows" => $rows]);
                    // echo "delete $id";
                break;

                default:
                    $this->respondMethodNotAllowed("GET, PATCH, DELETE");
            }
        }
    }
. . .
```



Using Mary's API Key I can insert a new task into the database

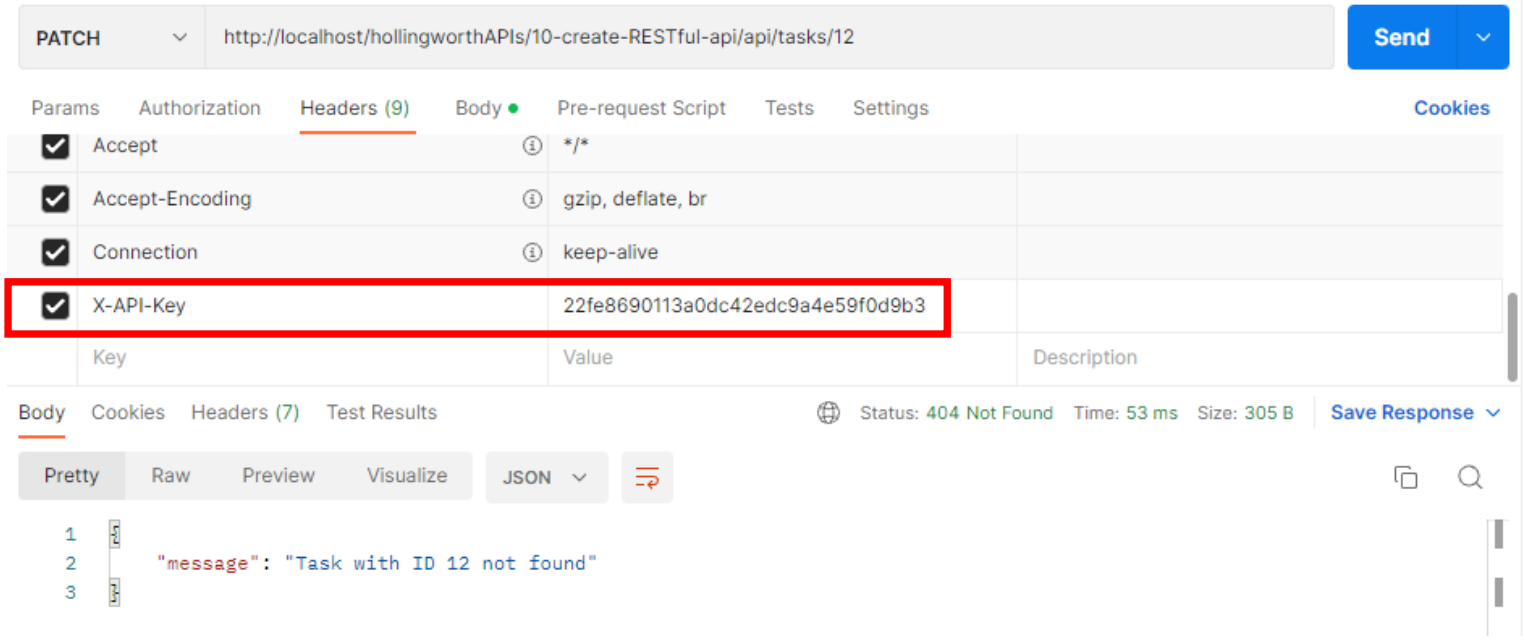
I have added another task using Mary's API Key and I can get it by task id.

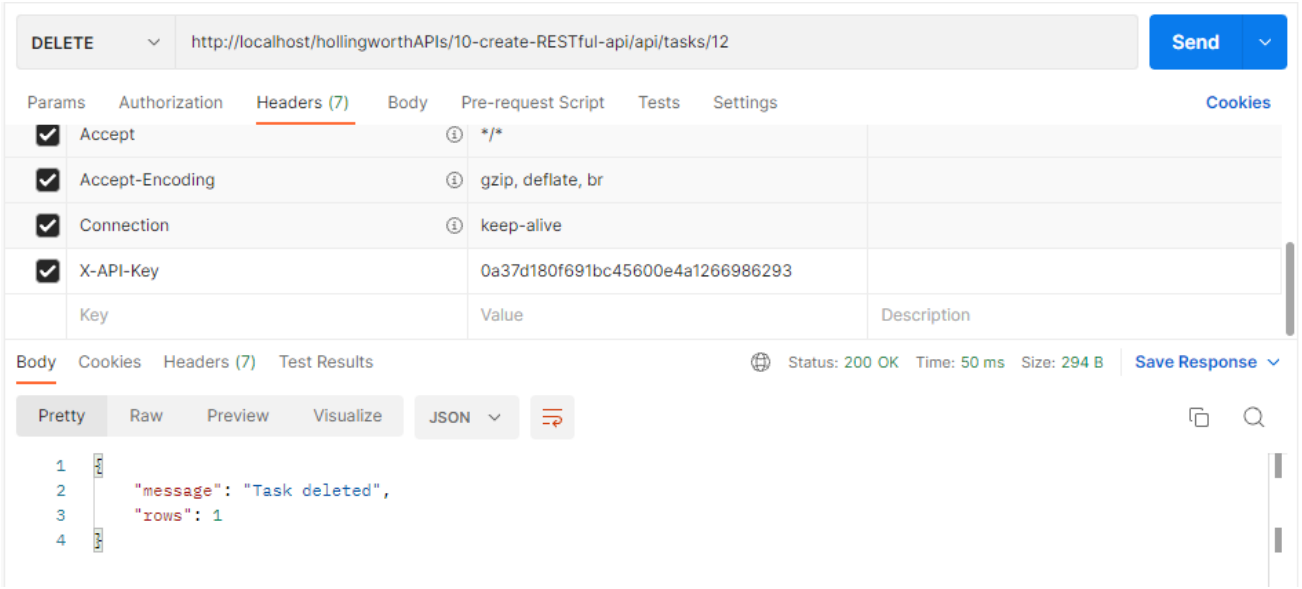




I can update the task using PATCH api call with Mary's API Key.

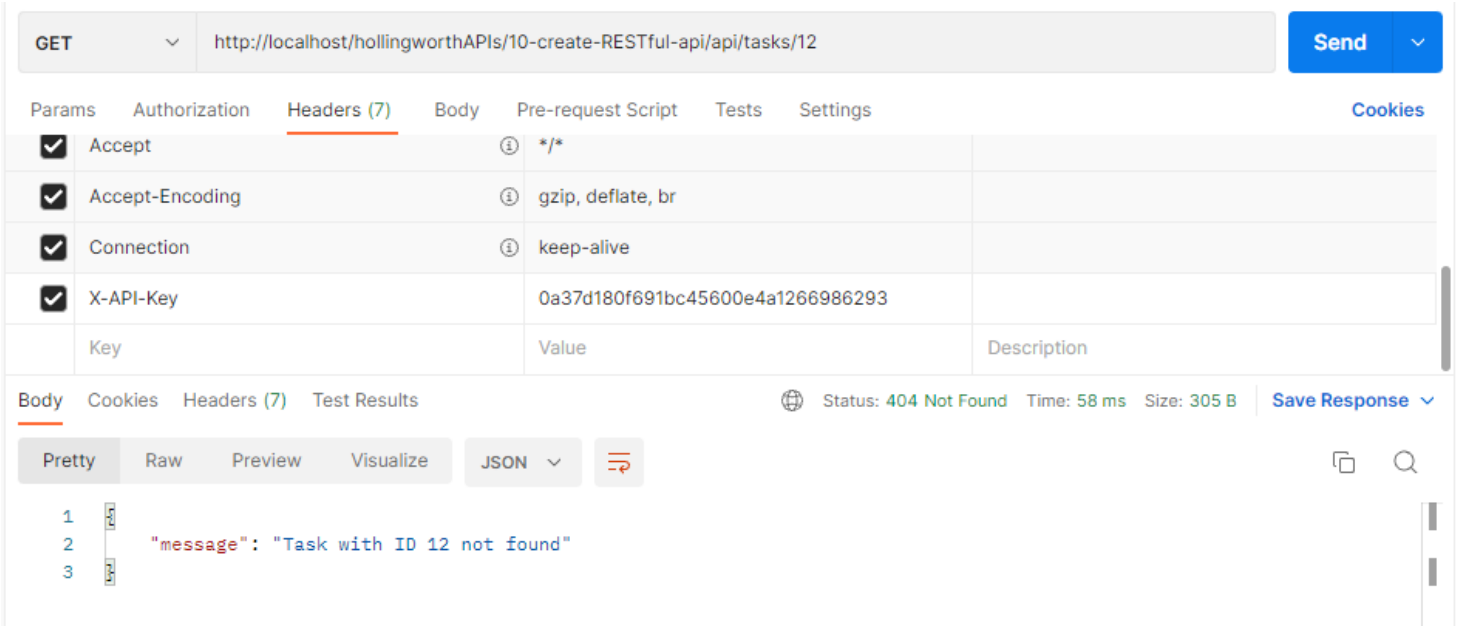
If I try and update the task using PATCH or erase the task using DELETE with Johns's API Key I get a 404 not found because the task does not belong to that user.





I can delete the task with the correct user API key.

Now if I try and get that record I get 404 because we just deleted it!



Cache database connection to avoid multiple connections in the same request

```php
<?php

class Database {

    private ?PDO $conn = null;

    public function __construct(
        private string $host,
        private string $name,
        private string $user,
        private string $password
    ) {
    }

    public function getConnection(): PDO {
        if ($this->conn === null) {
            $dsn = "mysql:host={$this->host};dbname={$this->name};charset=utf8";

            $this->conn = new PDO($dsn, $this->user, $this->password, [
                PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION,
                PDO::ATTR_STRINGIFY_FETCHES => false,
                PDO::ATTR_EMULATE_PREPARES => false
            ]);
        }
        return $this->conn;
    }
}
```

In the taskGateway and the UserGateway I am using a database connection which means two database connections will be made. To avoid this we can store the database connection in the database class.

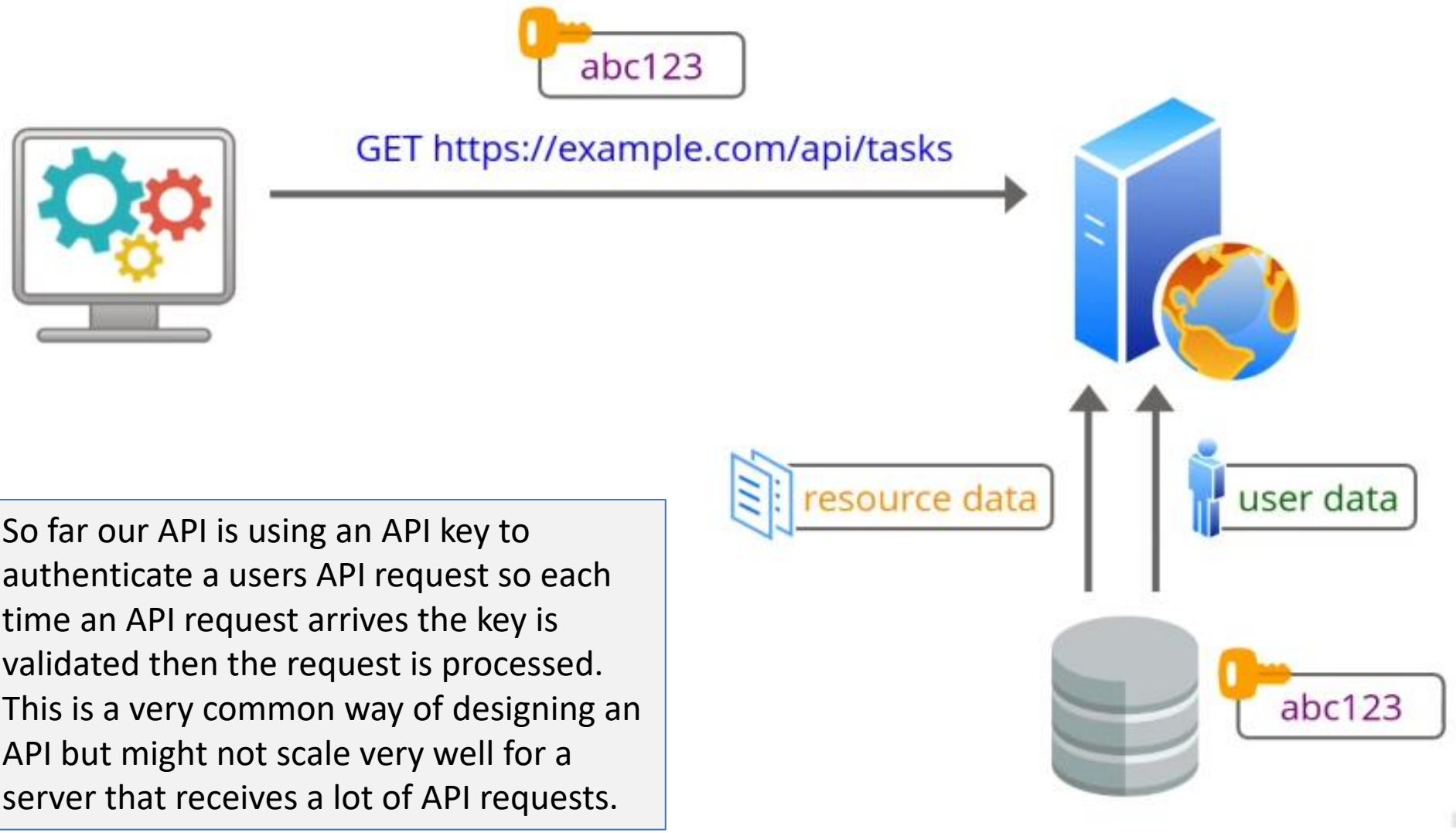
We first declare a private variable for the connection and set it to null. Because we are using type declarations we also need to set this to nullable.

Now in the get connection we first check if the $conn variable is null and assign the connection to it if it is null

We return the $conn. Note that if it is already connected then the code will skip the if statement and just return the $conn stored in the private variable.

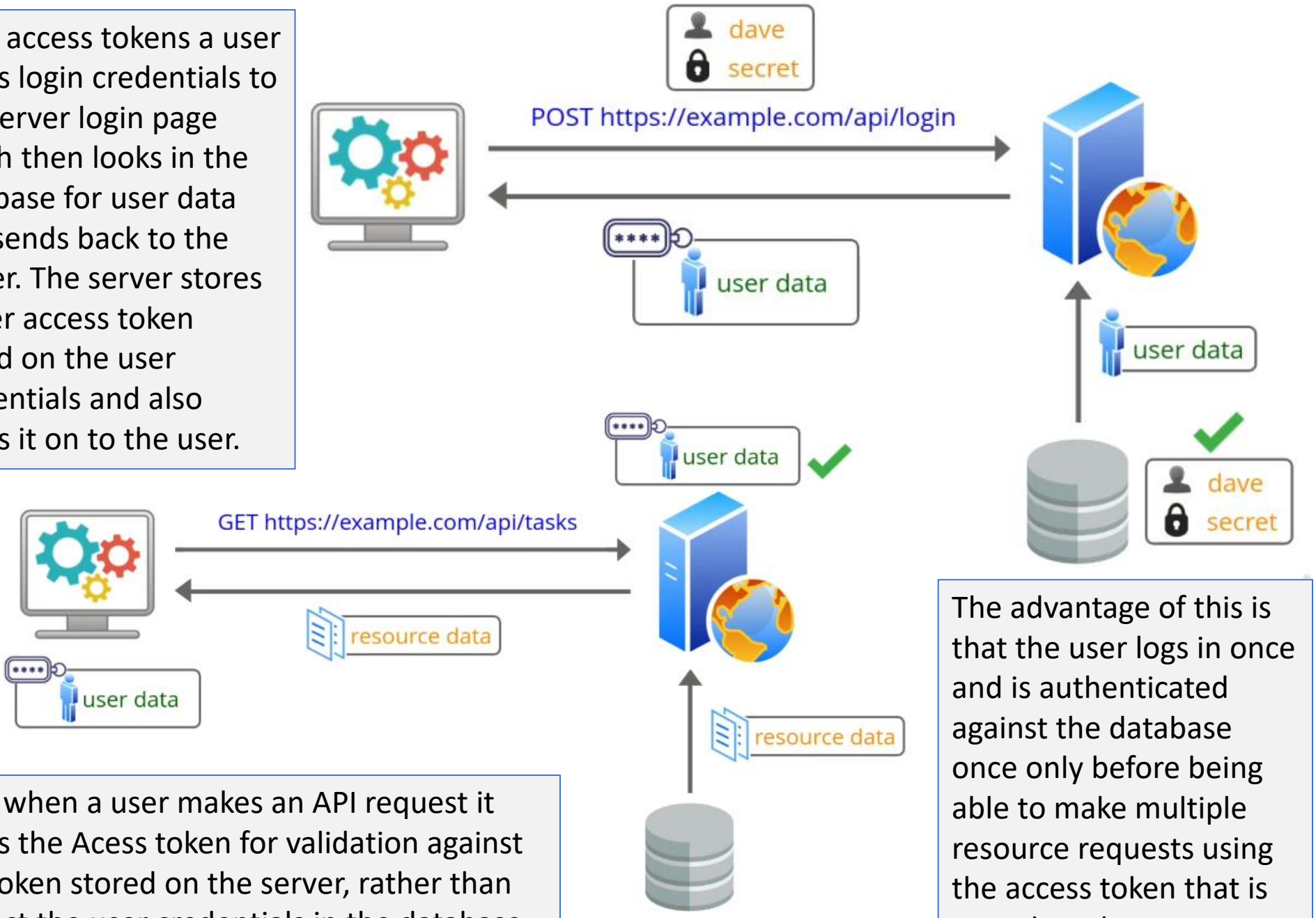
An Introduction to Authentication Using Access Tokens

An Introduction to Authentication Using Access Tokens



So far our API is using an API key to authenticate a users API request so each time an API request arrives the key is validated then the request is processed. This is a very common way of designing an API but might not scale very well for a server that receives a lot of API requests.

With access tokens a user sends login credentials to the server login page which then looks in the database for user data and sends back to the server. The server stores a user access token based on the user credentials and also sends it on to the user.



Now when a user makes an API request it sends the Acess token for validation against the token stored on the server, rather than against the user credentials in the database.

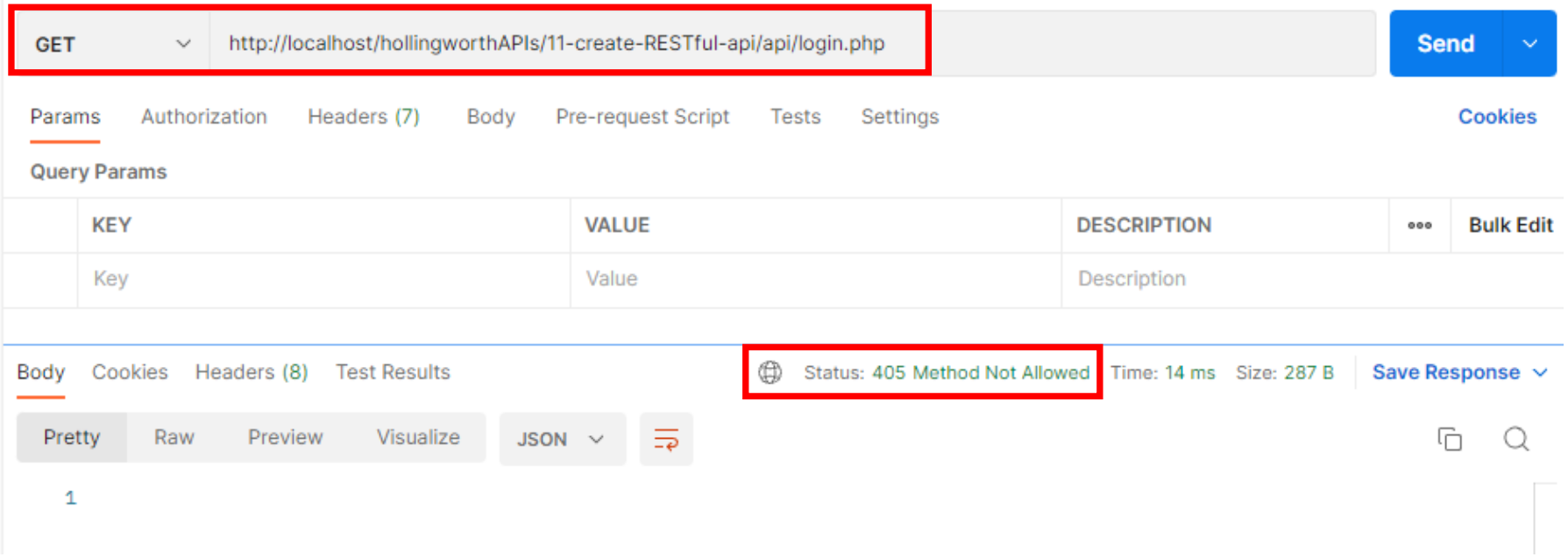
The advantage of this is that the user logs in once and is authenticated against the database once only before being able to make multiple resource requests using the access token that is stored on the server.

Create the login script and return a 400 if username and password are missing

