**What are plugins in drupal8?**

**Drupal 8 introduces plugins. Plugins power many items in Drupal, such as blocks, field types, field formatters, and many more. Plugins and plugin types are provided by modules. They provide a swappable and specific functionality. Breakpoints, as discussed in Chapter 5, Front End for the Win, are plugins. In this chapter, we will discuss how plugins work in Drupal 8 and show you how to create blocks, fields, and custom plugin types.**

**what is plugin manager in Drupal 8?**

The **plugin manager** is the central controlling class that defines how the **plugins** of a particular type will be discovered and instantiated. This class is called directly in any module wishing to invoke a **plugin** type.

PLUGIN MANAGER ARE DEFINED AS A SERVICE.

**Difference between private and protected in php?**

**Private**: Method or property with **private** visibility can only be accessible inside the class. You can not access **private** method or variable from outside of your class.

**Protected**: Method or variable with **protected** visibility can only be access **in the** child class. **Protected** will be used **in the** process of inheritance.

**CREATE A MENU LINK IN DRUPAL 8**

Step 1: Create a new directory in the modules directory called *hello*.

Step 2: In the hello directory, create a file called *hello.info.yml* and add the following code to that file:

name: Hello

description: An experimental module to build our first Drupal 8 module

package: Custom

type: module

version: 1.0

core: 8.x

Step 3: In the hello directory, create a file for the route called *hello.routing.yml* and add the following code to that file:

hello.content:

path: '/hello/{name}'

defaults:

\_controller: 'Drupal\hello\Controller\HelloController::content'

\_title: 'Hello world'

name: 'there'

requirements:

\_permission: 'access content'

Step 4: In the hello directory, create a new directory called src and inside that create a directory called Controller. In the Controller directory, create a file for the controller called *HelloController.php* and add the following code to it. Add open PHP tags to the top of the file (I currently can't add this to the code snippet as it breaks the code highlighter).

<?php

/\*\*

\* @file

\* Contains \Drupal\hello\Controller\HelloController.

\*/

namespace Drupal\hello\Controller;

use Drupal\Core\Controller\ControllerBase;

class HelloController extends ControllerBase {

public function content($name) {

return array(

'#type' => 'markup',

'#markup' => $this->t('Hello @name', array('@name' => $name)),

);

}

}

Step 5: In the hello directory, create a file for the menu links called *hello.links.menu.yml* and add the following code to it:

hello.demo:

title: 'Hello'

description: 'Hello page'

parent: main

menu\_name: main

route\_name: hello.content

Step 6: Optionally change the menu link and nest it under the Home menu link:

hello.demo:

title: 'Hello'

description: 'Hello page'

