

+++++++>>
2017.02.10.9.38.PM

Continuing with the tutorial and seeing if we are able to get data onto HTML pages from the Drupal Environment

By default, the REST module enables the node entity resource for all GET, POST, PATCH, and DELETE operations. It supports basic or cookie authentication and the HAL or JSON formats. You can see these default settings in *sites/default/files/config_XXXX/active/rest.settings.yml*. To enable REST on other entities (e.g. users, files, or fields), you'll need to edit this file. There is, however, a handy contributed module created by our very own [Juampy](#) named [REST UI](#). This module provides a user interface for enabling and disabling resources, serialization formats, and authentication providers. See the screenshot below for an example of the configuration options REST UI provides.

REST resources

Here you can enable and disable available resources. Once a resource has been enabled, you can restrict its formats and authentication by clicking on its "Edit" link.

Enabled

RESOURCE NAME	PATH	DESCRIPTION	OPERATIONS
Content	/entity/node/{id}	GET authentication: basic_auth formats: hal_json, json POST authentication: basic_auth formats: hal_json DELETE authentication: basic_auth formats: hal_json PATCH authentication: basic_auth formats: hal_json	Edit ▾

the following Doc is probably for a different version of Drupal, we no longer have to handle GET in the documentation as shown

Let me take this opportunity to explain the available resource options, GET, POST, PATCH, and DELETE. These are all common HTTP methods. Each operation signifies a type of action to be performed on a resource. For example, your browser used GET when you requested to view this blog post. To read a resource, we GET. To create a resource, we POST. To update a resource, we PATCH. And to delete a resource, we DELETE. Read [this post](#) if you're wondering why we use PATCH and not PUT.

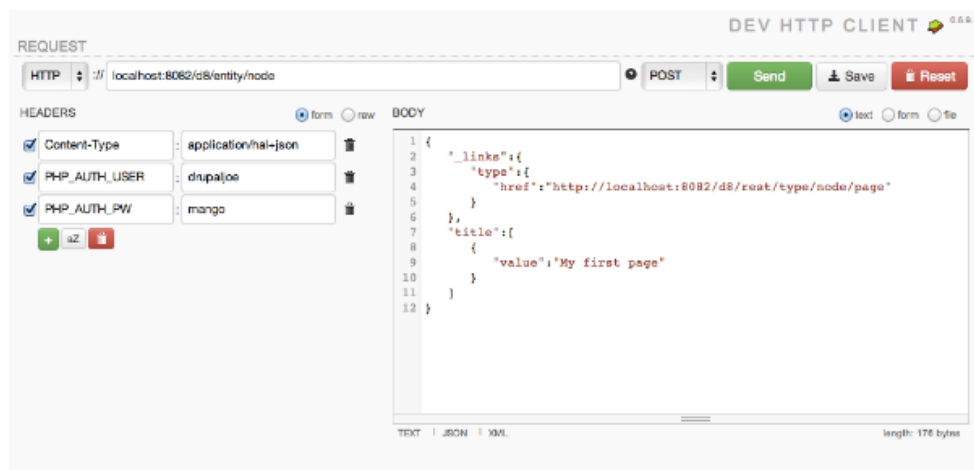
Now back to our setup. For all enabled resources, the REST module can set user permissions. Go to *admin/people/permissions* and set up your permissions, as required. Here is an example of how I set mine:

PERMISSION	ANONYMOUS USER	AUTHENTICATED USER	ADMINISTRATOR
RESTful Web Services			
Access GET on <i>Content</i> resource	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Access POST on <i>Content</i> resource	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Access DELETE on <i>Content</i> resource	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Access PATCH on <i>Content</i> resource	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>

Using the REST API

Now we're ready to start using our REST API. I recommend the [Dev HTTP Client](#) Chrome extension for testing calls to any API. You can also write scripts in [Guzzle](#), which is a great new tool included in Drupal 8 Core or cURL via the command line or PHP.

Start by creating a node entity. To do this, we must send a POST to */entity/node* with the Content-Type header set to *application/hal+json* and declare the required *type* and *title* fields in the request BODY. But don't forget that because we are using Basic Auth, we need to set the headers *PHP_AUTH_USER* and *PHP_AUTH_PW* to authenticate as our user. Here is how it looks in the Dev HTTP Client:



The screenshot shows the Dev HTTP Client interface. The URL is `localhost:8082/drupal/entity/node` and the method is `POST`. The headers are set to `Content-Type: application/hal+json`, `PHP_AUTH_USER: drupaljoe`, and `PHP_AUTH_PW: mango`. The request body is a JSON object:

```
1 {
2   "_links":{
3     "type":{
4       "href":"http://localhost:8082/drupal/type/node/page"
5     }
6   },
7   "title":{
8     "value":"My first page"
9   }
10 }
11 }
12 }
```

The interface also shows a "length: 178 bytes" at the bottom right.

May need to look at this site to grab a PHP and Drupal specific UI development Toolbar-administration <http://docs.guzzlephp.org/en/latest/>

Guzzle 6

Docs / Guzzle, PHP HTTP client

Guzzle Documentation

Guzzle is a PHP HTTP client that makes it easy to send HTTP requests and trivial to integrate with web services.

- Simple interface for building query strings, POST requests, streaming large uploads, streaming large downloads, using HTTP cookies, uploading JSON data, etc...
- Can send both synchronous and asynchronous requests using the same interface.
- Uses PSR-7 interfaces for requests, responses, and streams. This allows you to utilize other PSR-7 compatible libraries with Guzzle.
- Abstracts away the underlying HTTP transport, allowing you to write environment and transport agnostic code; i.e., no hard dependency on cURL, PHP streams, sockets, or non-blocking event loops.
- Middleware system allows you to augment and compose client behavior.

```
$client = new GuzzleHttp\Client();
$res = $client->request('GET', 'https://api.github.com/user', [
    'auth' => ['user', 'pass']
]);
echo $res->getStatusCode();
// "200"
echo $res->getHeader('content-type');
// 'application/json; charset=utf8'
echo $res->getBody();
// {"type": "User"...}

// Send an asynchronous request.
$request = new \GuzzleHttp\Psr7\Request('GET', 'http://httpbin.org');
$promise = $client->sendAsync($request)->then(function ($response) {
    echo 'I completed! ' . $response->getBody();
});
$promise->wait();
```

Additional information from the same site can be found here... <https://drupalize.me/blog/201402/your-first-restful-view-drupal-8>