```php
<?php
declare(strict_types=1);
require dirname(__DIR__) . "/api/bootstrap.php";

if ($_SERVER["REQUEST_METHOD"] !== "POST") {
    http_response_code(405);
    header("allow: POST");
    exit;
}
```

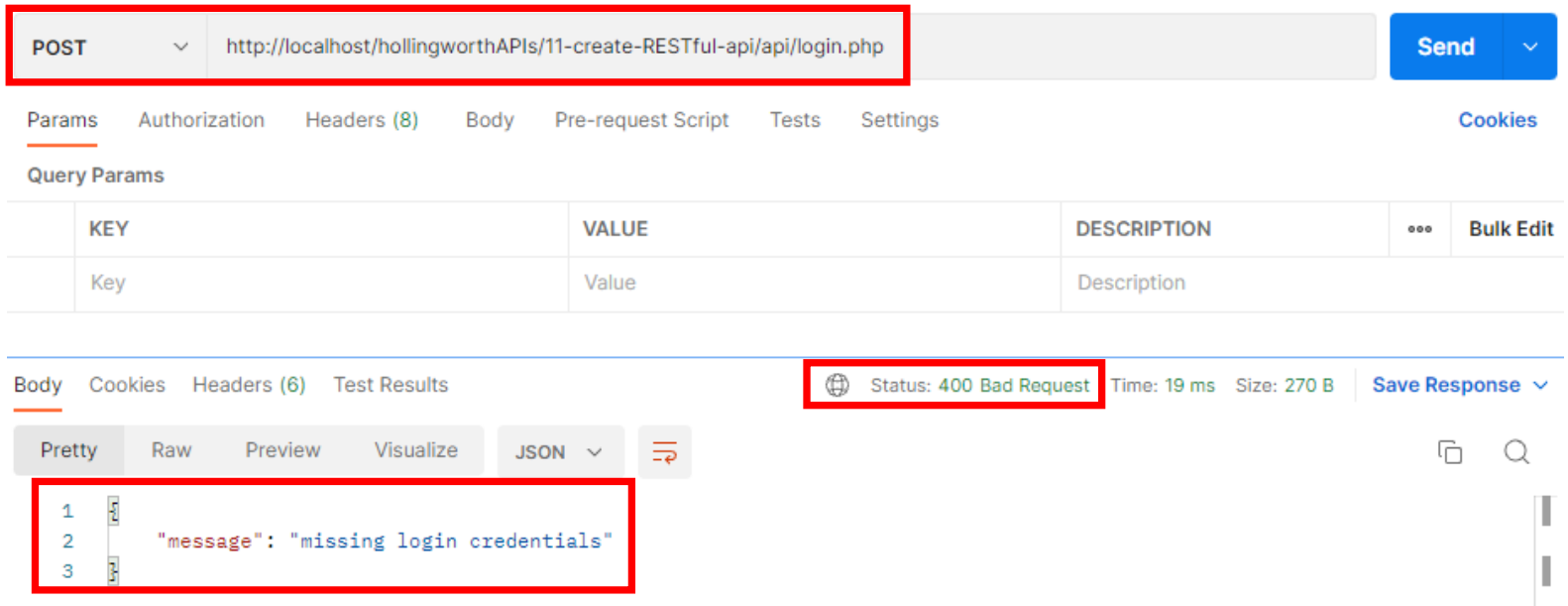
In src folder we create a file called login.php that first checks if the method is POST and returns a 405 Method not allowed if not.

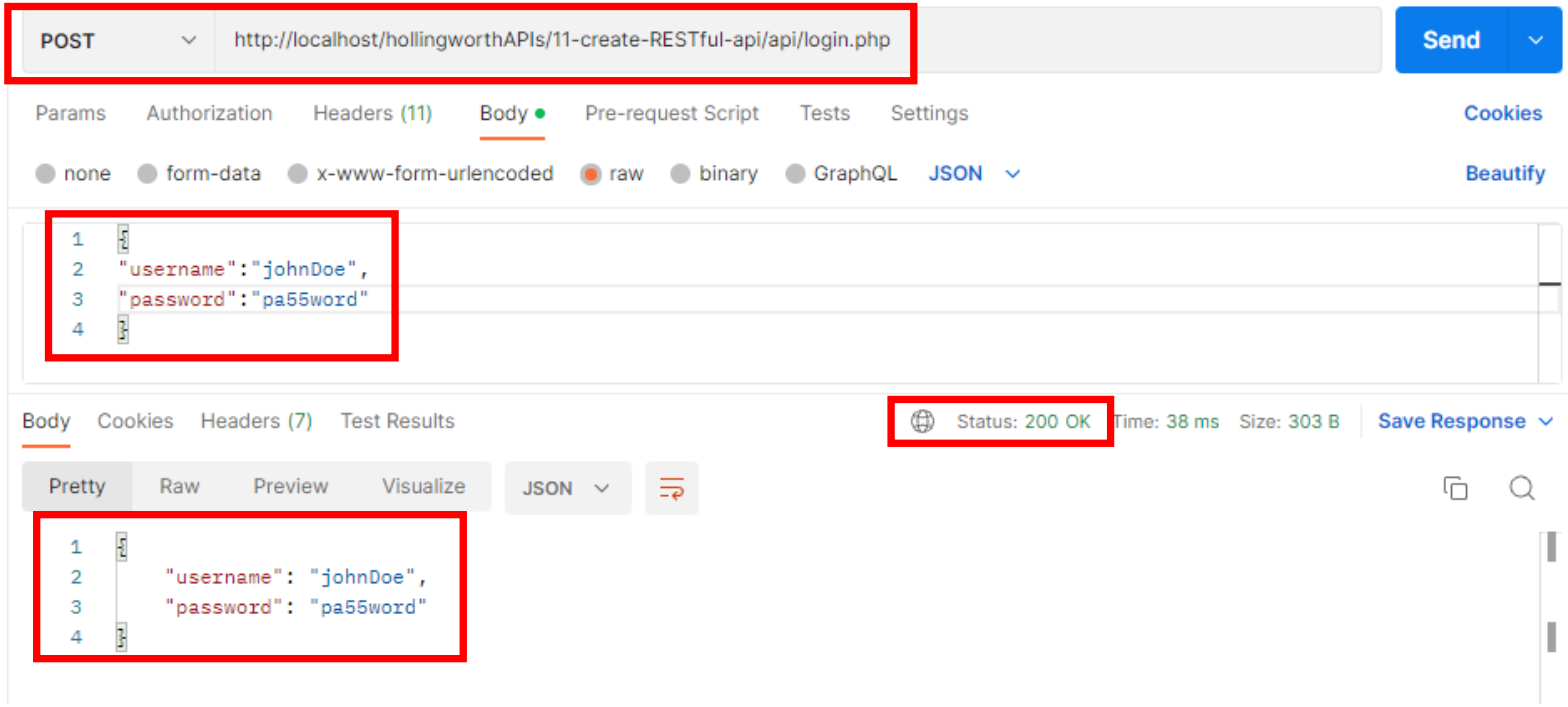
Note how this endpoint is not RESTful.



```php
<?php
declare(strict_types=1);
require dirname(__DIR__) . "/api/bootstrap.php";

if ($_SERVER["REQUEST_METHOD"] !== "POST") {
    http_response_code(405);
    header("allow: POST");
    exit;
}

$data = (array) json_decode(file_get_contents("php://input"), true);
if (!array_key_exists("username", $data) || !array_key_exists("password", $data)) {
    http_response_code(400);
    echo json_encode(["message" => "missing login credentials"]);
    exit;
}
```

Now we convert the php request into an array of data and check for username and password returning a 400 bad request and custom JSON message for missing login credentials.





But sending a POST request to the login script with a JSON object containing username and password returns a 200 and for the moment just returns the same object.

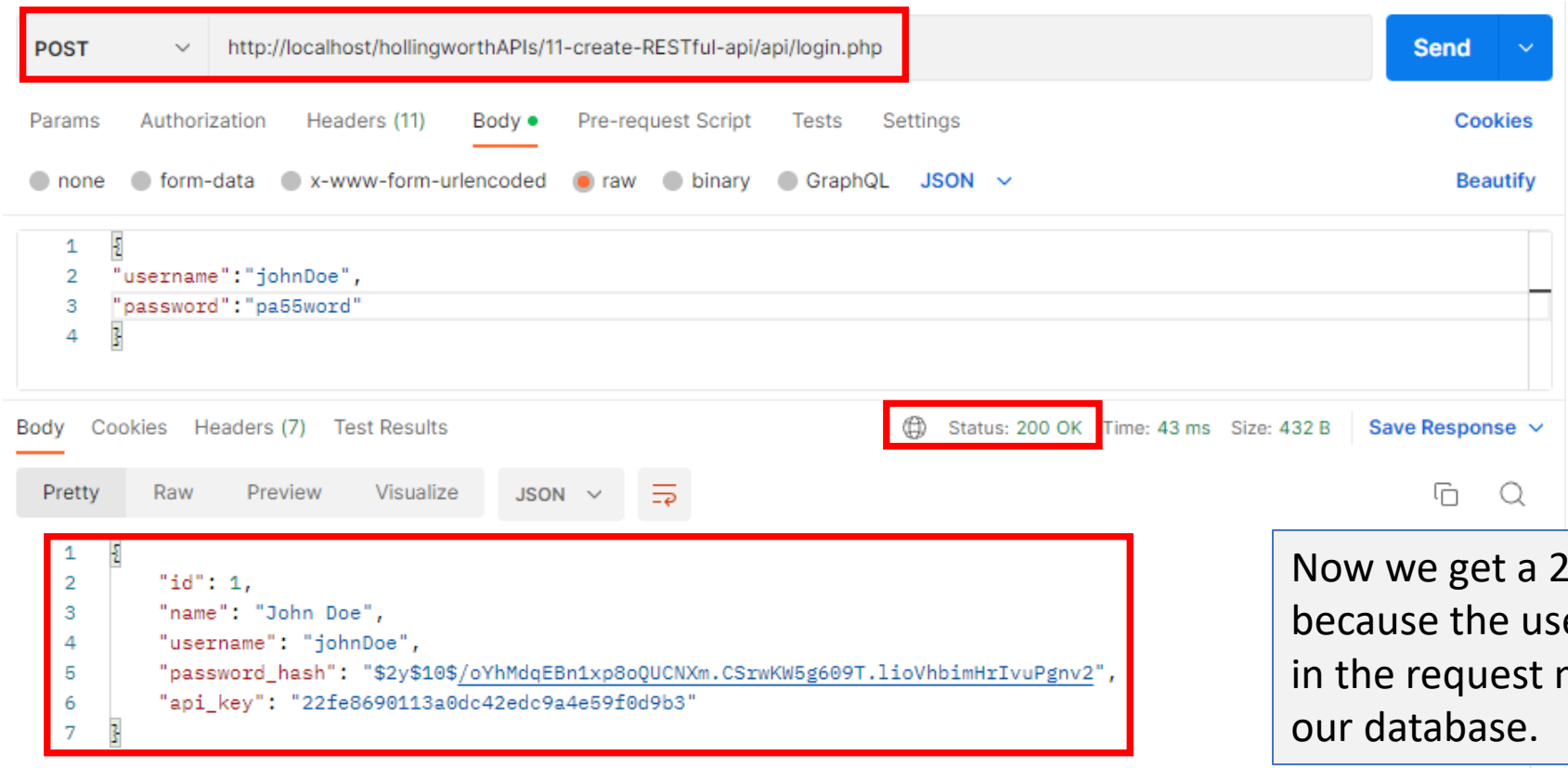
Select the user record based on the username in the request

```php
<?php

class UserGateway {
    private PDO $conn;

    public function __construct(Database $database) {
        $this->conn = $database->getConnection();
    }

    public function getByAPIkey(string $key) : array | false {
        $sql = "SELECT * FROM user WHERE api_key=:api_key";
        $stmt = $this->conn->prepare($sql);
        $stmt->bindValue(":api_key", $key, PDO::PARAM_STR);
        $stmt->execute();
        return $stmt->fetch(PDO::FETCH_ASSOC);
    }

    public function getByUsername(string $username): array | false {
        $sql = "SELECT * FROM user WHERE username=:username";
        $stmt = $this->conn->prepare($sql);
        $stmt->bindValue(":username", $username, PDO::PARAM_STR);
        $stmt->execute();
        return $stmt->fetch(PDO::FETCH_ASSOC);
    }
}
```

In the userGateway class we define a method that returns the user data array based on username.

```php
$database = new Database($_ENV["DB_HOST"], $_ENV["DB_NAME"],
$_ENV["DB_USER"], $_ENV["DB_PASS"]);
$user_gateway = new UserGateway($database);
$user = $user_gateway->getByUsername($data["username"]);

echo json_encode($user);
```

In the login.php script We instantiate a new database object passing in our mysql credentials then instantiate a new user_gateway object and call the previously created method to get user by username.

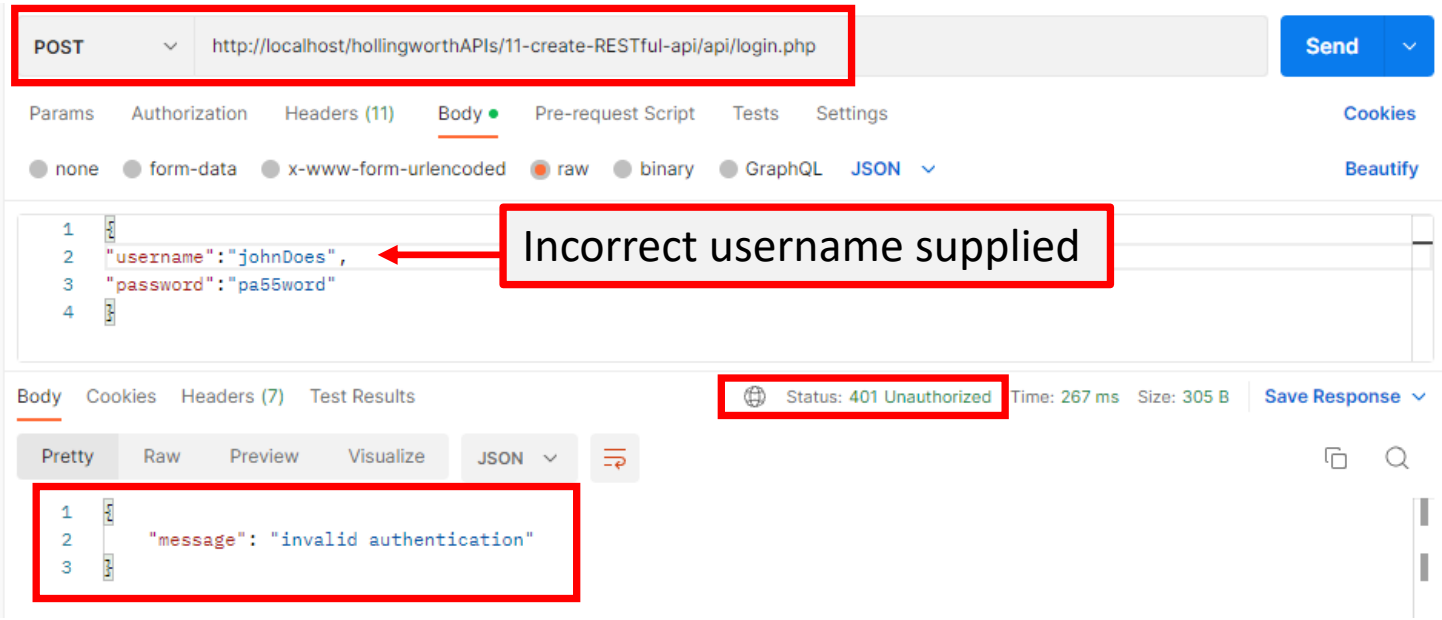


Now we get a 200 status code because the username passed in the request matches one in our database.

Check the username and password and return a 401 status code if invalid

```php
<?php

declare(strict_types=1);
require __DIR__ . "/bootstrap.php";

if ($_SERVER["REQUEST_METHOD"] !== "POST") {
    http_response_code(405);
    header("Allow: POST");
    exit;
}

$data = (array) json_decode(file_get_contents("php://input"), true);
if ( !array_key_exists("username", $data) ||
!array_key_exists("password", $data)) {
    http_response_code(400);
    echo json_encode(["message" => "missing login credentials"]);
    exit;
}

$database = new Database($_ENV["DB_HOST"], $_ENV["DB_NAME"],
$_ENV["DB_USER"], $_ENV["DB_PASS"]);
$user_gateway = new UserGateway($database);
$user = $user_gateway->getByUsername($data["username"]);

if ($user === false) {
    http_response_code(401);
    echo json_encode(["message" => "invalid authentication"]);
    exit;
}
```

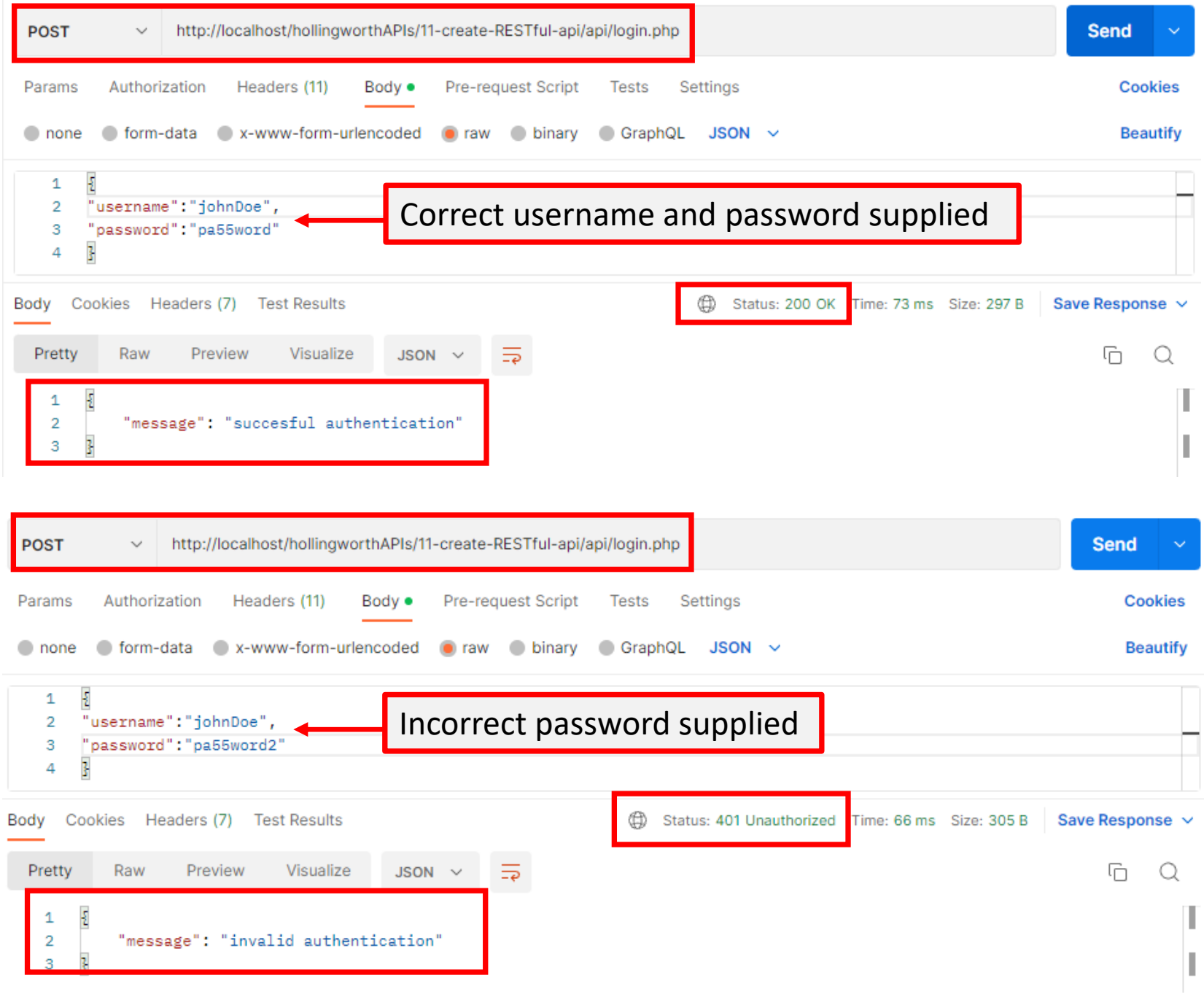
If the returned user is a false value, ie the submitted username does not exist in the database we can return a 401 unothorised status code and a custom JSON message


POST
http://localhost/hollingworthAPIs/11-create-RESTful-api/api/login.php
Send
Params
Authorization
Headers (11)
Body
Pre-request Script
Tests
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
{
"username":"johnDoes",
"password":"pa55word"
}
Incorrect username supplied
Body
Cookies
Headers (7)
Test Results
Status: 401 Unauthorized
Time: 267 ms
Size: 305 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
{
"message": "invalid authentication"
}

```php
<?php

declare(strict_types=1);
require __DIR__ . "/bootstrap.php";

if ($_SERVER["REQUEST_METHOD"] !== "POST") {
    http_response_code(405);
    header("Allow: POST");
    exit;
}

$data = (array) json_decode(file_get_contents("php://input"), true);
if ( !array_key_exists("username", $data) || !array_key_exists("password", $data)) {
    http_response_code(400);
    echo json_encode(["message" => "missing login credentials"]);
    exit;
}

$database = new Database($_ENV["DB_HOST"], $_ENV["DB_NAME"], $_ENV["DB_USER"],
$_ENV["DB_PASS"]);
$user_gateway = new UserGateway($database);
$user = $user_gateway->getByUsername($data["username"]);

if ($user === false) {
    http_response_code(401);
    echo json_encode(["message" => "invalid authentication"]);
    exit;
}

if (!password_verify($data["password"], $user["password_hash"])) {
    http_response_code(401);
    echo json_encode(["message" => "invalid authentication"]);
    exit;
}

echo json_encode(["message" => "succesful authentication"]);
```

We can validate the password against the one that we got back in the user array with the php function verify password. If this fails then we respond with a 401 unothorised status code.

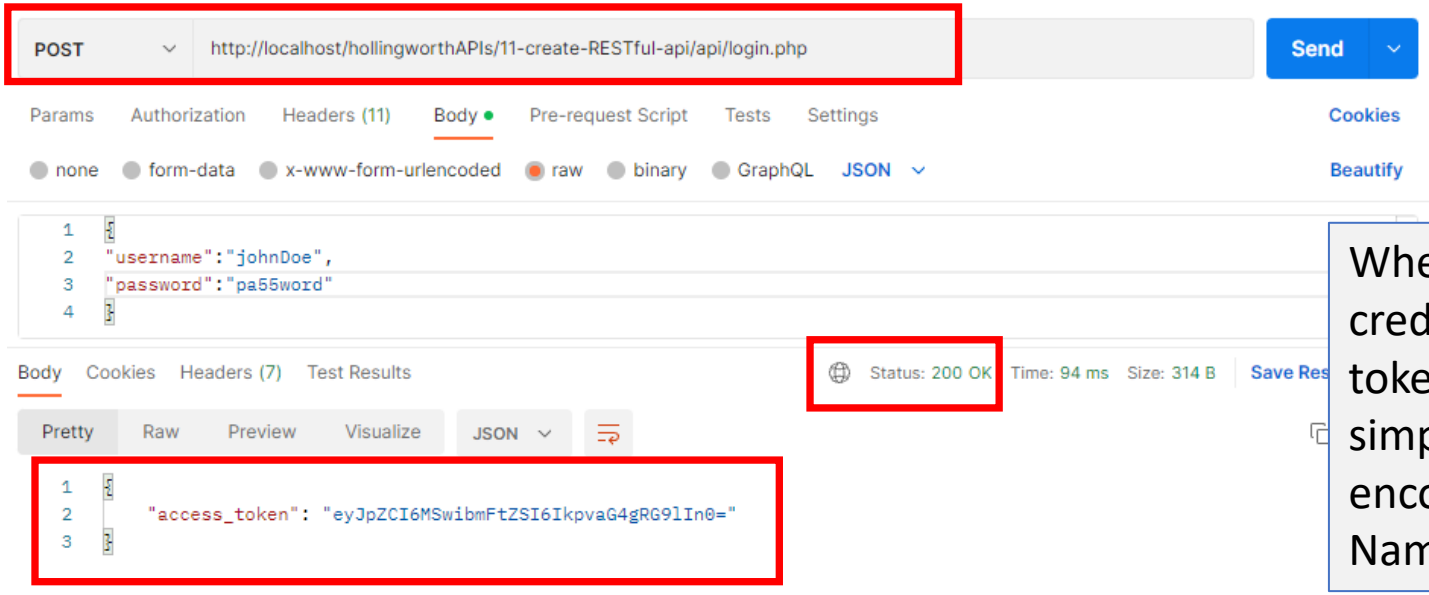
If authentication is successful then for now a simple success message is sent.


POST
http://localhost/hollingworthAPIs/11-create-RESTful-api/api/login.php
Send
Params
Authorization
Headers (11)
Body
Pre-request Script
Tests
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
1 {
2 "username":"johnDoe",
3 "password":"pa55word"
4 }
Correct username and password supplied
Body
Cookies
Headers (7)
Test Results
Status: 200 OK
Time: 73 ms
Size: 297 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
1 {
2 "message": "succesful authentication"
3 }
POST
http://localhost/hollingworthAPIs/11-create-RESTful-api/api/login.php
Send
Params
Authorization
Headers (11)
Body
Pre-request Script
Tests
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
1 {
2 "username":"johnDoe",
3 "password":"pa55word2"
4 }
Incorrect password supplied
Body
Cookies
Headers (7)
Test Results
Status: 401 Unauthorized
Time: 66 ms
Size: 305 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
1 {
2 "message": "invalid authentication"
3 }

Generate an encoded access token containing user details

```
$payload = [
    "id" => $user["id"],
    "name" => $user["name"]
    // Other data such as user role, user department etc
];