parent: standard.front\_page

menu\_name: main

route\_name: hello.content

Step 7: Clear the cache

**SERVICES IN DRUPAL 8?**

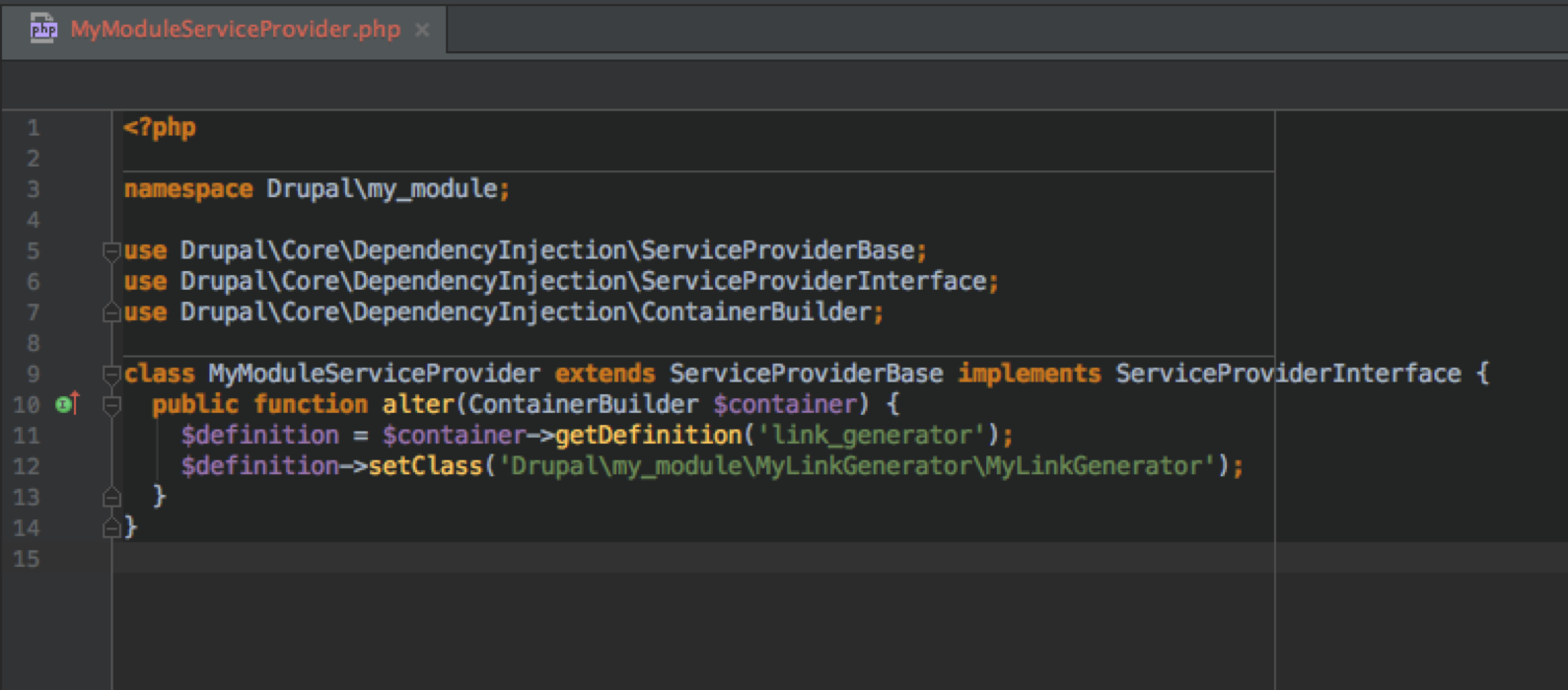
In **Drupal 8** speak, a service is any object managed by the **services** container. **Drupal 8** introduces the concept of **services** to decouple reusable functionality and makes these **services** pluggable and replaceable by registering them with a service container.