$access_token = base64_encode(json_encode($payload));
echo json_encode(["access_token" => $access_token]);
```

In the index.php file we can create a payload array that contains the user id and user name then convert it to json and base64_encode this into an ASKII string.
https://www.php.net/manual/en/function.base64-encode.php



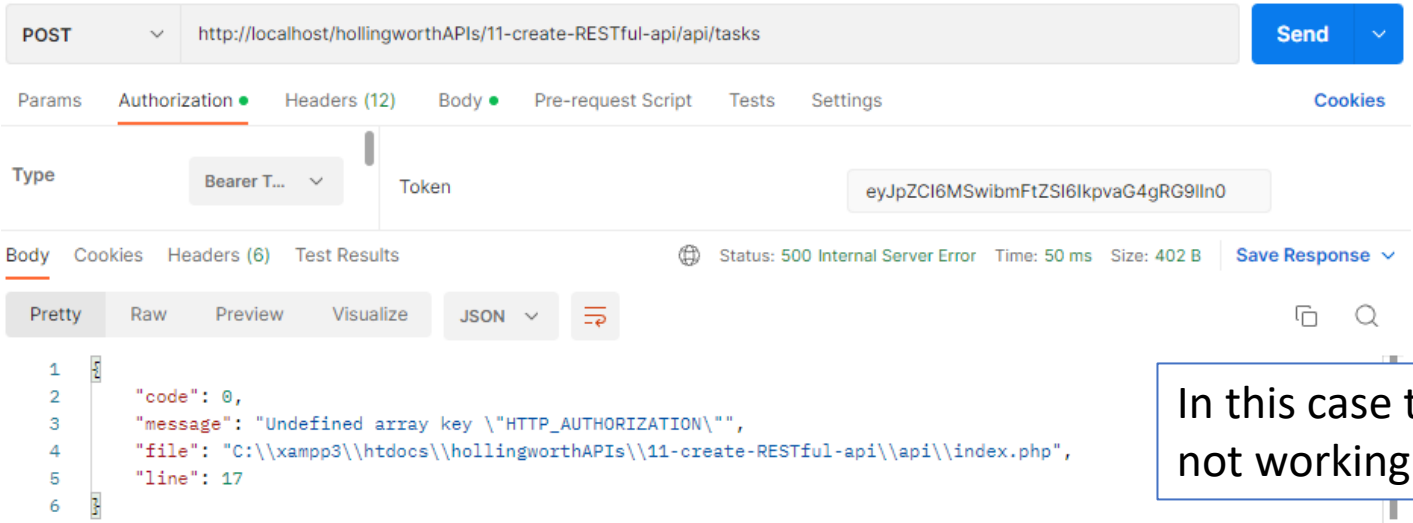
When a user logs in with correct credentials they get an access token that is sent as JSON. It is a simple string of characters but encoded within it is the ID and Name of the user.

Pass the access token to the task API endpoints in the authorisation header

It is standard practice to pass an access token in autorisiation header. When we use an autorisation header we have to pass an authorisation scheme as well ass the access token. The standard authorisation scheme is bearer, i.e. the bearer of this token is authorised.
Authorization: <auth-scheme> <authorization-parameters>
https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Authorization

```
<?php
declare(strict_types=1);
require dirname(__DIR__) . "/api/bootstrap.php";

$path = parse_url($_SERVER["REQUEST_URI"], PHP_URL_PATH);
$parts = explode("/", $path);
$resource = $parts[4];
$id = $parts[5] ?? null;
if ($resource != "tasks") {
    http_response_code(404);
    exit;
}

$database = new Database($_ENV["DB_HOST"], $_ENV["DB_NAME"], $_ENV["DB_USER"], $_ENV["DB_PASS"]);
$user_gateway = new UserGateway($database);

var_dump($_SERVER["HTTP_AUTHORIZATION"]);
exit;
```

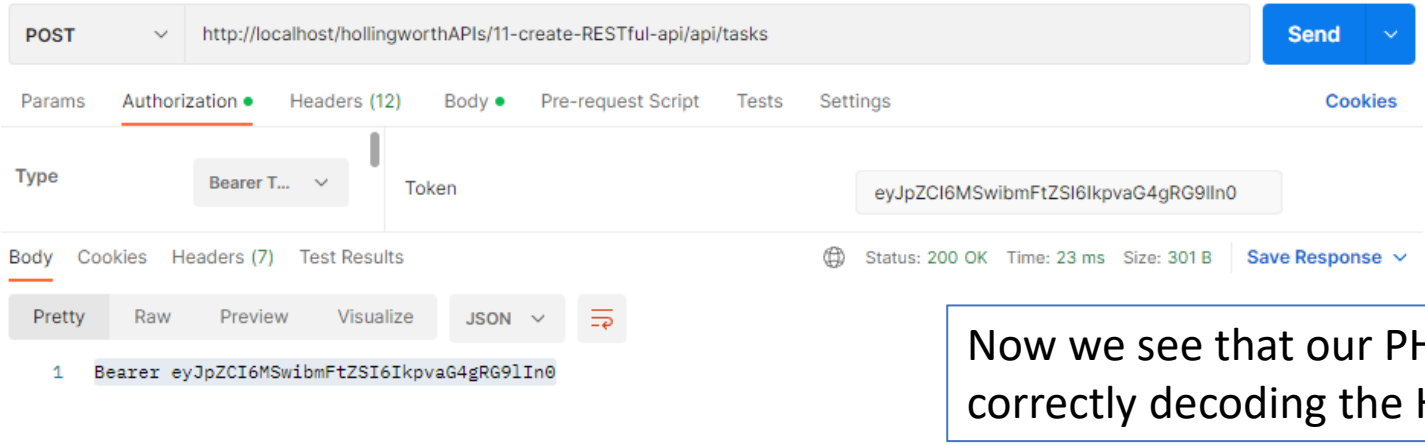
Now when we make a HTTP request, We can capture the HTTP_AUTORIZATION header from the $_SERVER super global in the front end controller. But this depends on the Apache web server version.



In this case the HTTP_AUTORIZATION is not working with the version of apache.

The workaround is to get the http autorisation header by using the apache_request_headers function().

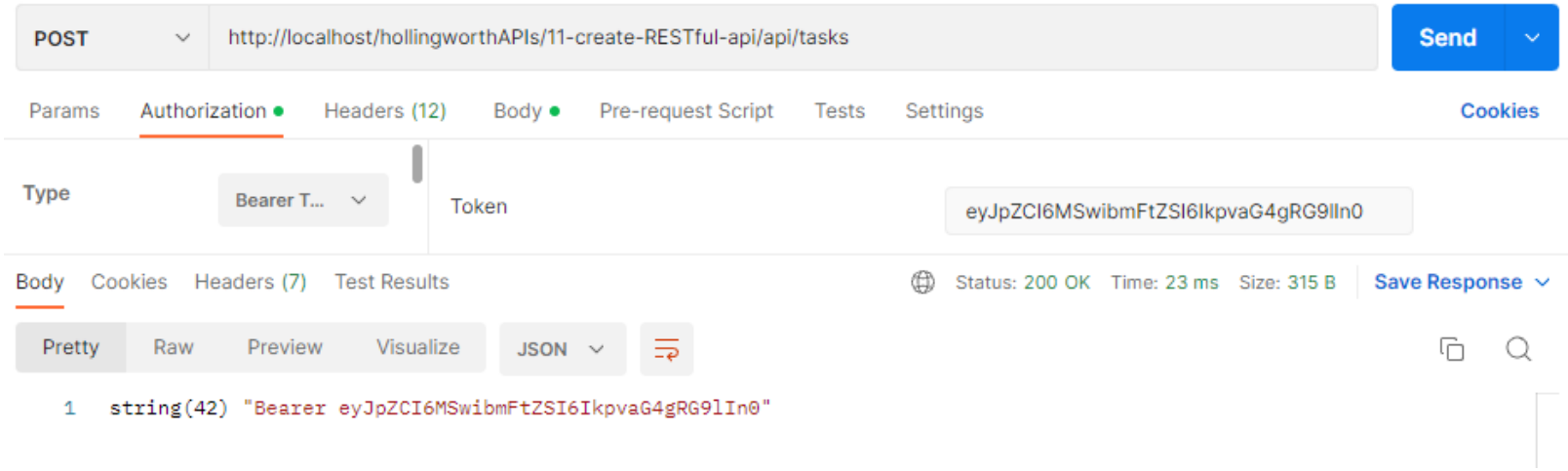
```
// var_dump($_SERVER["HTTP_AUTHORIZATION"]);
$headers = apache_request_headers();
echo $headers["Authorization"];
exit;
```



Now we see that our PHP front end controller is correctly decoding the HTTP authorisation header.

```
RewriteEngine On
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d
RewriteCond %{REQUEST_FILENAME} !-l
RewriteRule . index.php [L]

SetEnvIf Authorization "(.*)" HTTP_AUTHORIZATION=$1
```

Another method is to configure apache so that the authorisation header is available by configuring the apache config file or using a htacces file.

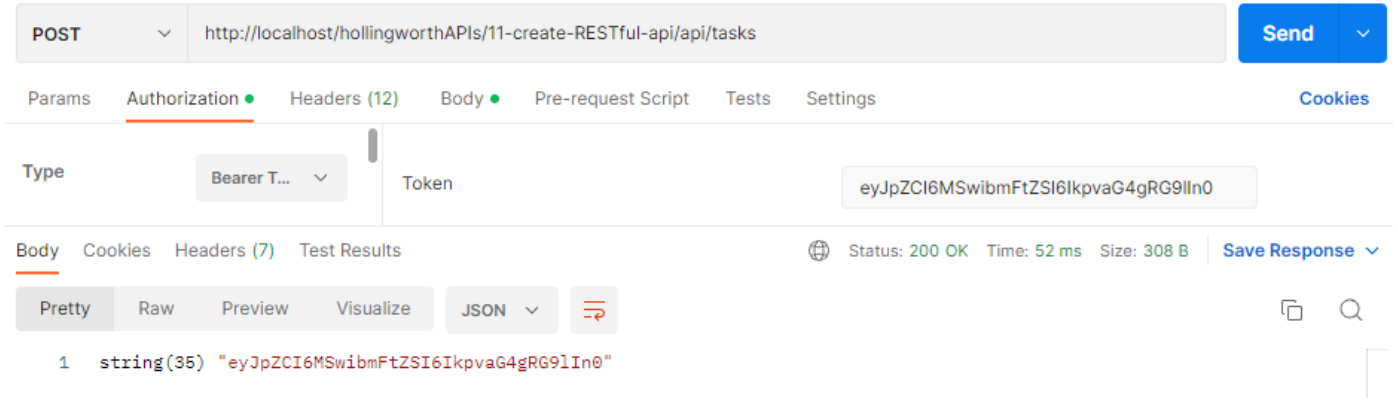


Validate the access token and decode its contents

```
public function authenticateAccessToken(): bool {
    if (!preg_match("/^Bearer\s+(.*)$/", $_SERVER["HTTP_AUTHORIZATION"], $matches)) {
        http_response_code(400);
        echo json_encode(["message" => "incomplete authorization header"]);
        return false;
    }
    var_dump($matches[1]);
    exit;
}
```

in the Auth class we have a method for authenticating API keys so we can add a new function to authenticate access tokens. We will do this by matching the the word bearer in a regular expression and capture the rest of the expression by putting it in parenthesis (.*). If the access token contains bearer then the second part of the $matches array will be the access token.

Note how we see the results of the second element of the matches array which is actually the acess token string.

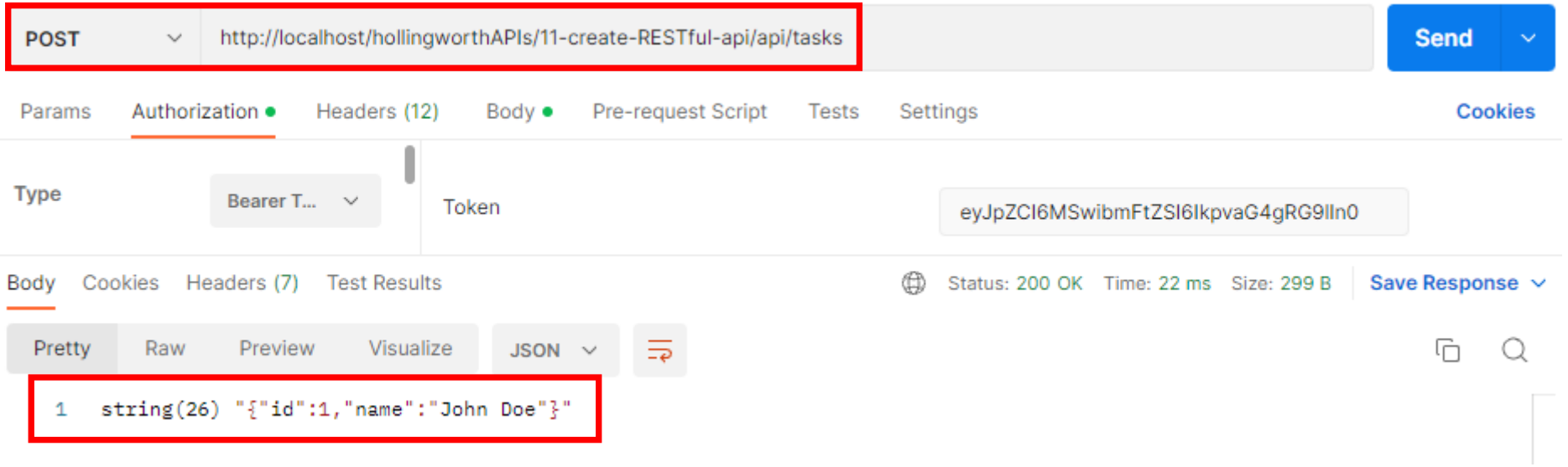



```
$auth = new Auth($user_gateway);
if (!$auth->authenticateAccessToken()) {
    exit;
}
```

In the front end controller (index.php) we can call the authenticateAccessToken method instread of the AuthenticateAPIkey method.

```php
public function authenticateAccessToken(): bool {
    if ( ! preg_match("/^Bearer\s+(.*)$/", $_SERVER["HTTP_AUTHORIZATION"], $matches)) {
        http_response_code(400);
        echo json_encode(["message" => "incomplete authorization header"]);
        return false;
    }
    $plain_text = base64_decode($matches[1], true);
    if ($plain_text === false) {
        http_response_code(400);
        echo json_encode(["message" => "invalid authorization header"]);
    }
    var_dump($plain_text);
    exit;
}
```

We need to base64_decode the $matches[1] and handle the scenario that it is unable to decode it.



```
public function authenticateAccessToken(): bool {
    if ( ! preg_match("/^Bearer\s+(.*)$/", $_SERVER["HTTP_AUTHORIZATION"], $matches)) {
        http_response_code(400);
        echo json_encode(["message" => "incomplete authorization header"]);
        return false;
    }
    $plain_text = base64_decode($matches[1], true);
    if ($plain_text === false) {
        http_response_code(400);
        echo json_encode(["message" => "invalid authorization header"]);
        return false;
    }
    $data = json_decode($plain_text, true);
    if ($data === false) {
        http_response_code(400);
        echo json_encode(["message" => "invalid JSON"]);
        return false;
    }
    // var_dump($data);
    // exit;
    return true;
}
```

```
Array
(
    [id
] => 1
    [name
] => John Doe
)
```

Now we can decode the plain text from JSON to get an array that contains the user ID and user name. if the access key successfully decodes from base64 and then JSON we can return true.

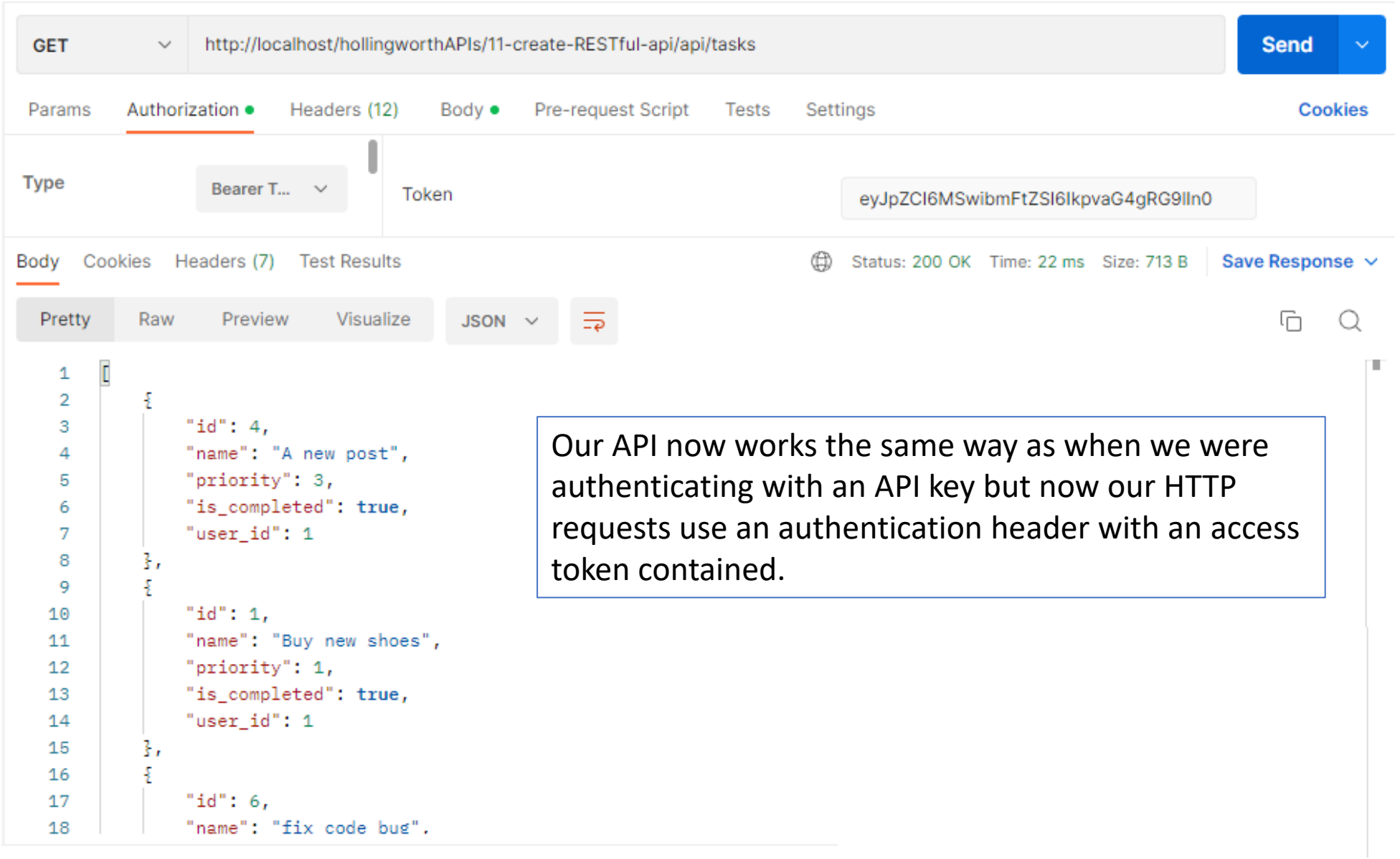
In the index.php page after calling this function we can just `echo "valid authentication";` and `exit;`

Get the authenticated user data from the token

```
public function authenticateAccessToken(): bool {
    if (!preg_match("/^Bearer\s+(.*)$/", $_SERVER["HTTP_AUTHORIZATION"], $matches)) {
       http_response_code(400);
       echo json_encode(["message" => "incomplete authorization header"]);
       return false;
    }
    $plain_text = base64_decode($matches[1], true);
    if ($plain_text === false) {
        http_response_code(400);
        echo json_encode(["message" => "invalid authorization header"]);
        return false;
    }
    $data = json_decode($plain_text, true);
    if ($data === null) {
        http_response_code(400);
        echo json_encode(["message" => "invalid JSON"]);
        return false;
    }
    $this->user_id = $data["id"];
    return true;
}
```

In the Auth class we can now set the user_id to that of the Id we decode from the access token. The code in index.php runs as per the API key.

```
$auth = new Auth($user_gateway);
if ( ! $auth->authenticateAccessToken()) {
    exit;
}
$user_id = $auth->getUserID();
$task_gateway = new TaskGateway($database);
$controller = new TaskController($task_gateway, $user_id);
$controller->processRequest($_SERVER['REQUEST_METHOD'],
$id);
?>
```


GET
http://localhost/hollingworthAPIs/11-create-RESTful-api/api/tasks
Send
Params
Authorization
Headers (12)
Body
Pre-request Script
Tests
Settings
Cookies
Type
Bearer T...
Token
eyJpZCI6MSwibmFtZSI6IkpvaG4gRG9lIn0
Body
Cookies
Headers (7)
Test Results
Status: 200 OK
Time: 22 ms
Size: 713 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
[
{
"id": 4,
"name": "A new post",
"priority": 3,
"is_completed": true,
"user_id": 1
},
{
"id": 1,
"name": "Buy new shoes",
"priority": 1,
"is_completed": true,
"user_id": 1
},
{
"id": 6,
"name": "fix code bug",
Our API now works the same way as when we were authenticating with an API key but now our HTTP requests use an authentication header with an access token contained.

An Introduction to Authentication Using JSON web Tokens (JWTs)

An Introduction to JSON web Tokens (JWTs)

Previously we used an access token that was comprised of two user parameters that were encoded using Base64 but this can clearly be forged, I.e. someone else sending an identical access token to use that users API credits.

The industry standard is JWT https://jwt.io

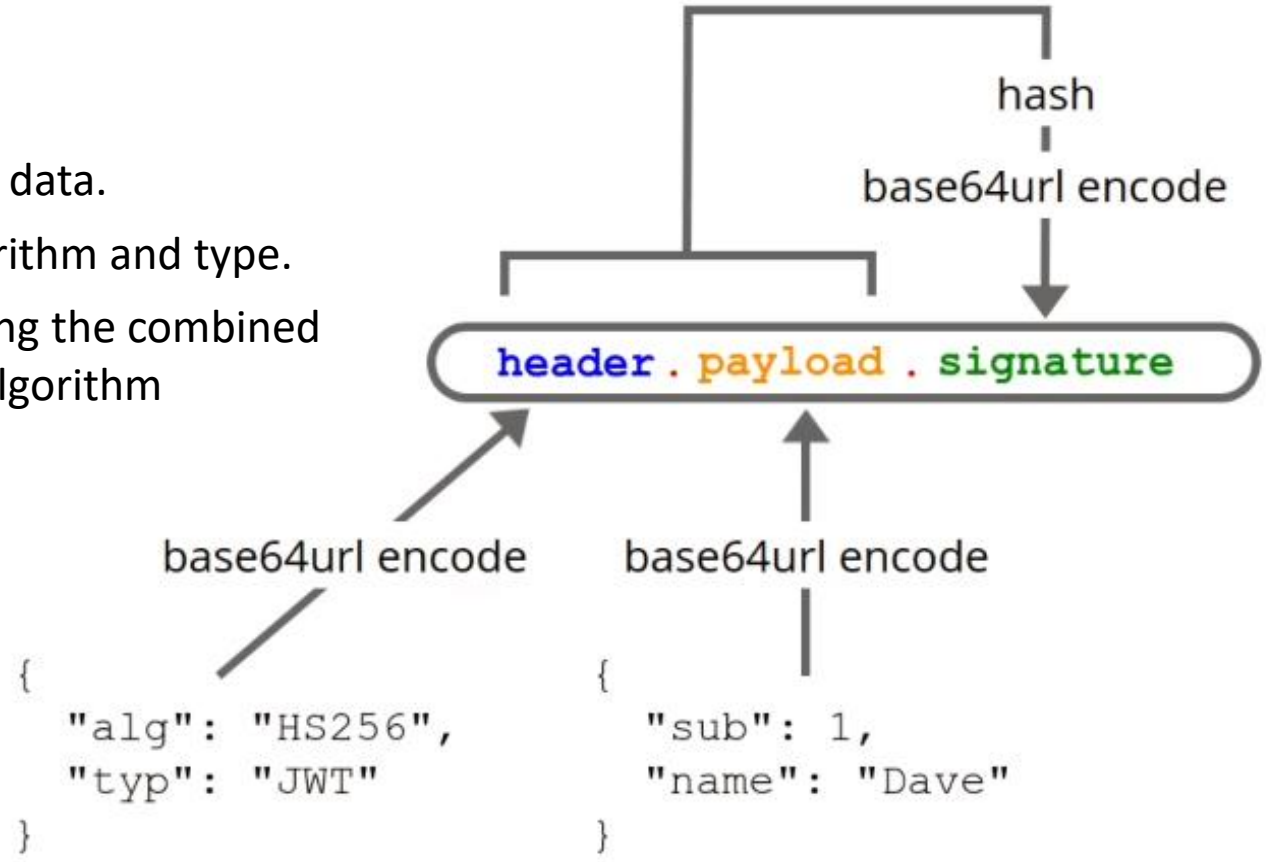
JWT standard defines a way to encode data in a token that is URL friendly and that cannot be forged because it is digitally signed.

The JWT is made of three parts:

1) The **payload** is base64 encoded data.
2) The **header** is made of the algorithm and type.
3) The **Signature** is made by hashing the combined header and payload using the algorithm specified in the header.

A JWT cannot be forged because if the data changes then the signature would also be different.

JWT is what we will use as an access token.



Create a class to encode a payload in a JWT

There are some existing packages to create JWTs but as it is straight forward we will develope our own code to do this. Strting by creating a JWTcodec.php file in the src folder.

https://en.wikipedia.org/wiki/Base64#The_URL_applications
About base64

https://github.com/firebase/php-jwt
A pre built JWT package

https://www.php.net/manual/en/function.hash-hmac.php
PHP inbuilt function to generate a keyed hash value using HMAC method.

https://www.php.net/manual/en/function.hash-equals.php
PHP inbuilt function that performs Timing attack safe string comparison of two strings.
This function should be used to mitigate timing attacks; for instance, when testing crypt() password hashes.

https://www.allkeysgenerator.com/Random/Security-Encryption-Key-Generator.aspx
A webpage that can generate random keys in many different lengths and also HEXify it.

```php
<?php

class JWTcodec {

    public function encode(array $payload): string {
        $header = json_encode([
            "typ" => "JWT",
            "alg" => "HS256"
        ]);
        $header = $this->base64urlEncode($header);

        $payload = json_encode($payload);
        $payload = $this->base64urlEncode($payload);

        $randomHexKey = "5A7134743777217A25432A462D4A614E645267556B586E3272357538782F413F";
        $signature = hash_hmac("SHA256", $header . "." . $payload, $randomHexKey, true);
        $signature = $this->base64urlEncode($signature);

        return $header . "." . $payload . "." . $signature;
    }

    private function base64urlEncode(string $text) {
        return str_replace(["+", "/", "e"],["-","_",""], base64_encode($text));
    }
}
```

This class method takes a payload and returns a JWT string by building the three components of a JWT.

Build the base64 encoded header.

Build the base64 encoded payload.

Build the base64 encoded signature by hashing the header+payload with a random key.

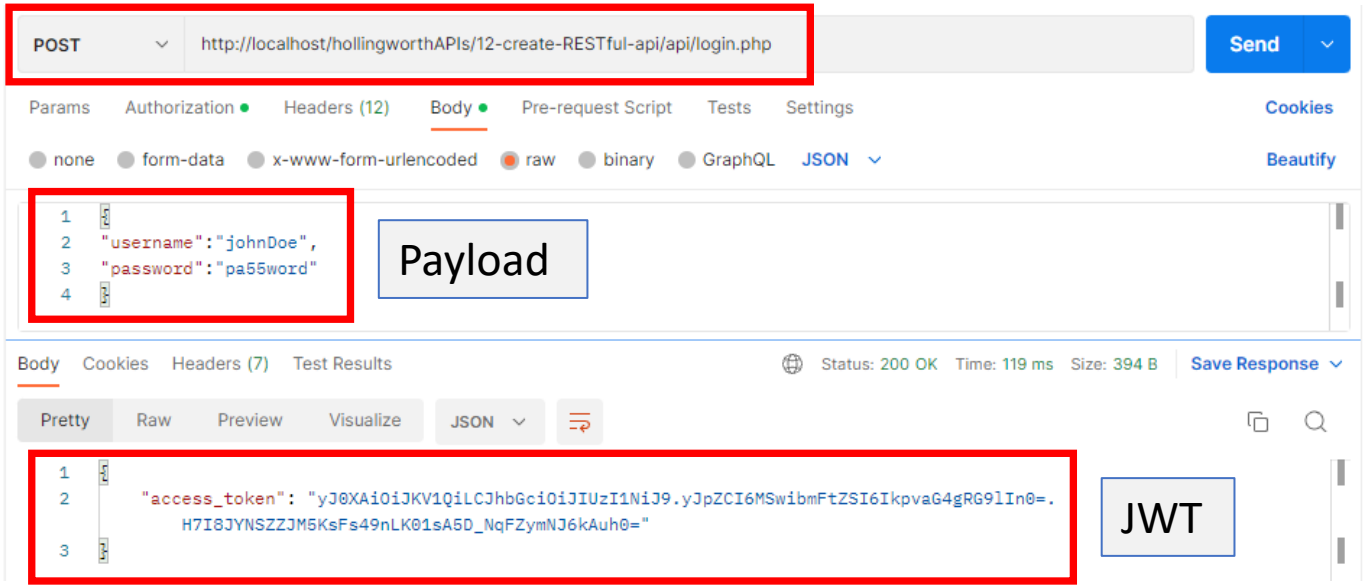
Return JWT access token.

The base64URLencode replaces undesirable characters such as plus, slash and e.

Generate a JWT access token in the login endpoint containing JWT claims

```php
<?php

...

$payload = [
    "sub" => $user["id"],
    "name" => $user["name"]
    // Other data such as user role, user department etc
];

$codec = new JWTcodec;
$access_token = $codec->encode($payload);
echo json_encode(["access_token" => $access_token]);
```

At the bottom of the login.php script we can now use the codec to generate our access token.



Now when we send a login request we get a JWT access token back.

in a JWT token the indexes for specific items in the payload need to be defined as per a specific standard, RFC 7519. https://www.iana.org/assignments/jwt/jwt.xhtml

These are all pieces of information about a spcefic user. In our example we are sending user ID and name as the payload of the JWT claim. The ID is known as sunb or subject in the specifications.

JSON Web Token Claims

Registration Procedure(s)
Specification Required
Expert(s)
John Bradley, Brian Campbell, Michael B. Jones, Chuck Mortimore
Reference
[RFC7519]
Note
Registration requests should be sent to the mailing list described in [RFC7519].

Available Formats

csv

```
$payload = [
    "sub" => $user["id"],
    "name" => $user["name"]
    // Other data such as user role, user department etc
];
```

Claim Name	Claim Description	Change Controller	Reference
iss	Issuer	[IESG]	[RFC7519, Section 4.1.1]
sub	Subject	[IESG]	[RFC7519, Section 4.1.2]
aud	Audience	[IESG]	[RFC7519, Section 4.1.3]
exp	Expiration Time	[IESG]	[RFC7519, Section 4.1.4]
nbf	Not Before	[IESG]	[RFC7519, Section 4.1.5]
iat	Issued At	[IESG]	[RFC7519, Section 4.1.6]
jti	JWT ID	[IESG]	[RFC7519, Section 4.1.7]
name	Full name	[OpenID Foundation Artifact Binding Working Group]	[OpenID Connect Core 1.0, Section 5.1]
given_name	Given name(s) or first name(s)	[OpenID Foundation Artifact Binding Working Group]	[OpenID Connect Core 1.0, Section 5.1]
family_name	Surname(s) or last name(s)	[OpenID Foundation Artifact Binding Working Group]	[OpenID Connect Core 1.0, Section 5.1]
middle_name	Middle name(s)	[OpenID Foundation Artifact Binding Working Group]	[OpenID Connect Core 1.0, Section 5.1]

Add a method to decode the payload from the JWT

```php
<?php

class JWTcodec {

. . .

    public function decode($string $token): array {

        if(preg_match("/^(?<header>\.+)\.(?<payload>.+)\.(?<signature>.*)$/", $token, $matches !==1)) {
            throw new InvalidArgumentException("invalid token format");
        }

        $randHexKey = "5A7134743777217A25432A462D4A614E645267556B586E3272357538782F413F";
        $signature = hash_hmac("SHA256", $matches["header"] . "." . $matches["payload"], $randHexKey, true);
        $signature_from_token = $this->base64urlDecode($matches["signature"]);

        if(!has_equals($signature, $signature_from_token)) {
            throw new Exception("signature does not match");
        }



        $payload = json_decode($this->base64urlDecode($matches["payload"]), true);
        return $payload;
    }

. . .
    private function base64urlDecode(string $text) {
        return str_replace(["-","_"],["+", "/"], base64_encode($text));
    }
}
```

This class method takes a JWT token string and returns the payload array after comparing the signature.

A regular expression is used to match the three components of the token.

We can compare the signature we created with the signature from the token to check they match. This is where the security comes from.

If the signature passes authentication we can json and base64 decode the payload and return it.

The base64URLdecode reverses the encode function.

Pass in the secret key used for hashing as a dependancy

```
DB_HOST="localhost"
DB_NAME="hollingworthapis"
DB_USER="api_db_user"
DB_PASS="pa55word"

SECRET_KEY="5A7134743777217A25432A462D4A614E645267556B586E3272357538782F413F"
```

In the dot env file we add a new key value pair called secret key.

```
$codec = new JWTcodec($_ENV["SECRET_KEY"]);
$access_token = $codec->encode($payload);
echo json_encode(["access_token" => $access_token]);
```

In the login.php script when we instantiate the JWTcodec class we now need to pass in the key from the dot env file using the $_ENV superglobal.

```php
<?php
class JWTcodec {

    public function __construct(private string $key) {}

    public function encode(array $payload): string {
        $header = json_encode(["typ" => "JWT", "alg" => "HS256"]);
        $header = $this->base64urlEncode($header);

        $payload = json_encode($payload);
        $payload = $this->base64urlEncode($payload);

        $signature = hash_hmac("sha256", $header . "." . $payload, $this->key, true);
        $signature = $this->base64urlEncode($signature);

        return $header . "." . $payload . "." . $signature;
    }

    public function decode(string $token): array
    {
        if (preg_match("/^(?<header>.+)\.(?<payload>.+)\.(?<signature>.+)$/", $token, $matches) !== 1) {
            throw new InvalidArgumentException("invalid token format");
        }

        $signature = hash_hmac("sha256", $matches["header"] . "." . $matches["payload"], $this->key, true);
        $signature_from_token = $this->base64urlDecode($matches["signature"]);

        if (!hash_equals($signature, $signature_from_token)) {
            throw new Exception("signature doesn't match");
        }

        $payload = json_decode($this->base64urlDecode($matches["payload"]), true);
        return $payload;
    }
    . . .
```

We add a constructor function that has the key as a private variable. We can call the key in out code.

Authenticate the task endpoints using the JWT

```php
<?php

class Auth {

    private int $user_id;
    public function __construct(private UserGateway $user_gateway,
                                private JWTcodec $codec) {}

. . .

    public function authenticateAccessToken(): bool {
        if (!preg_match("/^Bearer\s+(.*)$/", $_SERVER["HTTP_AUTHORIZATION"], $matches)) {
            http_response_code(400);
            echo json_encode(["message" => "incomplete authorization header"]);
            return false;
        }
        try {
            $data = $this->codec->decode($matches[1]);
        }
        catch (Exception $e) {
            http_response_code(400);
            echo json_encode(["message" => $e->getMessage()]);
            return false;
        }
        $this->user_id = $data["sub"];
        return true;
    }
}
```

In the Auth class we also need the JWT codec in the constructor

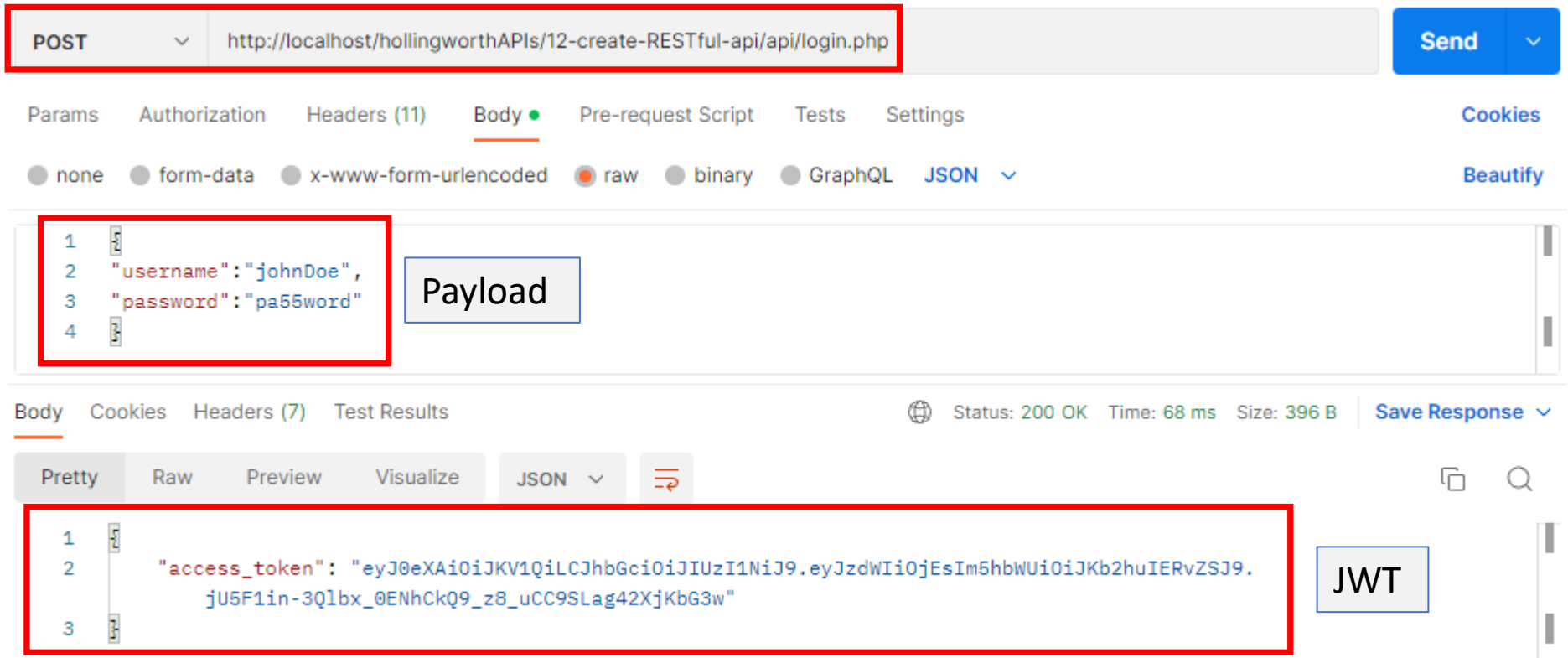
In the Auth class we can get the $data which is the second part or payload of the JWT.

If the JWT token does not pass validation then we can hadle this in the catch.

As before we retuen the user_id. Not how it is now called sub from the JWT token.

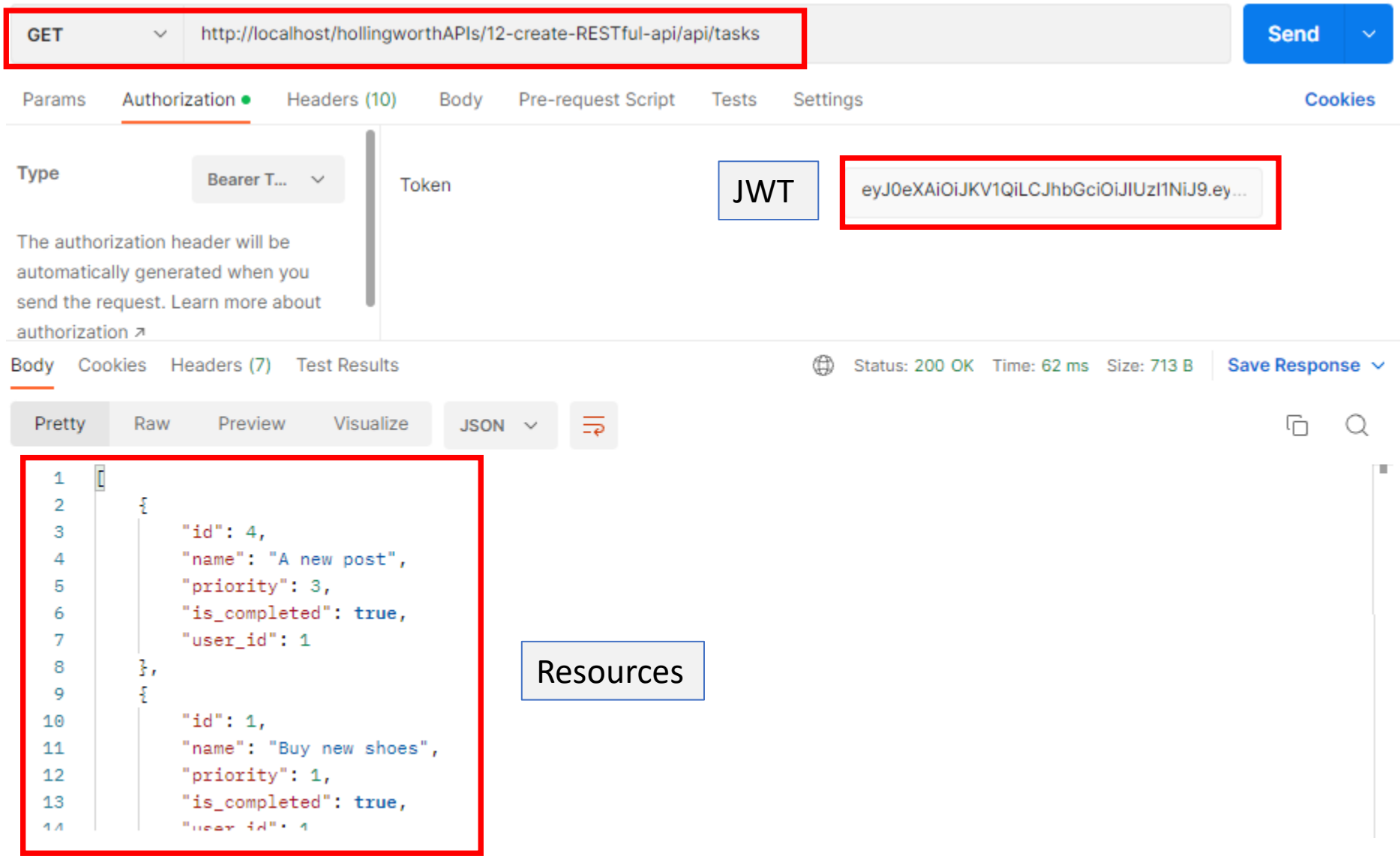
```php
<?php

declare(strict_types=1);
require dirname(__DIR__) . "/api/bootstrap.php";

$path = parse_url($_SERVER["REQUEST_URI"], PHP_URL_PATH);
$parts = explode("/", $path);
$resource = $parts[4];
$id = $parts[5] ?? null;
if ($resource != "tasks") {
    http_response_code(404);
    exit;
}

$database = new Database($_ENV["DB_HOST"], $_ENV["DB_NAME"],
$_ENV["DB_USER"], $_ENV["DB_PASS"]);
$user_gateway = new UserGateway($database);
$codec = new JWTcodec($_ENV["SECRET_KEY"]);
$auth = new Auth($user_gateway, $codec);
if ( ! $auth->authenticateAccessToken()) {
    exit;
}
$user_id = $auth->getUserID();
$task_gateway = new TaskGateway($database);
$controller = new TaskController($task_gateway, $user_id);
$controller->processRequest($_SERVER['REQUEST_METHOD'], $id);
?>
```

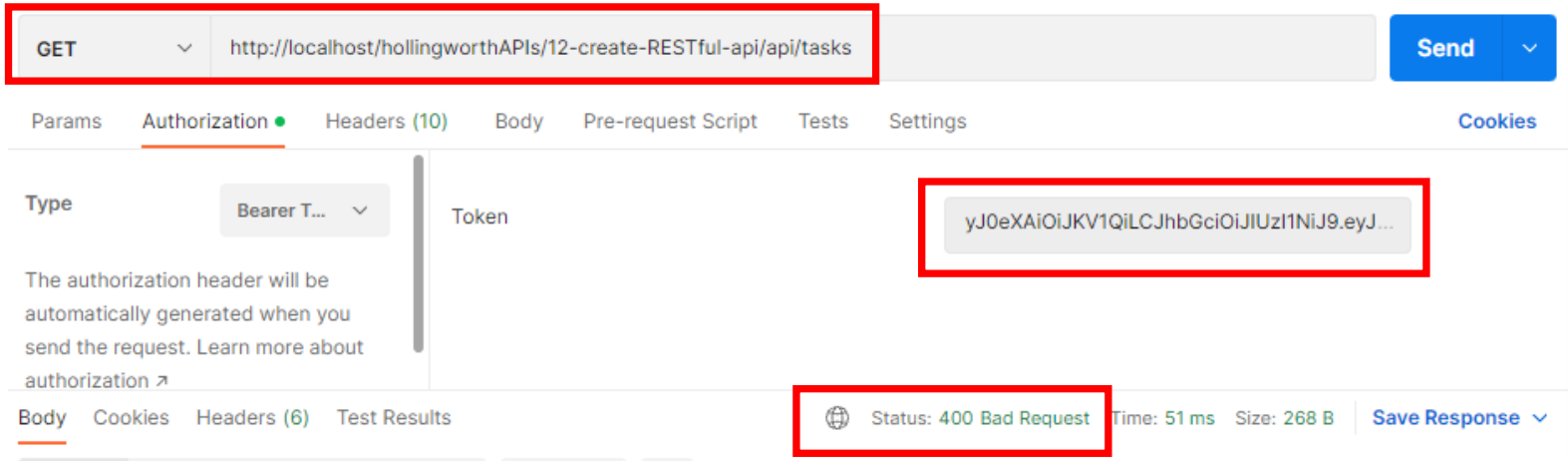
In the Front end controller when we call the authentication method we also need to pass in the codec.



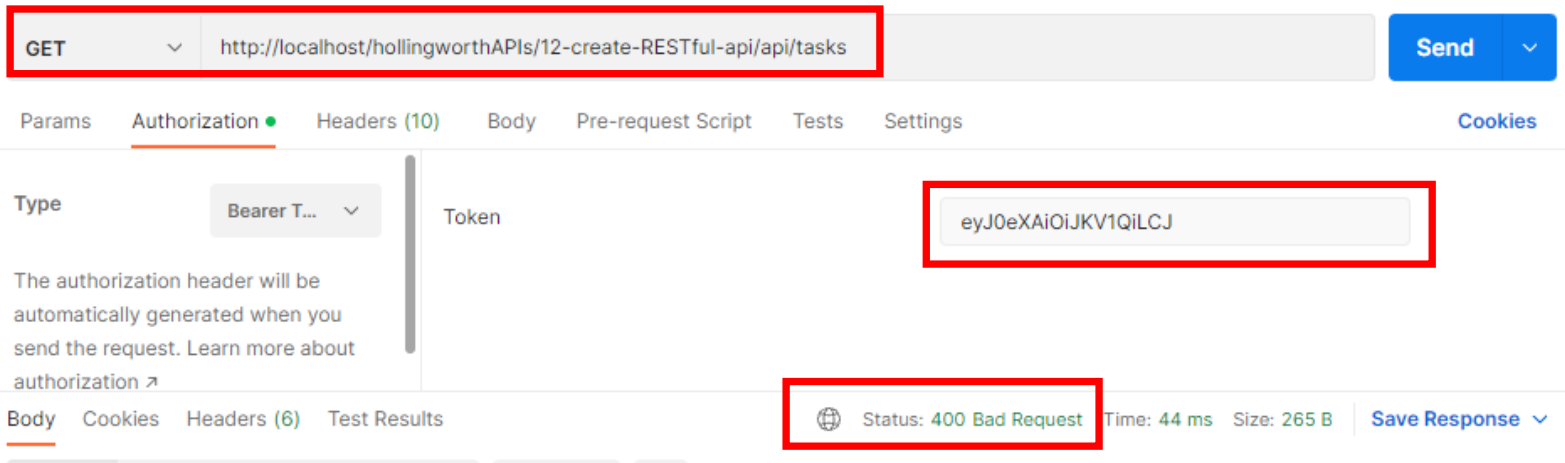
Now we send a login with a payload and get a JWT back

The JWT can be used to get sesources from the API endpoint.





In this example I have modified the JWT to be incorrect by removing the first character and I get back the custom error message signature does not match.



In this example I have modified the JWT to be incorrect by removing most of the characters and I get back the custom error message invalid token format.

Use a custom eception class to return 401 if the signature is invalid

In the previous page examples we get back a standard 400 bad request status code but we should really get a 401 unothorised.