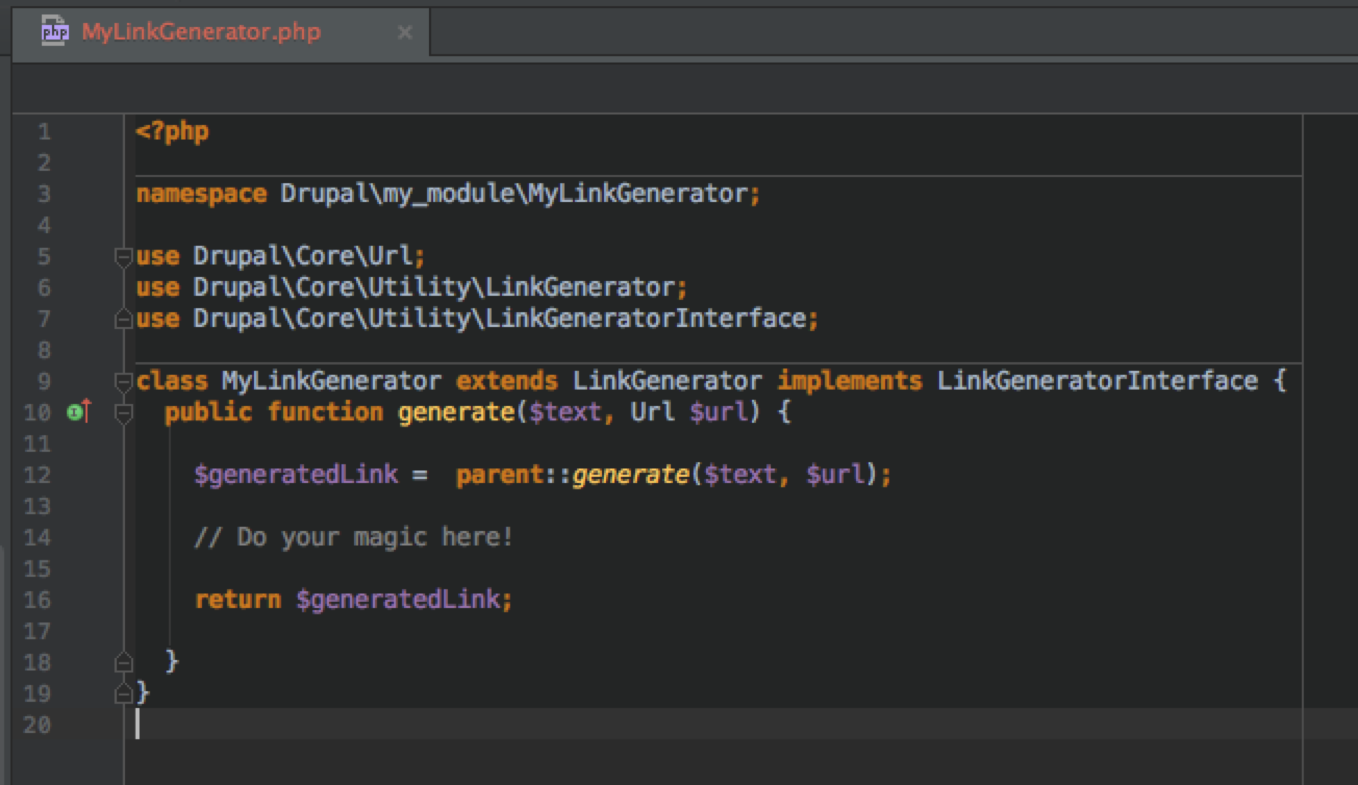
**Name some of drupal core services?**

<https://www.drupal.org/docs/8/api/services-and-dependency-injection/services-and-dependency-injection-in-drupal-8>

**How to override drupal core services?**

<https://www.bounteous.com/insights/2017/04/19/drupal-how-override-core-drupal-8-service/>





**Expose CUSTOM TABLE TO VIEWS?**

<https://www.drupal.org/project/view_custom_table>

View custom table module provide you functionality to integrate your custom table data to views, and access all it's column in views. This module use hook\_view\_data to add custom tables in views. this module provides you following functionalities.

**Cache tags FTW!**

Cache Tags are one of the key improvements of Drupal 8. They allow Drupal to know which entities are used during a page view. This means that Drupal saves the cache tags along with the data it caches. ***So if Drupal needs to clear the caches because one entity is updated (let’s assume node with ID 77), instead of* clearing all caches**, it can easily check ***which caches actually used node 77 and clear only those***. (You can learn more about cache tags on [drupal.org](https://www.drupal.org/docs/8/api/cache-api/cache-tags)).

A cache tag is a string.

Cache tags are passed around in sets (order doesn't matter) of strings, so they are typehinted to string[]. They're sets because a single cache item can depend on (be invalidated by) many cache tags.

By convention, they are of the form thing:identifier — and when there's no concept of multiple instances of a thing, it is of the form thing. The only rule is that it cannot contain spaces.

**what preprocess function can do apart from variables ovverride?**

You can affect the output of certain HTML via preprocess functions. For example, if you wanted to add a class to a menu and preferred to do this at the PHP level you can. This is a good way to alter theme-specific markup, but if you want to make theme-independent markup it is better to code a custom module.

To work with preprocess functions:

1. Create or edit a file in your theme directory called mytheme.theme
2. Create a function such as mytheme\_preprocess\_HOOK where HOOK refers to the item you wish to affect \*
3. Write your changes and save
4. Rebuild the cache so your changes are available (if you have drush installed, drush cr on the command line)

\* HOOK names follow twig template suggestions. To create a hook for page.html.twig you create mytheme\_preprocess\_page. To create a hook for node--article.html.twig you create mytheme\_preprocess\_node\_\_article (replacing dashes with underscores). To discover hook names, see [Locating Template Files with Debugging](https://www.drupal.org/node/2358785).

Let's assume we wanted to add a class of my-menu to all of the menus on your site. Assuming your theme is called "mytheme" you would write the following function:

/\*\*

\* Implements hook\_preprocess\_HOOK() for menu.html.twig.

\*/

function mytheme\_preprocess\_menu(&$variables) {

// If there is not an existing class array, create an empty array.

if (!isset($variables['attributes']['class'])) {

$variables['attributes']['class'] = [];

}

// Merge with any classes that may have been set by other hook\_preprocess\_menu invocations

$variables['attributes']['class'] = array\_merge($variables['attributes']['class'], ['my-menu']);

}

Differences from Drupal 7

* There is no longer a template.php file. That file has been replaced by mytheme.theme. However, it still functions in much the same way allowing for hooks to modify output.

**HOW TO CREATE OWN PROFILE IN DRUPAL 8?**

### **Creating the file structure**

<https://www.drupal.org/docs/8/distributions/creating-distributions/how-to-write-a-drupal-8-installation-profile>

Your installation profile will reside in its own profilename directory in the /profiles directory of a Drupal 8 site.

All installation profiles must have a [profilename.info.yml](https://www.drupal.org/docs/8/distributions/creating-distributions/how-to-write-a-drupal-8-installation-profile" \l "info) file. They may also have:

1. [profilename.install file](https://www.drupal.org/docs/8/distributions/creating-distributions/how-to-write-a-drupal-8-installation-profile#install)
2. [profilename.profile](https://www.drupal.org/docs/8/distributions/creating-distributions/how-to-write-a-drupal-8-installation-profile#profile)
3. [config folder](https://www.drupal.org/docs/8/distributions/creating-distributions/how-to-write-a-drupal-8-installation-profile#config)
4. [translations folder](https://www.drupal.org/docs/8/distributions/creating-distributions/how-to-write-a-drupal-8-installation-profile#translations)
5. *profilename*.info.yml file should look similar to this:
6. name: Profile Name
7. type: profile
8. description: 'Description of your profile.'
9. core: 8.x

**How to switch themes programmatically?**

<https://jimconte.com/blog/web/dynamic-theme-switching-in-drupal-8>

Drupal 8's **ThemeNegotiatorInterface** provides a way for developers to create their own management class that decides which theme should go with which content.

## Implement The Interface

To start, [take a look at the documentation](https://api.drupal.org/api/drupal/core!lib!Drupal!Core!Theme!ThemeNegotiatorInterface.php/interface/ThemeNegotiatorInterface), and create a class that implements the interface.

**<?php**

/\*\*

\* @file

\* Contains \Drupal\jcmodule\Theme\ThemeNegotiator

\*/

namespace Drupal\jcmodule\Theme;

use Drupal\Core\Routing\RouteMatchInterface;

use Drupal\Core\Theme\ThemeNegotiatorInterface;

class ThemeNegotiator implements ThemeNegotiatorInterface {

View source

ThemeNegotiatorInterface requires two methods to be implemented

// Whether this theme negotiator should be used to set the theme.

function applies(RouteMatchInterface $route\_match)

// Determines the active theme for the request.

function determineActiveTheme(RouteMatchInterface $route\_match)

View source

If applies return true, determineActiveTheme is called and either the theme is chosen or null is returned

To keep my class easy to maintain, I created my own common negotiateRoute function that satisfies the requirements for both interface functions.. If my function returns a theme name, that theme is used.

Here is my common class function

/\*\*

\* Function that does all of the work

\* @param RouteMatchInterface $route\_match

\* @return null|string

\*/

private function negotiateRoute(RouteMatchInterface $route\_match)

{

if ($route\_match->getRouteName() == 'user.login')

{

return 'seven';

}

elseif ($route\_match->getRouteName() == 'some.other.route')

{

return 'some\_other\_theme';

}

return null;

}

View source

## Symfony Service Requirement

For Drupal to find your implementation, you need to set up your class as a [Symfony service](http://symfony.com/doc/2.8/service_container.html" \t "_blank) and tag it.

# Module services file jcmodule.services.yml

services:

jcmodule.theme.negotiator:

class: Drupal\jcmodule\Theme\ThemeNegotiator

tags:

- { name: theme\_negotiator, priority: 1000 }

View source

* The tag name theme\_negotiator tells Drupal that this is a class that implements ThemeNegotiatorInterface, and should be used to determine the theme in the current request.
* In my example, routes are checked to determine the theme, but you can add whatever logic you may need. This is where you can get creative. Switching the theme based on a user's role can be done within this context:
* $userRolesArray = \Drupal::currentUser()->getRoles();
* if (in\_array("administrator", $userRolesArray))
* {
* return 'seven';
* }

**Load template of a page based on field value of the node?**

We add theme settings in theme-settings.php. If you don’t already have it, add it in the root of your theme, same place as mytheme.info.yml

We define variables in mytheme.theme

Conditional statements in Twig:

{% if the\_thing %}  
 // do this  
{% endif %}

{% if mytheme\_contact\_title %}  
<h2>{{ mytheme\_contact\_title }}</h2>  
{% endif %}

{% if mytheme\_contact\_title or mytheme\_contact\_address or mytheme\_contact\_email or mytheme\_contact\_phone or mytheme\_contact\_phone %} // that whole contact block{% endif %}

Provide a custom template/hook for your custom module.