```php
<?php

class InvalidSignatureException extends Exception {}
```

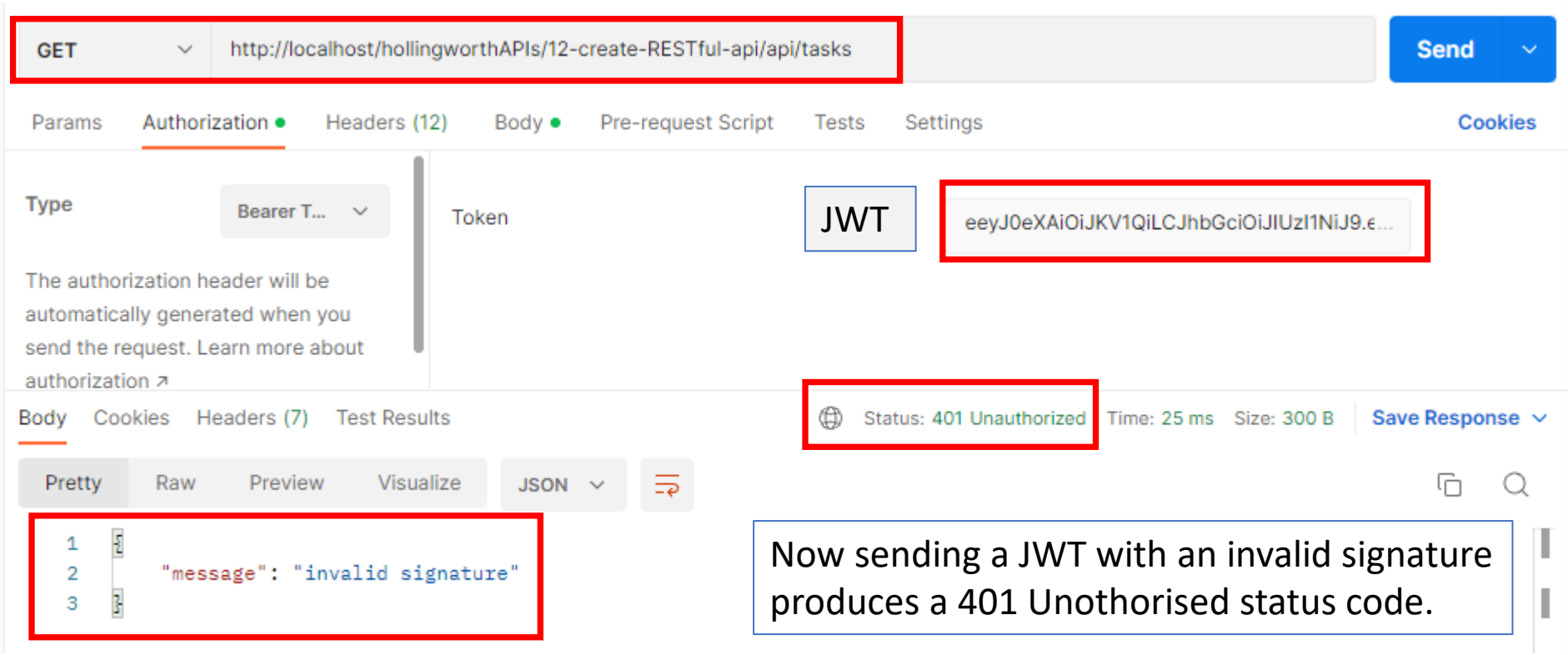
In the src folder a new file is created called InvalidSignatureException.php which is a class that extends the exceptions class.

```php
if ( ! hash_equals($signature, $signature_from_token)) {
    throw new InvalidSignatureException;
}
```

In the codec file if the signatures do not match then we need to throw to the newly created error class.

```php
 try {
    $data = $this->codec->decode($matches[1]);
}
catch (invalidSignatureException) {
    http_response_code(401);
    echo json_encode(["message" => "invalid signature"]);
    return false;
}
catch (Exception $e) {
    http_response_code(400);
    echo json_encode(["message" => $e->getMessage()]);
    return false;
}
$this->user_id = $data["sub"];
return true;
```

In the Auth file we can handle the InvalidSignatureException in a separate catch block returning a 401 status code and a custom JSON message.

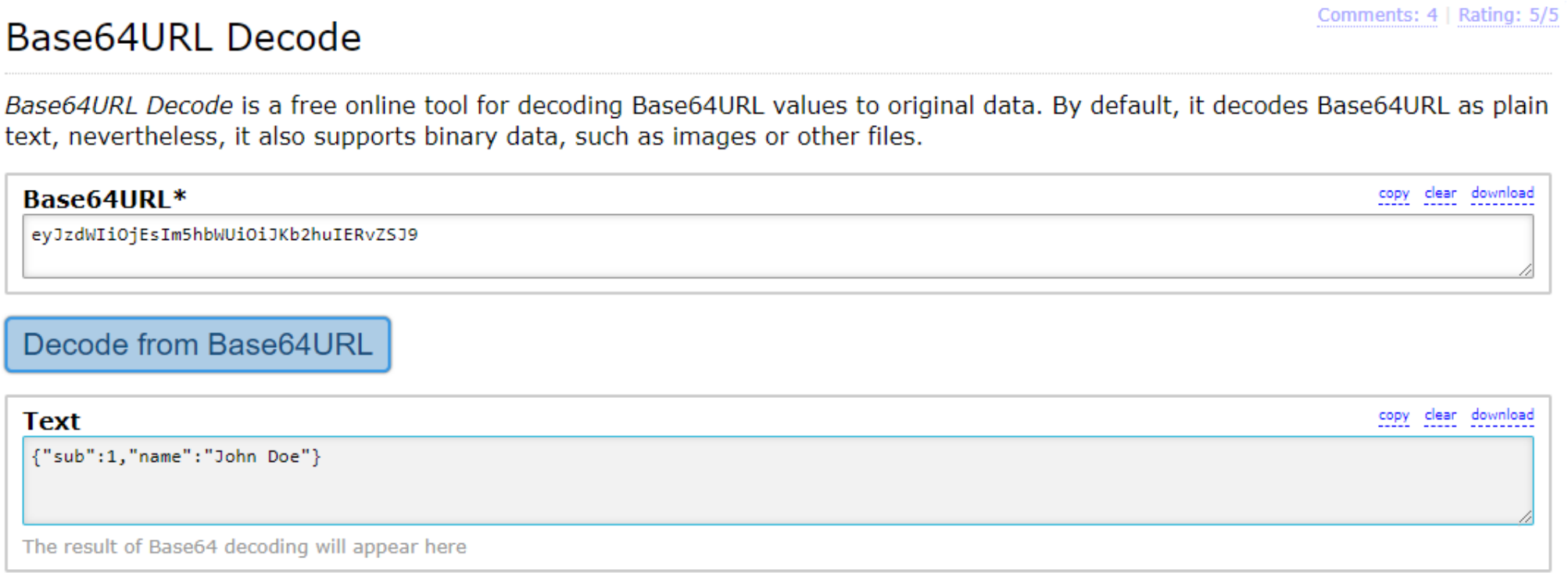

GET
http://localhost/hollingworthAPIs/12-create-RESTful-api/api/tasks
Send
Params
Authorization
Headers (12)
Body
Pre-request Script
Tests
Settings
Cookies
Type
Bearer T...
The authorization header will be automatically generated when you send the request. Learn more about authorization
Token
JWT
eeyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.e...
Body
Cookies
Headers (7)
Test Results
Status: 401 Unauthorized
Time: 25 ms
Size: 300 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
1 {
2 "message": "invalid signature"
3 }
Now sending a JWT with an invalid signature produces a 401 Unothorised status code.

Don't store sensative data in the JWT

The JWT access token contains three dot separated parts with the middle bit being the payload. The payload is encoded but not encrypted so it shoud not contain sensitive user data.

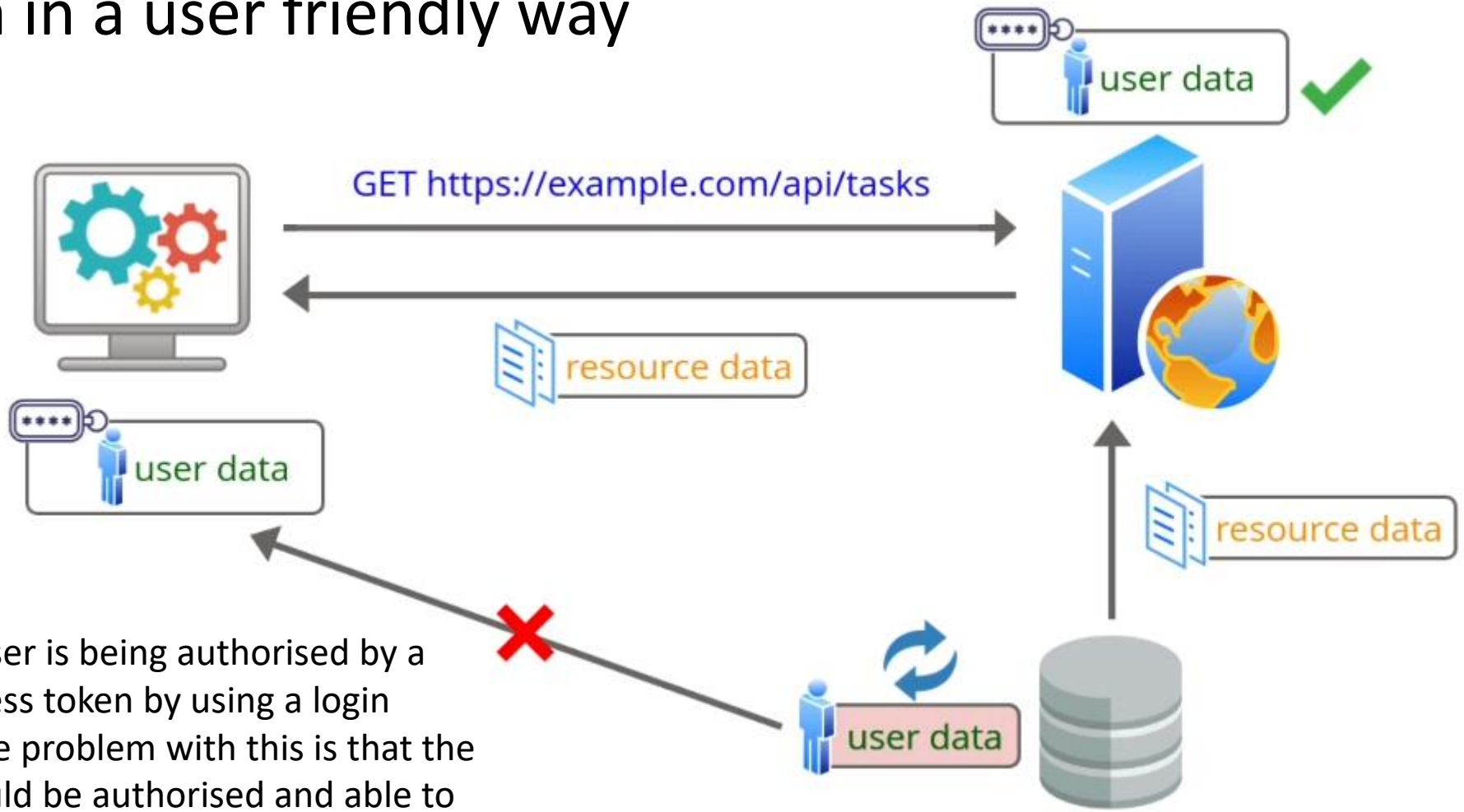

```
{
   "access_token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJzdWIiOjEsIm5hbWUiOiJKb2huIERvZSJ9.jU5F1in-3Qlbx_0ENhCkQ9_z8_uCC9SLag42XjKbG3w"
}
```


There are many Base64 decoders online so we can easily get the payload from the JWT access token. https://base64.guru/standards/base64url/decode



Expiring and Refreshing Access Tokens

Why access tokens need to expire and how to refresh them in a user friendly way

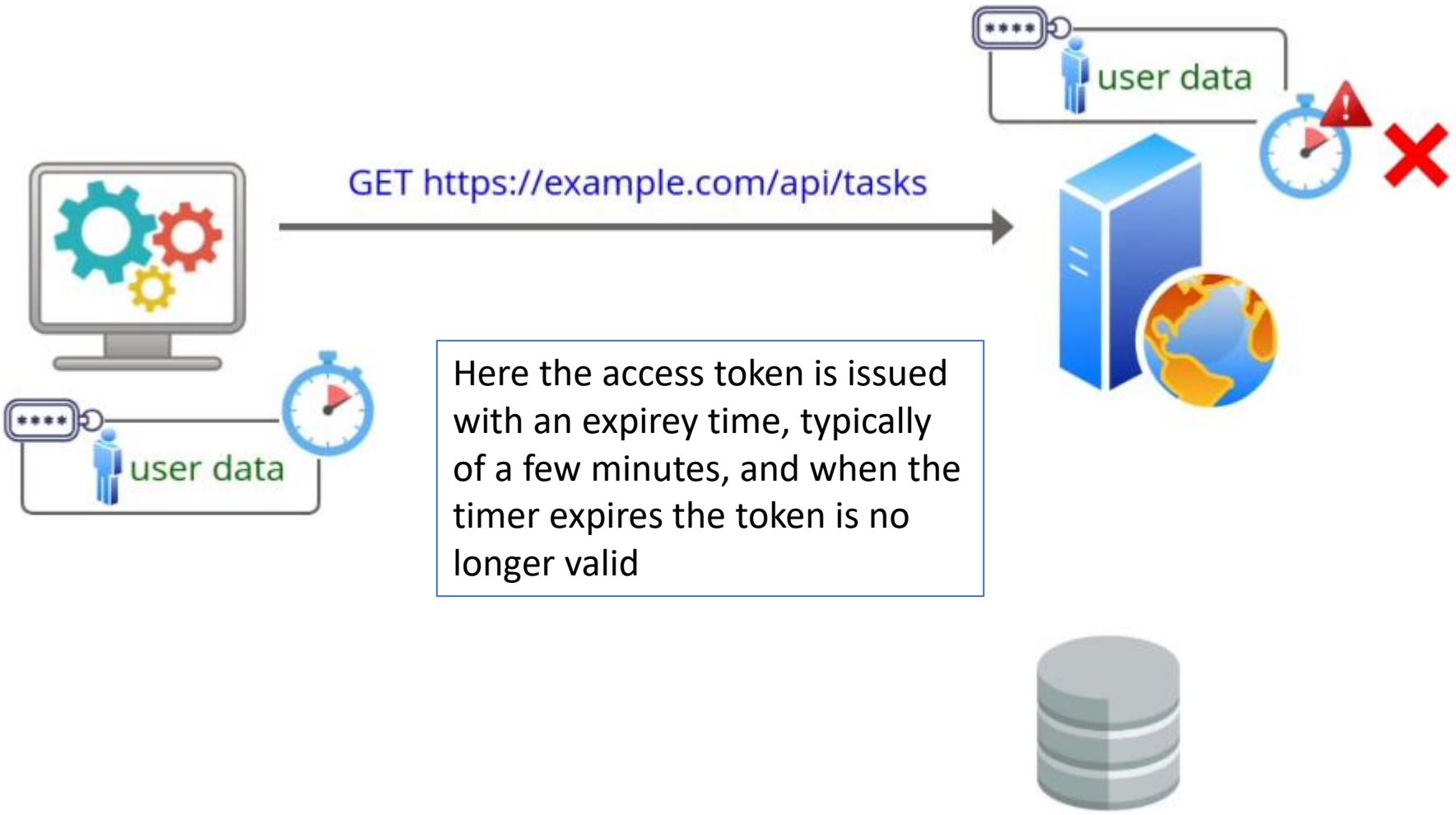


Now a user is being authorised by a JWT access token by using a login page. The problem with this is that the user would be authorised and able to use the JWT access token indefinately.

If a users priveleges or permissions chage in the databsse then they will still be able to access the API because the token is valid indefinitely.

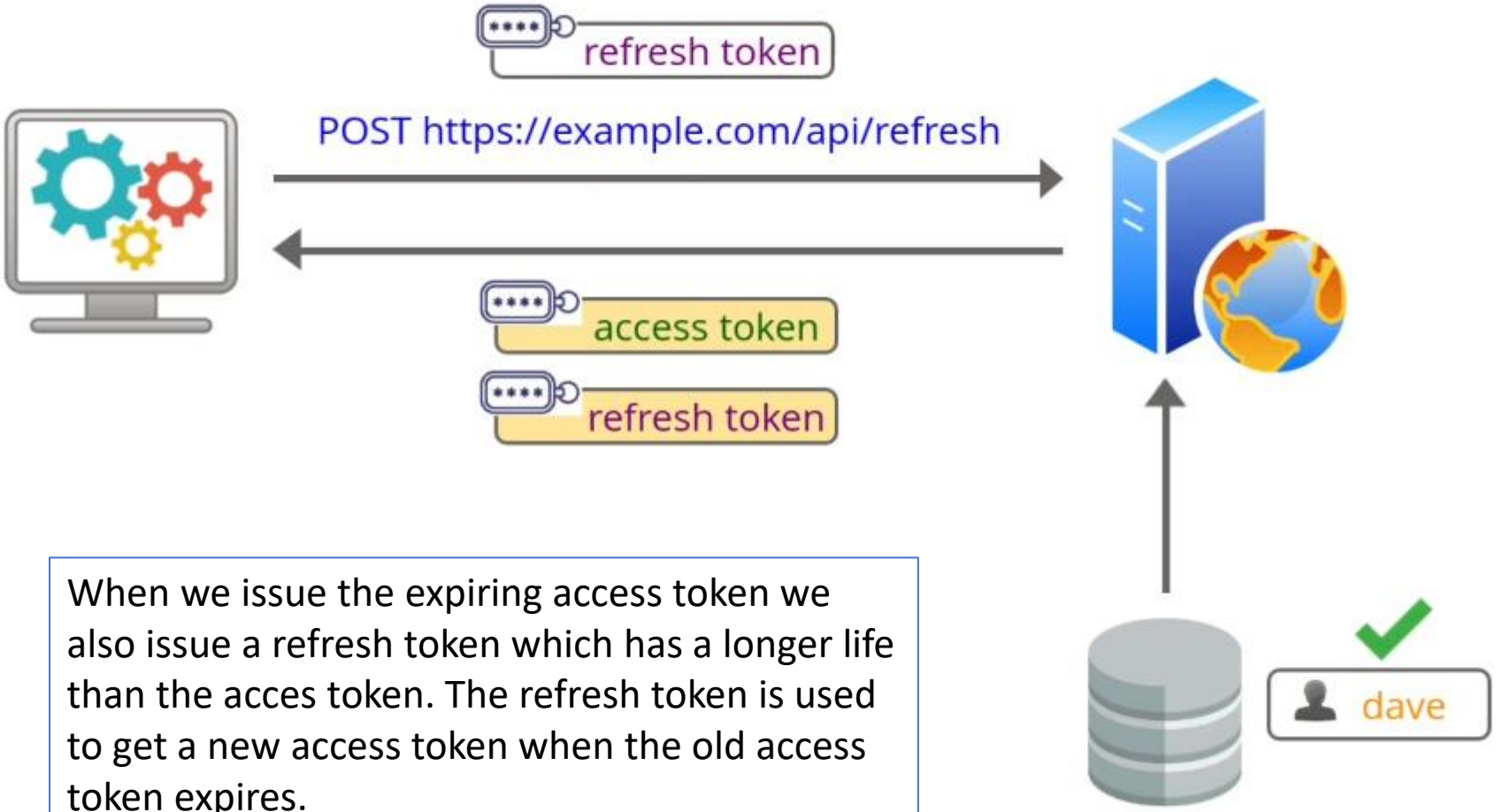
```
DB_HOST="localhost"
DB_NAME="hollingworthapis"
DB_USER="api_db_user"
DB_PASS="pa55word"

SECRET_KEY="5A7134743777217A25432A462D4A614E645267556B586E3272357538782F413F"
```

One way to invalidate a JWT access token would be to change the secrey key but this would invalidate JWTs for all users.



Here the access token is issued with an expirey time, typically of a few minutes, and when the timer expires the token is no longer valid

Once the access token has expired we could ask the user to login again and get a new token but this would not be vary user friendly.



When we issue the expiring access token we also issue a refresh token which has a longer life than the acces token. The refresh token is used to get a new access token when the old access token expires.

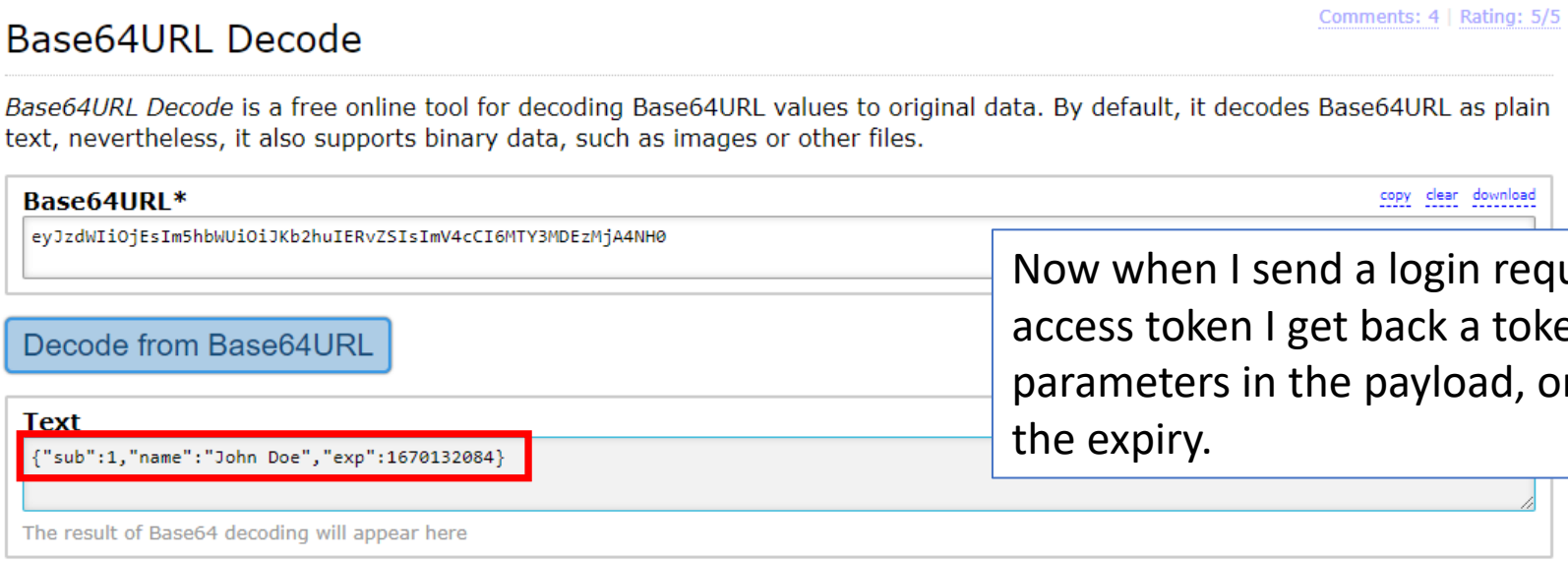
Add an expiry claim to the Access token payload when logging in

In the login.php script where the payload is generated an expiry claim is added to the payload as per the RFC 7519 standard https://datatracker.ietf.org/doc/html/rfc7519#section-4.1.4

```
$payload = [
    "sub" => $user["id"],
    "name" => $user["name"],
    "exp" => time() + 20
];
```

The expiry claim value is in seconds and would typically be set to 5 minutes (300 seconds). The value of the expiry claim is a unix timestamp, i.e. number of seconds from 1st January 1970 so we can use the php function of time() to get the current time stamp and add 300 seconds.
I have added 20 for testing purposes. https://www.php.net/manual/en/function.time.php

"access_token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.**eyJzdWIiOjEsIm5hbWUiOiJKb2huIERvZSIsImV4cCI6MTY3MDEzMjA4NH0**.-1PbmaNWSIk8g_8nkBa4eyZmzZqkGDTBwg5-eooP6GU"



Now when I send a login request ot get an access token I get back a token with three parameters in the payload, one of which is the expiry.

Throw a custom exception to not acept the JWT if it has Expired

```
$signature = hash_hmac("sha256", $matches["header"] . "." . $matches["payload"], $this->key, true);
$signature_from_token = $this->base64urlDecode($matches["signature"]);
if (!hash_equals($signature, $signature_from_token)) {
    throw new InvalidSignatureException;
}
$payload = json_decode($this->base64urlDecode($matches["payload"]),
if ($payload["exp"] < time()) {
    throw new TokenExpiredException;
}
return $payload;
```

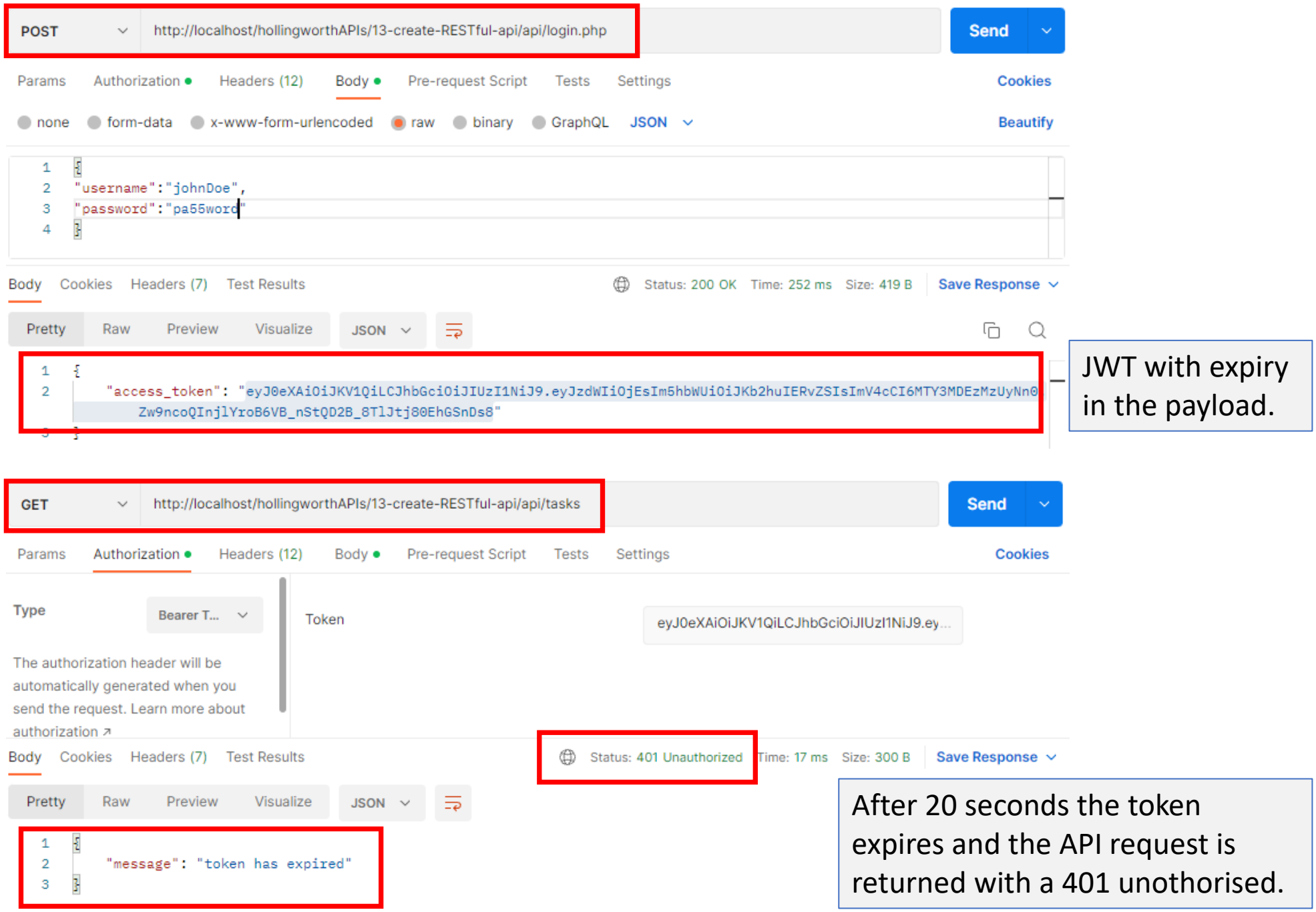
Now that the access token contains an expiration time we can check this in the codec when we decode the payload and throw a new exeption.

In the Auth.php script we can add another catch to handle the token expired exeption.

```
try {
   $data = $this->codec->decode($matches[1]);
}
catch (invalidSignatureException) {
    http_response_code(401);
    echo json_encode(["message" => "invalid signature"]);
    return false;
}
catch (TokenExpiredException) {
    http_response_code(401);
    echo json_encode(["message" => "token has expired"]);
    return false;
}
catch (Exception $e) {
    http_response_code(400);
    echo json_encode(["message" => $e->getMessage()]);
    return false;
}
$this->user_id = $data["sub"];
return true;
```

For simplicity we can add another php file to extend the Exception class. In a real world scenario we would organise these classes in namespaces.

```
<?php

class TokenExpiredException extends Exception {}
```


POST
http://localhost/hollingworthAPIs/13-create-RESTful-api/api/login.php
Send
Params
Authorization
Headers (12)
Body
Pre-request Script
Tests
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
{
"username":"johnDoe",
"password":"pa55word"
}
Body
Cookies
Headers (7)
Test Results
Status: 200 OK
Time: 252 ms
Size: 419 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
{
"access_token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJzdWIiOjEsIm5hbWUiOiJKb2huIERvZSIsImV4cCI6MTY3MDEzMzUyNn0
Zw9ncoQInjlYroB6VB_nStQD2B_8TlJtj80EhGSnDs8"
}
JWT with expiry in the payload.
GET
http://localhost/hollingworthAPIs/13-create-RESTful-api/api/tasks
Send
Params
Authorization
Headers (12)
Body
Pre-request Script
Tests
Settings
Cookies
Type
Bearer T...
Token
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.ey...
The authorization header will be automatically generated when you send the request. Learn more about authorization
Body
Cookies
Headers (7)
Test Results
Status: 401 Unauthorized
Time: 17 ms
Size: 300 B
Save Response
Pretty
Raw
Preview
Visualize
JSON
{
"message": "token has expired"
}
After 20 seconds the token expires and the API request is returned with a 401 unothorised.

Issue a refresh token in addition to the access token when logging in

```
$codec = new JWTCodec($_ENV["SECRET_KEY"]);
$access_token = $codec->encode($payload);
$refresh_token = $codec->encode(["sub" => $user["id"], "exp" => Time() + 432000]);
echo json_encode(["access_token" => $access_token, "refresh_token" => $refresh_token]);
```

In the login.php script we build the refresh token which contains the user Id (sub) and an expiry timestamp set to 5 days (432000 / 60 / 60 /24). We add the refresh token to the json encoded return.

{
"access_token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJzdWIiOjEsIm5hbWUiOiJKb2huIERvZSIsImV4cCI6MTY3MDEzNDUyNH0.6SAJ9PnihkZUkA69uvTaLeCuNlVWdac9wxEQcvyOpjs",
"refresh_token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJzdWIiOjEsImV4cCI6MTY3MDU2NjUwNH0.v5XRFAgT27dY2m7ylz8W8sb0Q4_LHKUTrXvi2k_SxIE"
}

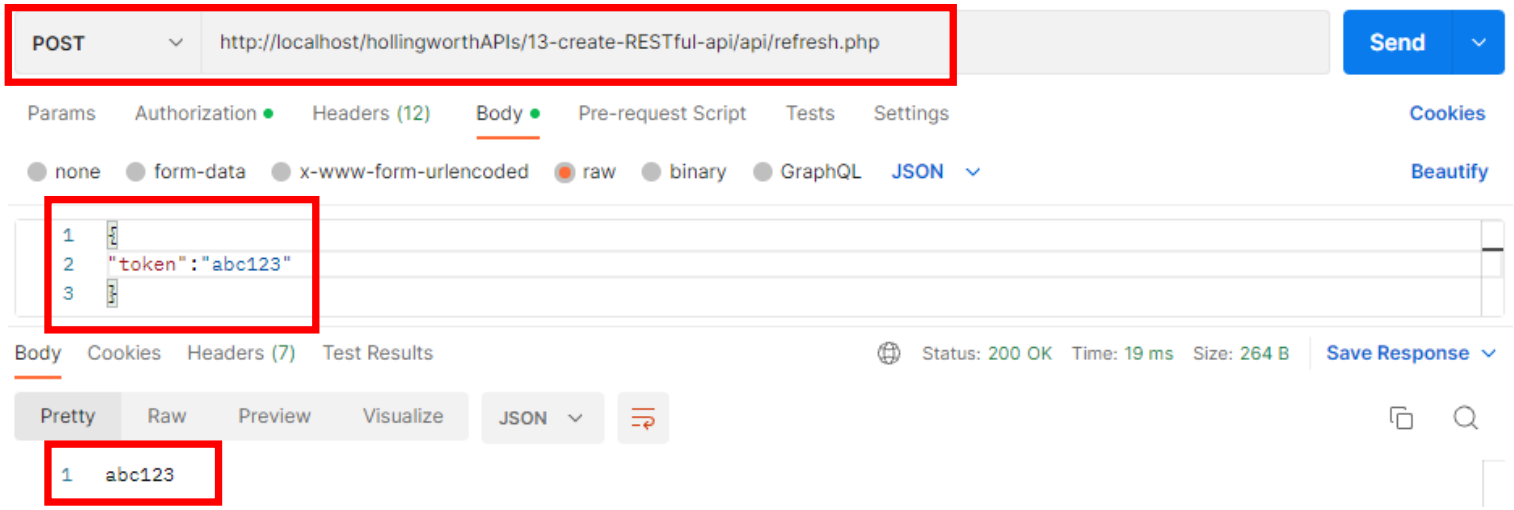
The payload of the refresh token being made of sub and exp.

Add a refresh endpoint and validate the refresh token in the request

A new file in the api folder called refresh.php is created and includes the type checking, bootstrap code and checks that method is POST. The next section of the code intercepts the refresh token.

Note that this script is not restful, we are calling it directly rather than doing any URL rewriting.

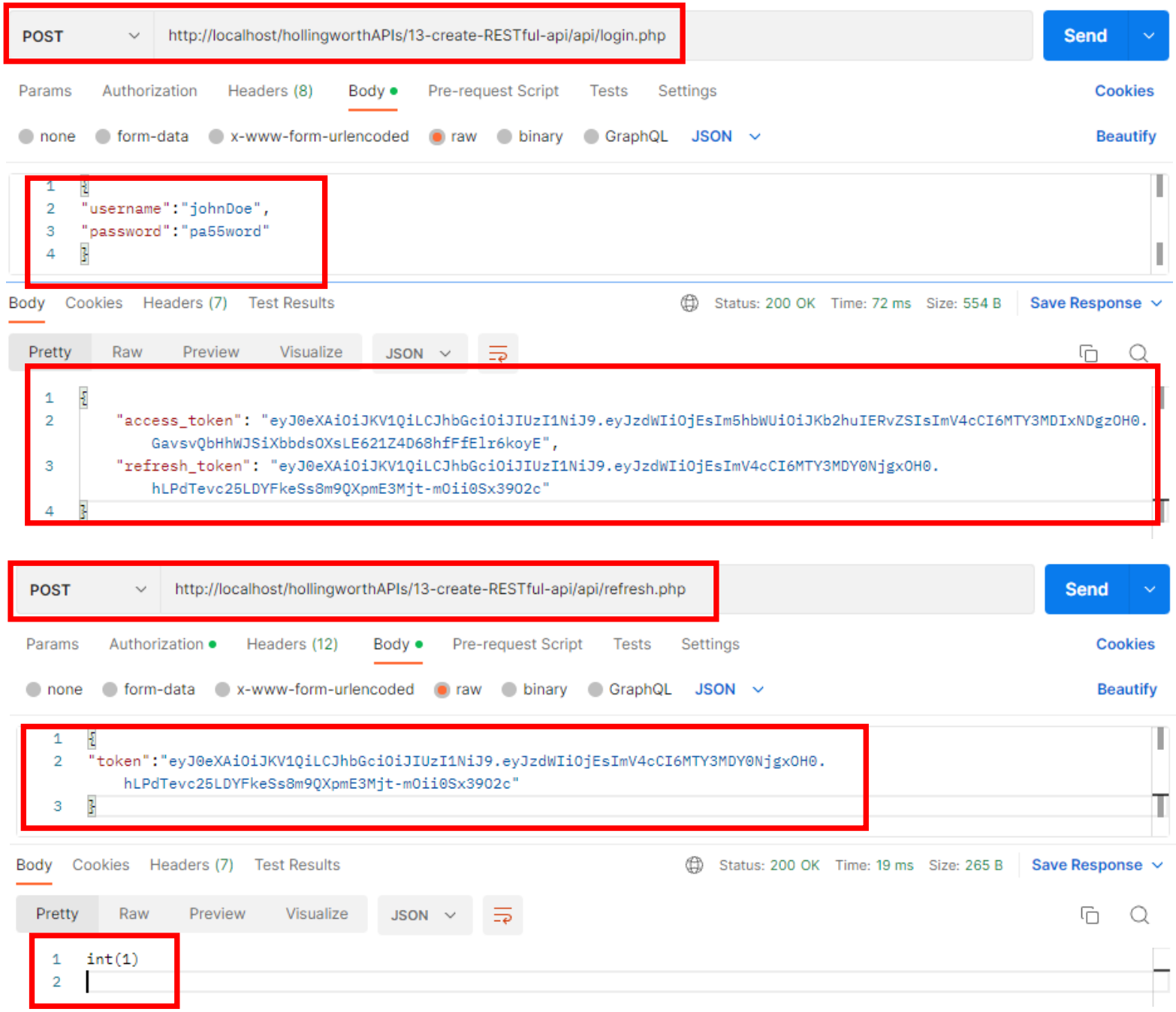
```php
<?php

declare(strict_types=1);
require __DIR__ . "/bootstrap.php";

if ($_SERVER["REQUEST_METHOD"] !== "POST") {
    http_response_code(405);
    header("Allow: POST");
    exit;
}

$data = (array) json_decode(file_get_contents("php://input"), true);
if (!array_key_exists("token", $data)) {
    http_response_code(400);
    echo json_encode(["message" => "missing token"]);
    exit;
}

echo $data["token"];
```



```php
<?php

declare(strict_types=1);
require __DIR__ . "/bootstrap.php";

if ($_SERVER["REQUEST_METHOD"] !== "POST") {
    http_response_code(405);
    header("Allow: POST");
    exit;
}

$data = (array) json_decode(file_get_contents("php://input"), true);
if (!array_key_exists("token", $data)) {
    http_response_code(400);
    echo json_encode(["message" => "missing token"]);
    exit;
}

$codec = new JWTCodec($_ENV["SECRET_KEY"]);
try {
    $payload = $codec->decode($data["token"]);
} catch (Exception) {
    http_response_code(400);
    echo json_encode(["message" => "invalid token"]);
    exit;
}

$user_id = $payload["sub"];
var_dump($user_id);
```

In the refresh.php script We need to decode the token using our codec code and handle any errors within a try catch block.

If the token is valid we can get the user_id from the payload.



When we make a login request we get an access token and a refresh token.

When we send the refresh token to the refresh endpoint (refresh.php) script we get the user ID.

If the refresh token signature is invalid then we get an invalid token message and 400 Bad Request.

Validate the user in the refresh token using the database

In the UserGateway.php class we create a method to find user by ID and return false or an array of user data.

```
public function getByid(int $id): array | false {
    $sql = "SELECT * FROM user WHERE id=:id";
    $stmt = $this->conn->prepare($sql);
    $stmt->bindValue(":id", $id, PDO::PARAM_STR);
    $stmt->execute();
    return $stmt->fetch(PDO::FETCH_ASSOC);
}
```

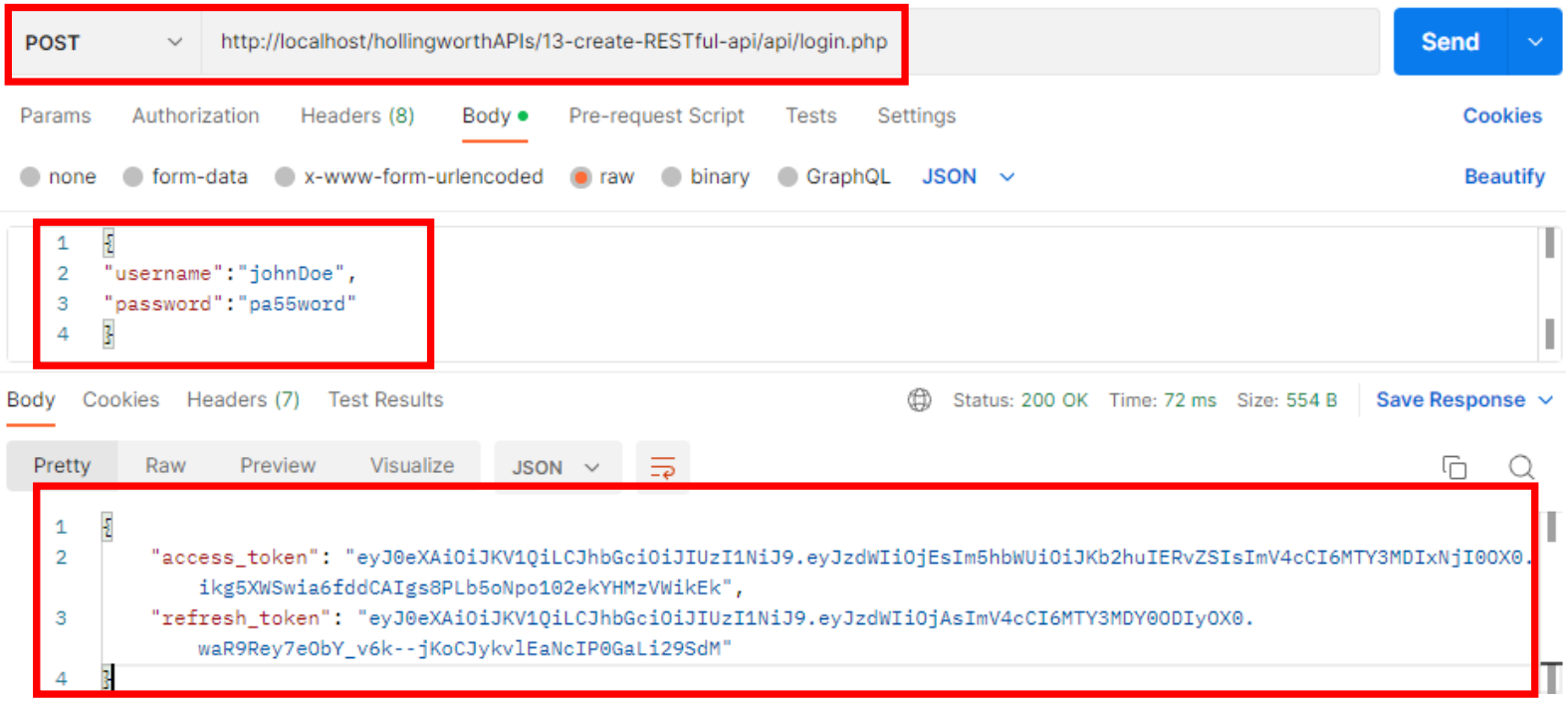
```
$user_id = $payload["sub"];

$database = new Database($_ENV["DB_HOST"], $_ENV["DB_NAME"],
$_ENV["DB_USER"], $_ENV["DB_PASS"]);
$user_gateway = new UserGateway($database);
$user = $user_gateway->getById($user_id);
if ($user === false) {
    http_response_code(401);
    echo json_encode(["message" => "invalid authentication"]);
    exit;
}
var_dump($user);
```

In the refresh.php script we can call this method with a new instance of the database and user gateway. If no user is found, I.e. false we output an appropriate message.

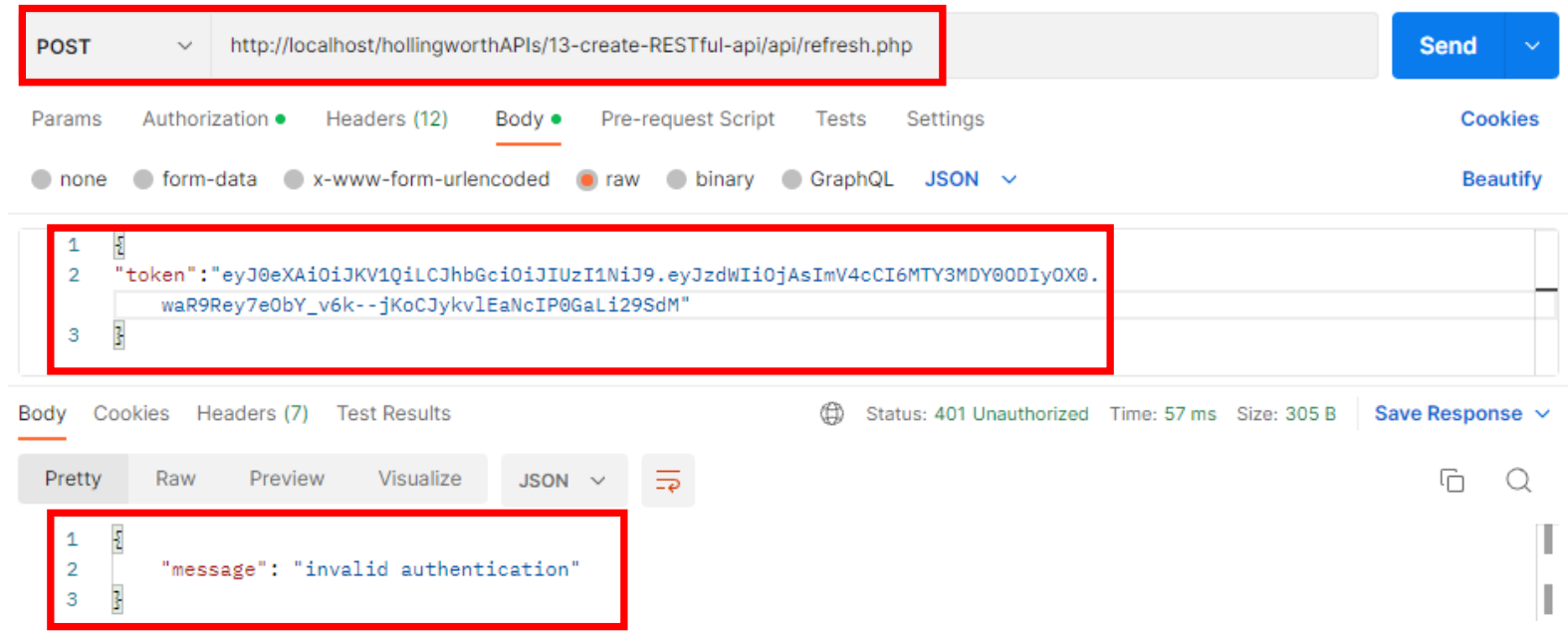
Otherwise for now we dump the user to screen.

In login.php the user ID in the refresh token is temporarily modified to zero to test this.

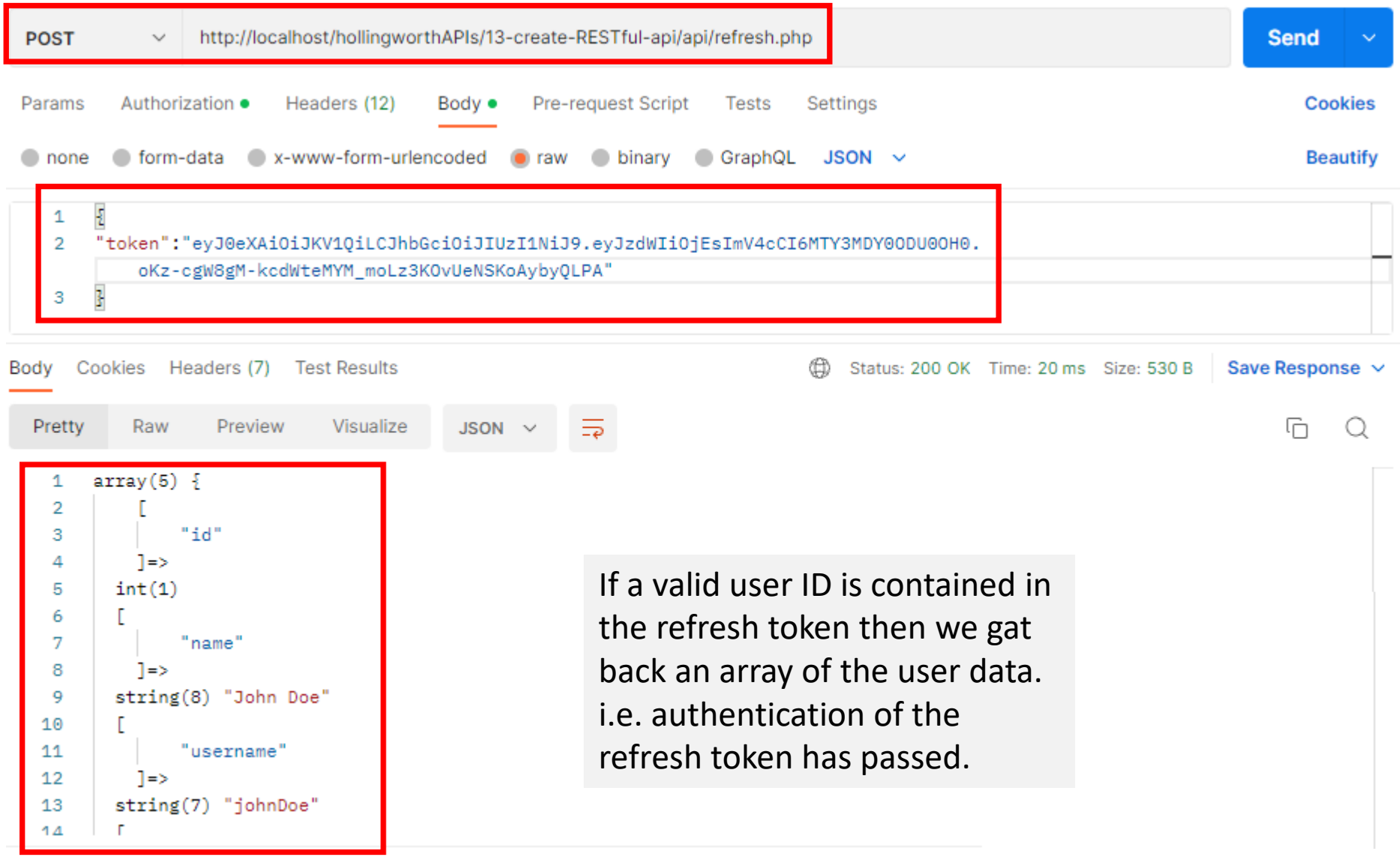
```
$codec = new JWTCodec($_ENV["SECRET_KEY"]);
$access_token = $codec->encode($payload);
$refresh_token = $codec->encode(["sub" => 0 /*$user["id"]*/, "exp" => time() + 432000]);
echo json_encode(["access_token" => $access_token, "refresh_token" => $refresh_token]);
```



Now the refresh token has an invalid signature because the user ID is not in the database.



When we send the bogus refresh token to the refresh endpoint (refresh.php) script we get 401 unathorised.



If a valid user ID is contained in the refresh token then we gat back an array of the user data. i.e. authentication of the refresh token has passed.

Issue a new access token and refresh token to the authenticated user

id	name	username	password_hash	api_key	is_active
1	John Doe	johnDoe	$2y$10$/oYhMdqEBn1xp8oQUCNXm.CSrwKW5g609T.lioVhbim...	22fe8690113a0dc42edc9a4e59f0d9b3	1
3	Mary Smith	MarySmith	$2y$10$GVE5QbpyicypG4nlm6MgJumtkmiwOZ2lyfhxKlnJORI...	0a37d180f691bc45600e4a1266986293	0

I have added a new column called is_active which is Boolean 1 for account active or Boolean 0 for account unsubscribed.

```
$database = new Database($_ENV["DB_HOST"], $_ENV["DB_NAME"],
$_ENV["DB_USER"], $_ENV["DB_PASS"]);
$user_gateway = new UserGateway($database);
$user = $user_gateway->getById($user_id);
if ($user === false) {
    http_response_code(401);
    echo json_encode(["message" => "invalid authentication"]);
    exit;
}
if ($user["is_active"] === 0) {
    http_response_code(401);
    echo json_encode(["message" => "user account unsusbscribed"]);
    exit;
}
```

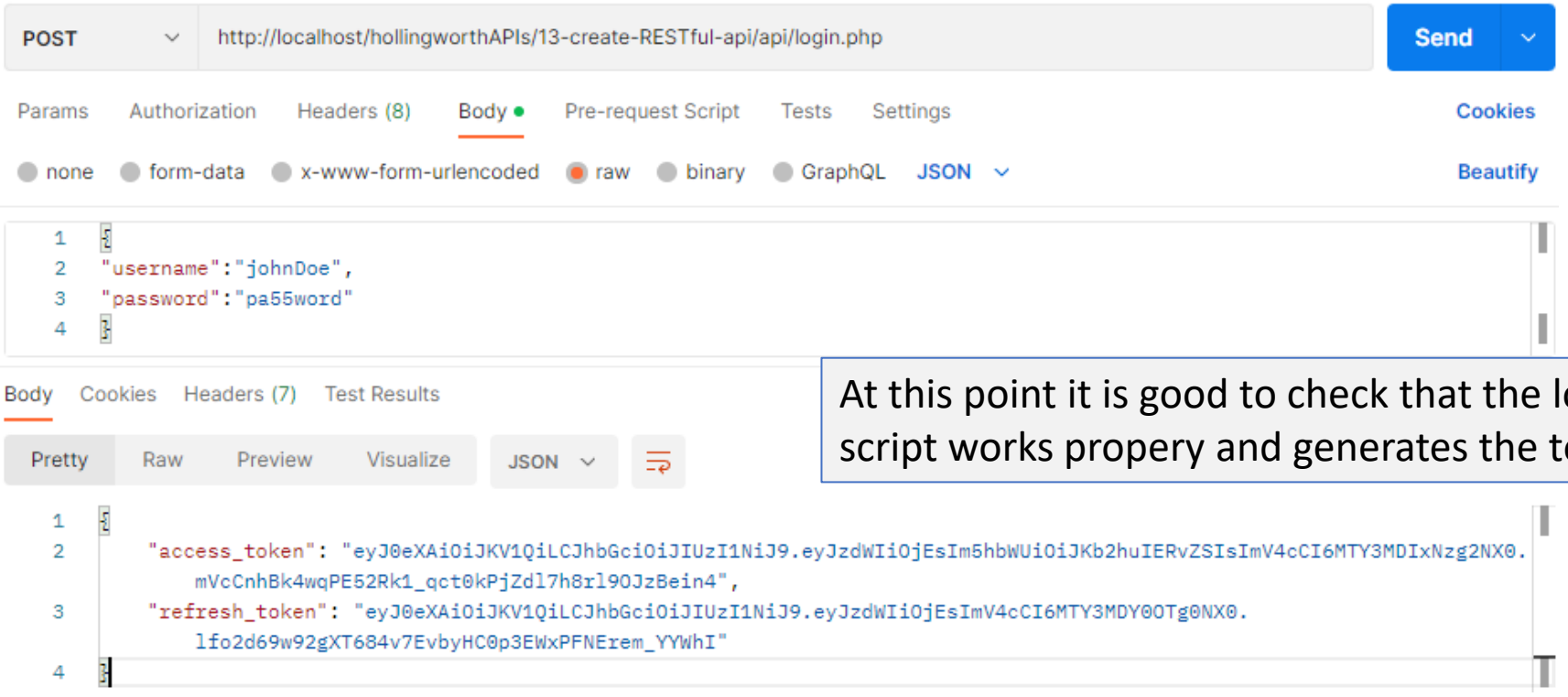
I can no check tha the user is subscribed and issue the appropriate status code and error message.

```php
<?php

declare(strict_types=1);
require __DIR__ . "/bootstrap.php";

if ($_SERVER["REQUEST_METHOD"] !== "POST") {
    http_response_code(405);
    header("Allow: POST");
    exit;
}

$data = (array) json_decode(file_get_contents("php://input"), true);
if (!array_key_exists("username", $data) || !array_key_exists("password", $data)) {
    http_response_code(400);
    echo json_encode(["message" => "missing login credentials"]);
    exit;
}

$database = new Database($_ENV["DB_HOST"], $_ENV["DB_NAME"], $_ENV["DB_USER"], $_ENV["DB_PASS"]);
$user_gateway = new UserGateway($database);
$user = $user_gateway->getByUsername($data["username"]);
if ($user === false) {
    http_response_code(401);
    echo json_encode(["message" => "invalid authentication"]);
    exit;
}

if (!password_verify($data["password"], $user["password_hash"])) {
    http_response_code(401);
    echo json_encode(["message" => "invalid authentication"]);
    exit;
}

$codec = new JWTCodec($_ENV["SECRET_KEY"]);
require __DIR__ . "/tokens.php";
```

In login.php script we will do some refactoring by removing the part where we generate tokens. The codec instantiation will be moved up and we will create a new file in the app folder called tokens.php which we will require in the login script.

```php
<?php

$payload = [
    "sub" => $user["id"],
    "name" => $user["name"],
    "exp" => time() + 20
];


$access_token = $codec->encode($payload);
$refresh_token = $codec->encode(["sub" => $user["id"], "exp" => time() + 432000]);
echo json_encode(["access_token" => $access_token, "refresh_token" => $refresh_token]);
```

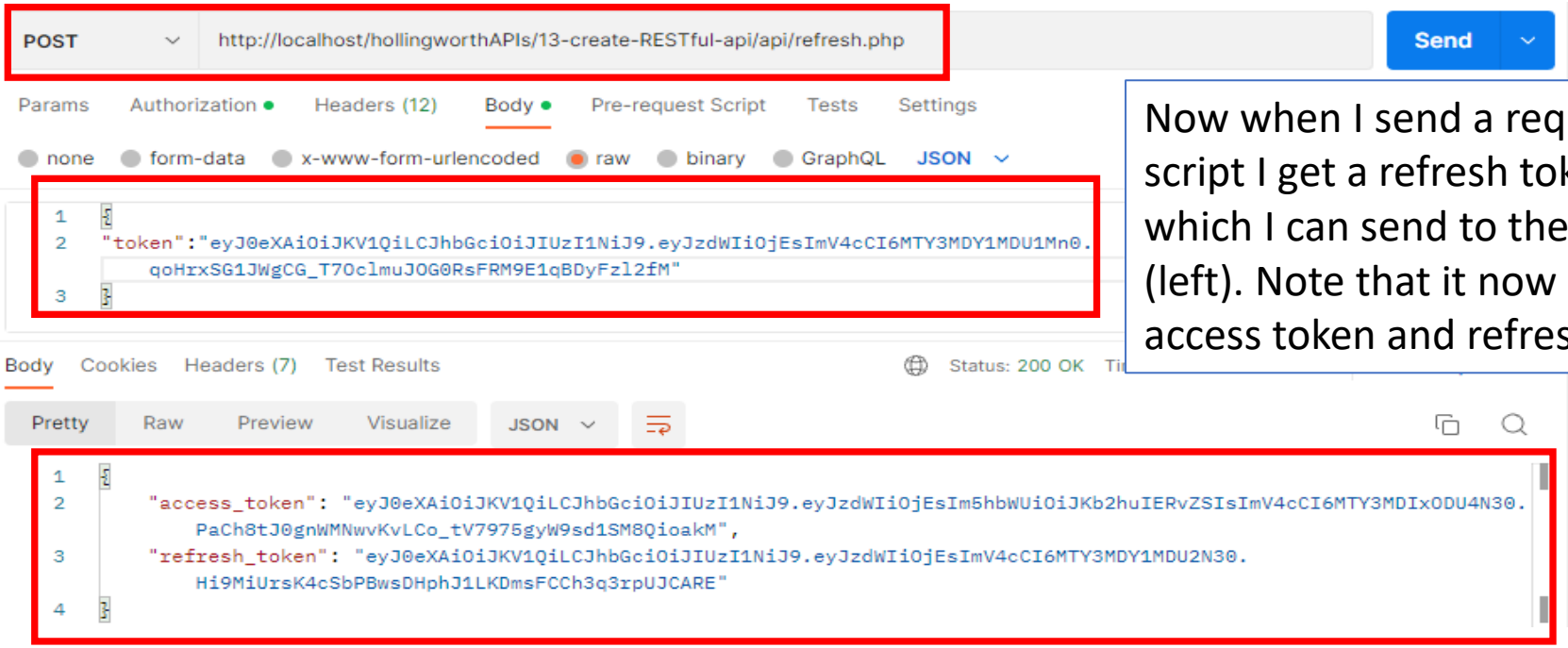
In tokens.php script we have the code that generates the tokens.



At this point it is good to check that the login script works propery and generates the tokens.

```php
<?php
. . .
$database = new Database($_ENV["DB_HOST"], $_ENV["DB_NAME"], $_ENV["DB_USER"], $_ENV["DB_PASS"]);
$user_gateway = new UserGateway($database);
$user = $user_gateway->getById($user_id);
if ($user === false) {
    http_response_code(401);
    echo json_encode(["message" => "invalid authentication"]);
    exit;
}
if ($user["is_active"] === 0) {
    http_response_code(401);
    echo json_encode(["message" => "user account unsusbscribed"]);
    exit;
}

require __DIR__ . "/tokens.php";
```

In the refresh script instead of dumping the user data to the screen we can require the tokens script (tokens.php).



Now when I send a request to the login script I get a refresh token (not shown) which I can send to the refresh script (left). Note that it now returns a new access token and refresh token.

Create a table to store a refresh token whitelist

The access token has a short lifespan so we can periodically check the database for changes to the user account i.e account unsubscribed. Having a short lifespan also makes it more secure in case it is stolen by an attacker. The refresh token has a longer lifespan and so if is compromised it could be used to obtain more access tokens for for a longer period of time.

One way to mitigate this is to create a whitelist of refresh tokens. whenever we issue a refresh token we store it in a database. When we process a refresh token we also check it in the database.

```
CREATE TABLE refresh_token (
token_hash VARCHAR(64) NOT NULL,
expires_at INT UNSIGNED NOT NULL,
PRIMARY KEY (token_hash),
INDEX (expires_at)
);
```

The new whitelist table contains two columns, the first the token hash is a 64 character string and is the primary key. The second column is an expires at interger that is unsigned and indexed.

Store the refresh token in the whitelist ehen issued in the refresh endpoint

```php
<?php

class RefreshTokenGateway {
    private PDO $conn;
    private string $key;

    public function __construct(Database $database, string $key) {
        $this->conn = $database->getConnection();
        $this->key = $key;
    }

    public function create(string $token, int $expiry): bool {
        $hash = hash_hmac("sha256", $token, $this->key);
        $sql = "INSERT INTO refresh_token (token_hash, expires_at) VALUES (:token_hash, :expires_at)";
        $stmt = $this->conn->prepare($sql);
        $stmt->bindValue(":token_hash", $hash, PDO::PARAM_STR);
        $stmt->bindValue(":expires_at", $expiry, PDO::PARAM_INT);
        return $stmt->execute();
    }
}
```

In the src folder we need a new class to handle interactions with the whitelist table.

We have private variables for the database connection and the secret key which we will declare in the constructor.

The create method takes a token and an expiry and inserts them into the database refresh_token table, first hashing the token.

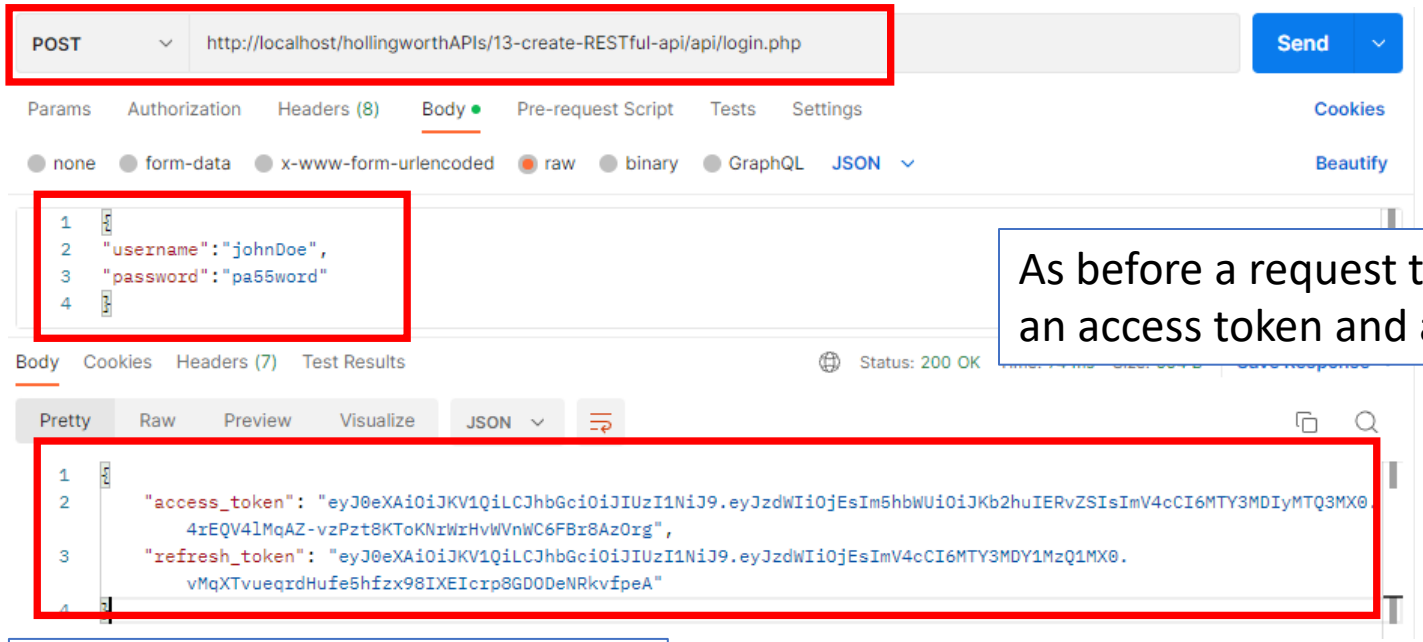
We wil return the result of the execute statement which is Boolean.

In the tokens.php script when we create the refresh token we can define the expiry as a variable and use it in the token.

```
$refresh_token_expiry = time() + 432000;
$access_token = $codec->encode($payload);
$refresh_token = $codec->encode(["sub" => $user["id"], "exp" => $refresh_token_expiry]);
echo json_encode(["access_token" => $access_token, "refresh_token" => $refresh_token]);
```

In the login.php script we can instantiate the refresh tokens gateway class and call the create method with the two variables of the refresh token and the refresh token expiry.

```
<?php

. . .

require __DIR__ . "/tokens.php";

$refresh_token_gateway = new RefreshTokenGateway($database, $_ENV["SECRET_KEY"]);
$refresh_token_gateway->create($refresh_token, $refresh_token_expiry);
```



As before a request to to the login script returns an access token and a refresh token.

The whitelist table also has the refresh token record.

token_hash	expires_at
338d77fd35e7cec1113c3dd83ceaa77543777ecf4136220cab...	1670653451

Replace the refresh token in the whitelist when issued in the refresh endpoint

```
public function delete(string $token): int {
    $hash = hash_hmac("sha256", $token, $this->key);
    $sql = "DELETE FROM refresh_token WHERE token_hash = :token_hash";
    $stmt = $this->conn->prepare($sql);
    $stmt->bindValue(":token_hash", $hash, PDO::PARAM_STR);
    $stmt->execute();
    return $stmt->rowCount();
}
```

In the tokenHashGateway.php class we need a method to remove the token.

In the refresh.php script we instantiate the token gateway class and use the delete method to remove the token from the data variable (found in the tokens.php script) then create a new token. Efectively swapping out the old refresh token for the new one.

```
<?php

. . .

require __DIR__ . "/tokens.php";

$refresh_token_gateway = new RefreshTokenGateway($database, $_ENV["SECRET_KEY"]);
$refresh_token_gateway->delete($data["token"]);
$refresh_token_gateway->create($refresh_token, $refresh_token_expiry);
```

POST http://localhost/hollingworthAPIs/13-create-RESTful-api/api/login.php Send

Params Authorization Headers (8) Body Pre-request Script Tests Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL JSON Beautify

```
{
"username":"johnDoe",
"password":"pa55word"
}
```

Body Cookies Headers (7) Test Results Status: 200 OK Time: 73 ms Size: 554 B Save Response

Pretty Raw Preview Visualize JSON

```
{
    "access_token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJzdWIiOjEsIm5hbWUiOiJKb2huIERvZSIsImV4cCI6MTY3MDIyMzM0NX0.
        Bh5GO2WJQwjJzn-5XS0Z_0xAWZPSCOnYy_rv29o5y0Y",
    "refresh_token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJzdWIiOjEsImV4cCI6MTY3MDY1NTA0NX0.
        bx8W3dds7Ww-oiPRyscqmqWU9maLu7fU1Jk1vui-Lds"
}
```

In the login script we make a request and get an access token and a refresh token. The refresh token is put in the whitelist table of the database.

token_hash	expires_at
4d74619e320ee6428b4204f564567f189f578c3cc908d37f80...	1670655045

POST http://localhost/hollingworthAPIs/13-create-RESTful-api/api/refresh.php Send

Params Authorization Headers (12) Body Pre-request Script Tests Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL JSON Beautify

```
{
"token":"eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJzdWIiOjEsImV4cCI6MTY3MDY1NTA0NX0.
    bx8W3dds7Ww-oiPRyscqmqWU9maLu7fU1Jk1vui-Lds"
}
```

Body Cookies Headers (7) Test Results Status: 200 OK Time: 75 ms Size: 554 B Save Response

Pretty Raw Preview Visualize JSON

```
{
    "access_token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJzdWIiOjEsIm5hbWUiOiJKb2huIERvZSIsImV4cCI6MTY3MDIyMzU3OH0.
        oGYp9UtWvrGJVZnWmy_iEcayN9p-FPQ9v9DcHj4AE0k",
    "refresh_token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJzdWIiOjEsImV4cCI6MTY3MDY1NTI3OH0.
        aL2x9Go2YSHp0rIDzwYlEq-rnhVGKUZjYC7TpXq4WAA"
}
```

Now making a request to the refresh script we get a new access token and new refresh token.

In the database there is still only one token but it is the the new token.

token_hash	expires_at
ef5e55e90ed744fa62790d55053a74f2ded058a918af5dae79...	1670655278

Validate the refresh token is on the whitelist and return 400 response if not

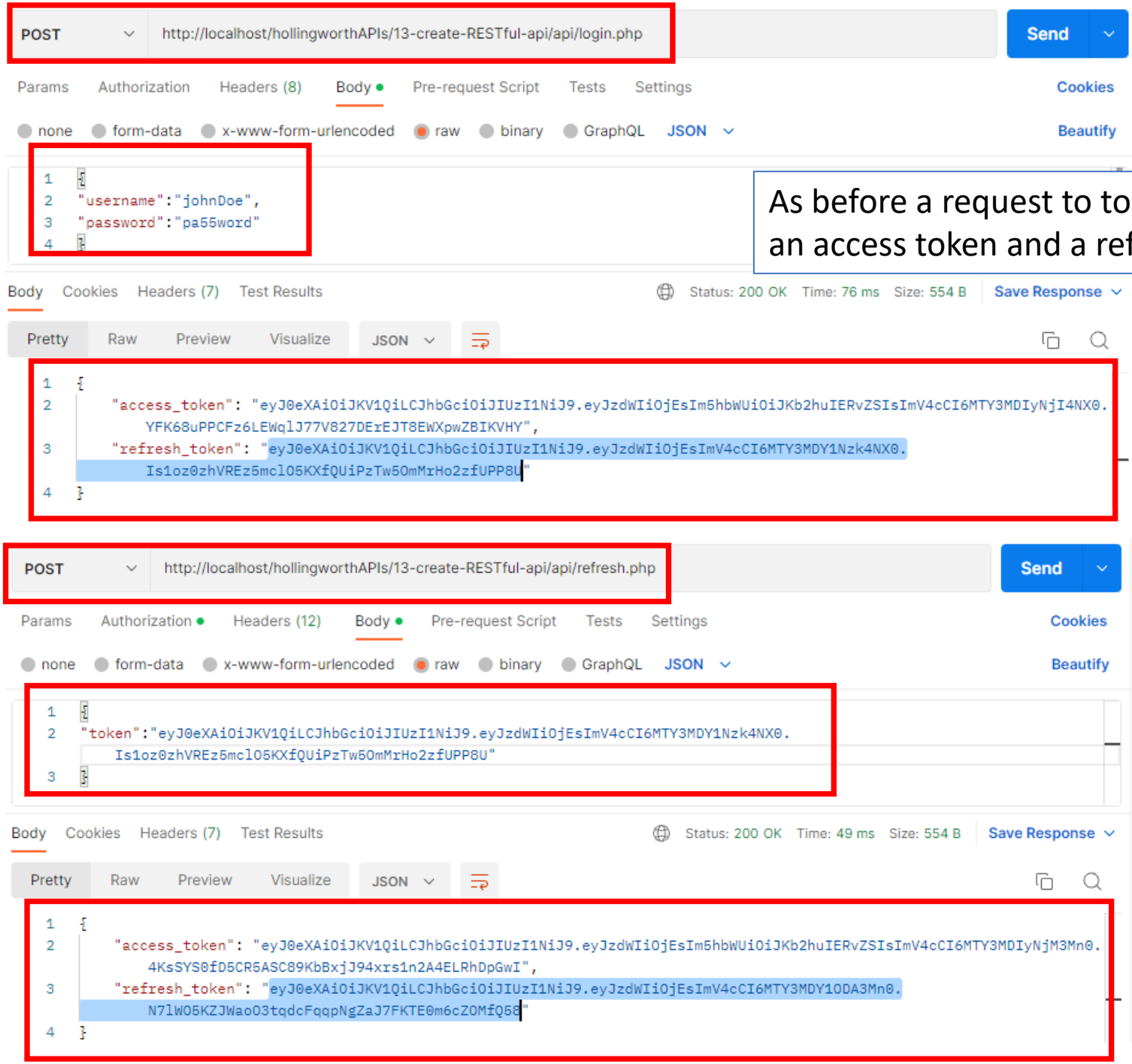
```
public function getByToken($token) : array | false {
    $hash = hash_hmac("sha256", $token, $this->key);
    $sql = "SELECT FROM refresh_token WHERE token_hash = :token_hash";
    $stmt = $this->conn->prepare($sql);
    $stmt->bindValue(":token_hash", $hash, PDO::PARAM_STR);
    $stmt->execute();
    return $stmt->fetch(PDO::FETCH_ASSOC);
}
```

In the tokenHashGateway.php class we need a method to get the token data array or a false.

In the refresh script we need to call the refresh token gateway class after the database connection and pass in the data[“token”] value into the getByToken method.

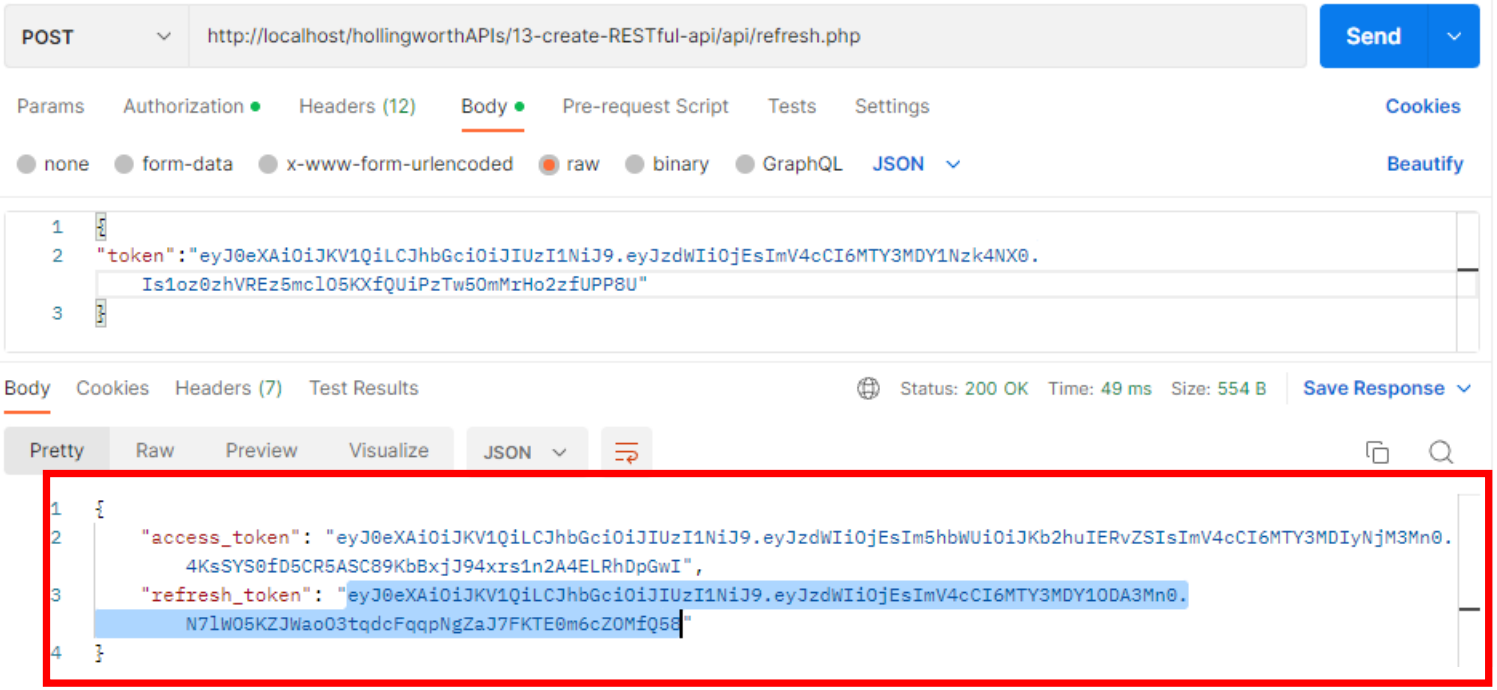
```
<?php

. . .

$database = new Database($_ENV["DB_HOST"], $_ENV["DB_NAME"], $_ENV["DB_USER"], $_ENV["DB_PASS"]);

$refresh_token_gateway = new RefreshTokenGateway($database, $_ENV["SECRET_KEY"]);
$refresh_token = $refresh_token_gateway->getByToken($data["token"]);
if ($refresh_token === false) {
    http_response_code(400);
    echo json_encode(["message" => "invalid token (not on whitelist)"]);
    exit;
}

. . .

require __DIR__ . "/tokens.php";
$refresh_token_gateway->delete($data["token"]);
$refresh_token_gateway->create($refresh_token, $refresh_token_expiry);
```

If this function returns false then it is not on the whitelist.

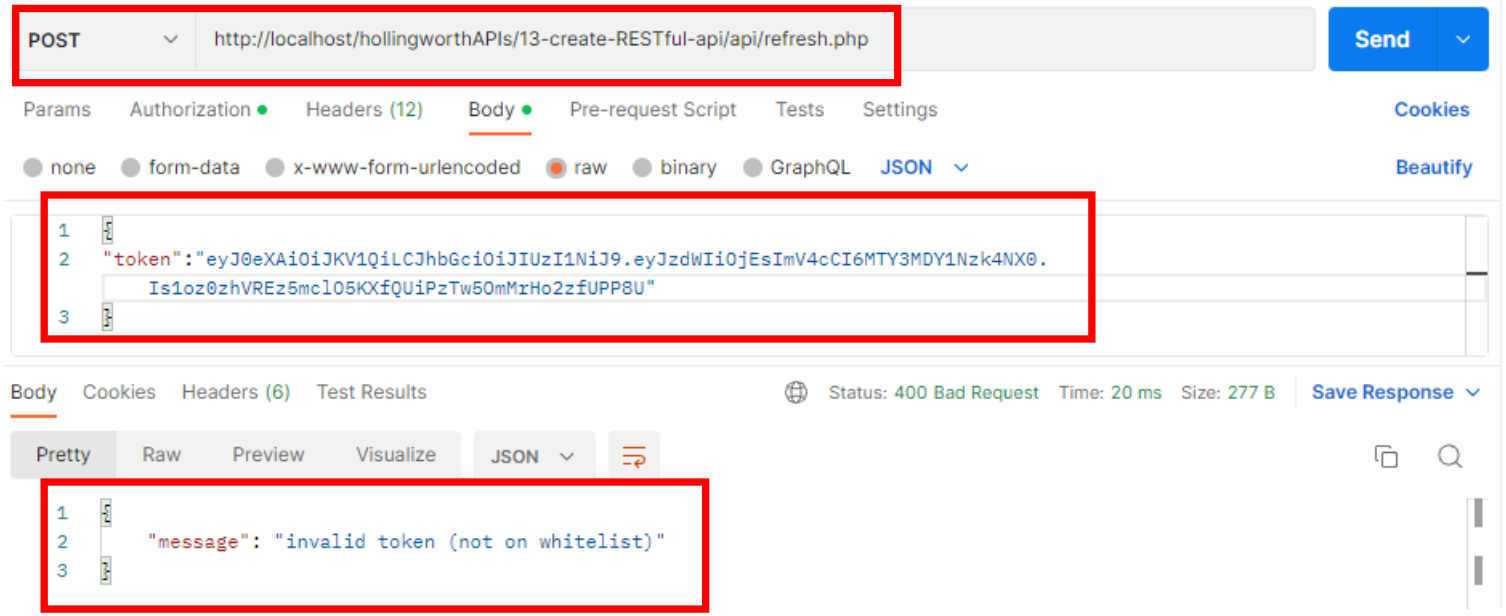


As before a request to to the login script returns an access token and a refresh token.

This refresh token can be used to get a new access token and new refresh token.



The higlighte refresh token has replaced the one we sent in the whitelist.



If we send the same refresh token as before then it is invalid, not on whitelist. Only the most recent request token is saved on the white list.

Add a logout endpoint to remove the an active refresh token from the whitelist

```php
<?php

declare(strict_types=1);
require __DIR__ . "/bootstrap.php";

if ($_SERVER["REQUEST_METHOD"] !== "POST") {
    http_response_code(405);
    header("Allow: POST");
    exit;
}

$data = (array) json_decode(file_get_contents("php://input"), true);
if ( ! array_key_exists("token", $data)) {
    http_response_code(400);
    echo json_encode(["message" => "missing token"]);
    exit;
}

$codec = new JWTCodec($_ENV["SECRET_KEY"]);
try {
    $payload = $codec->decode($data["token"]);
} catch (Exception) {
    http_response_code(400);
    echo json_encode(["message" => "invalid token"]);
    exit;
}

$database = new Database($_ENV["DB_HOST"], $_ENV["DB_NAME"], $_ENV["DB_USER"], $_ENV["DB_PASS"]);

$refresh_token_gateway = new RefreshTokenGateway($database, $_ENV["SECRET_KEY"]);
$refresh_token_gateway->delete($data["token"]);
```

When we issue a new refresh token we save it in the whitelist on the server replacing any existing one. A client using this API when logging out would have the access token and refresh token removed.

The logout.php script in the API folder is a copy of the login script but with bits removed.

Note that the logout script does not destroy a session but is another API enpoint. In this case it checks the access token is valid then deletes the refresh token if it exists.

POST http://localhost/hollingworthAPIs/13-create-RESTful-api/api/login.php Send

Params Authorization Headers (8) Body Pre-request Script Tests Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL JSON Beautify

```
{
"username":"johnDoe",
"password":"pa55word"
}
```

Body Cookies Headers (7) Test Results Status: 200 OK Time: 76 ms Size: 554 B Save Response

Pretty Raw Preview Visualize JSON

```
{
    "access_token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJzdWIiOjEsIm5hbWUiOiJKb2huIERvZSIsImV4cCI6MTY3MDIyODAyOH0.
        vTf1Nz8a05zgOc80NLCn6ZWJ3Qcv_r-Cm_6Be-5Icic",
    "refresh_token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJzdWIiOjEsImV4cCI6MTY3MDY1OTcyOH0.
        gjzJFtJYMr1SiMA9w4B55fRSBZ6fH2YIyfaSV_bzwnw"
}
```

When we login we get a refresh token which is stored in the whitelist.

token_hash	expires_at
4c525b93f6c93d69293d507658cf7746dc8f937b9011418453...	1670659492

POST http://localhost/hollingworthAPIs/13-create-RESTful-api/api/logout.php Send

Params Authorization Headers (8) Body Pre-request Script Tests Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL JSON Beautify

```
{
"token":"eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJzdWIiOjEsImV4cCI6MTY3MDY1OTcyOH0.
    gjzJFtJYMr1SiMA9w4B55fRSBZ6fH2YIyfaSV_bzwnw"
}
```

Body Cookies Headers (7) Test Results Status: 200 OK Time: 20 ms Size: 258 B Save Response

Pretty Raw Preview Visualize JSON

```
1
```

When we logout and pass in the refresh token this token is deleted from the whiteleist table.

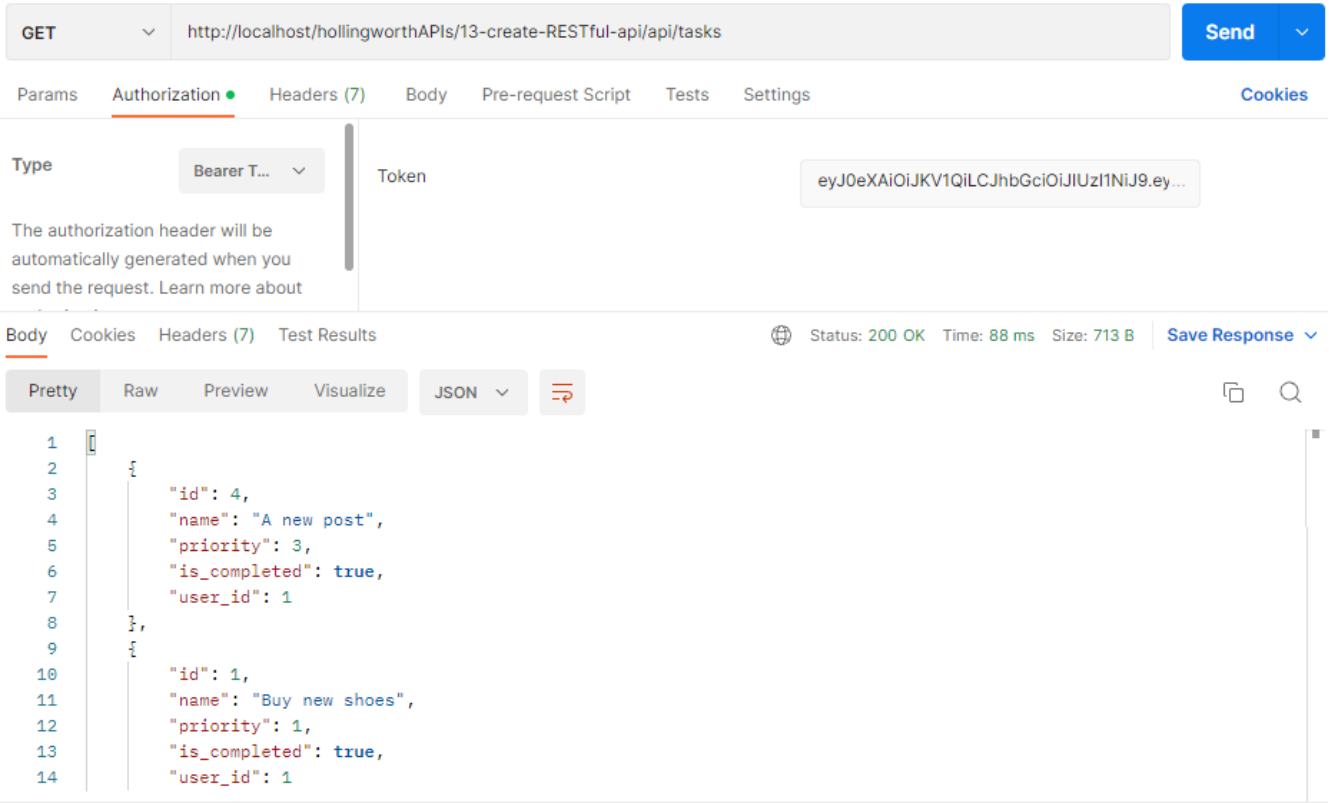
token_hash	expires_at

When we login we get an access token and a refresh token. The access token is valid for a short period of time until it expires. We can access resources with the access token until it expires where we will recive a token has expired message.

Once the access token has expired we can send the refresh token to the refresh API endpoint and get a new access token which we can use to continue accessing resources.

If the refresh token has expired (life of 5 days) or the user has logged out, then the user will have to login again.

This is where the trade off with refresh tokens occurs.



This avoids having to check the user table with every request and the time it takes to validate a user.

The disadvantage is that banning a user or disabling access is not immediate.

Add a script to clear out expired refresh tokens from the whitelist

If the client stops using the API for more than 5 days and the refresh token expires but the client never logs out then the expired refresh token will remain on the white list.

In the whitelist table we insert a dummy refresh token.

token_hash	expires_at
abc234	1000
474742afb38ce5a7052a6d02c4b3fbc6f8a438baec25df713a...	1670660339

We will do this by creating a script that will run as a task scheduler like CHRON because this is a maintenance script.

delete-expired-refresh-tokens.php script is created in the ROOT of out app and requires the autoload and environment variables for the database.

```php
<?php
declare(strict_types=1);
require __DIR__ . "/vendor/autoload.php";

$dotenv = Dotenv\Dotenv::createImmutable(__DIR__);
$dotenv->load();
```

```php
public function deleteExpired(): int {
    $sql = "DELETE FROM refresh_token WHERE expires_at < UNIX_TIMESTAMP()";
    $stmt = $this->conn->query($sql);
    return $stmt->rowCount();
}
```

In the refresh token gateway class we need a method that will delete all rows from the whitelist where ‘expires_at’ column is greater than the current UNIX timestamp.

```php
<?php
declare(strict_types=1);
require __DIR__ . "/vendor/autoload.php";

$dotenv = Dotenv\Dotenv::createImmutable(__DIR__);
$dotenv->load();

$database = new Database($_ENV["DB_HOST"], $_ENV["DB_NAME"], $_ENV["DB_USER"],
$_ENV["DB_PASS"]);

$refresh_token_gateway = new RefreshTokenGateway($database, $_ENV["SECRET_KEY"]);
echo $refresh_token_gateway->deleteExpired(), "\n";
```

The script above will now remove all expired refresh tokens from the whitelist table and can be scheduled to run periodically in CHRON.

Course Content

file_get_contents | json_decode | json_encode | var_dump | curl_init() | curl_setopt() | curl_setopt_array() | curl_exec() | curl_close() | curl_getinfo() | CURLOPT_URL |API endpoints | API HTTP response codes | CURLOPT_RETURNTRANSFER | CURLOPT_HTTPHEADER | custom cURL headers | cURL request payload | RESTfull APIs | Guzzle | SDK | postman | HTTPie | install and use PHP composer |php autoloader | Create a RESTful API program that performs CRUD | dot env dependancies file | API key authentication | API authentication via query string | API authentication via HTTP header | API authentication with access token | build php API access token app | apache_request_headers() |htaccess HTTP_AUTHORIZATION config | hash_hmac() | hash_equals() | base64_encode() | base64_decode() | preg_match() | Auuthentication with JSON web tokens (JWTs) | build a PHP JWT codec | build php JWT API app | JWT payload specifications | Acces token expiry & refresh |