**Drupal 8 method**

In the lotus.module file:

/\*\*

\* Implements hook\_theme() to add the template definition.

\*\*/

function lotus\_theme($existing, $type, $theme, $path) {

return array(

'lotus\_template' => array(

'variables' => array('test\_var' => NULL),

),

);

}

In the LotusController.php file:

//Calling from the Controller

/\*\*

\* @file

\* Contains \Drupal\lotus\Controller\LotusController.php

\*/

namespace Drupal\lotus\Controller;

use Drupal\Core\Controller\ControllerBase;

class LotusController extends ControllerBase {

public function content() {

return array(

'#theme' => 'lotus\_template',

'#test\_var' => $this->t('Test Value'),

);

}

}

In the template folder create the lotus-template.html.twig file:

<p> This is the lotus template with a value of {{ test\_var }} </p>

**Passing configuration values from PHP to a JavaScript library.**

## Drupal 8 method

In the lotus.libraries.yml file, you must declare a dependency to drupalSettings:

lotus-js:

version: 1.x

js:

js/lotus.js: {}

dependencies:

- core/jquery

- core/drupalSettings

In the lotus.module file:

function lotus\_preprocess\_html(&$variables) {

$lotus\_height = '300px';

//Add a JS library

$variables['#attached']['library'][] = 'lotus/lotus-js';

$variables['#attached']['drupalSettings']['lotus']['lotusJS']['lotus\_height'] = $lotus\_height;

}

So in your module\_preprocess\_html you should write

$variables['#attached']['drupalSettings'][‘module\_name’]['variable\_you\_want\_to\_pass']['lotus\_height'] = $lotus\_height;

}

In the lotus.js file:

(function ($, Drupal, drupalSettings) {

Drupal.behaviors.LotusBehavior = {

attach: function (context, settings) {

// can access setting from 'drupalSettings';

var lotusHeight = drupalSettings.lotus.lotusJS.lotus\_height;

$('lotusElement').css('height', lotusHeight);

}

};

})(jQuery, Drupal, drupalSettings);

**Cache for authenticated users in drupal 8?**

<https://www.drupal.org/docs/8/core/modules/dynamic-page-cache/overview>

# **Dynamic Page Cache overview**

Last [updated](https://www.drupal.org/node/2541358/discuss) on

14 June 2018

Drupal 8 provides the Dynamic Page Cache module that is recommended for websites of all sizes. It caches pages minus the personalized parts, and is therefore useful for all users (both anonymous & authenticated).

We can find this core module at: **core/modules/dynamic\_page\_cache**.

This feature improves performance because it makes it possible to cache pages with dynamic content. Pages requested by users (anonymous or authenticated) are stored the first time they are requested and can then reused. Personalized parts are excluded: they are [turned into placeholders automatically](https://www.drupal.org/developing/api/8/render/arrays/cacheability/auto-placeholdering). Depending on your site configuration and the complexity of particular pages, Dynamic Page Cache may significantly increase the speed of your site, even for authenticated users.

**how to share a table between 2 sites in multi site enviroment**

**how to create your own entity?**

We do two things, we **extend** an existing **[ContentEntityBase](https://api.drupal.org/api/drupal/core!lib!Drupal!Core!Entity!ContentEntityBase.php/class/ContentEntityBase/8)** class that already has the necessary methods to interact with the DB, and **implement** an **[ContentEntityInterface](https://api.drupal.org/api/drupal/core!lib!Drupal!Core!Entity!ContentEntityInterface.php/interface/ContentEntityInterface/8)** to describe...

the methods that we need to access our database. It does NOT describe in any way HOW we achieve that. That's what the IMPLEMENTing class does. We can IMPLEMENT this interface as many times as we need in as many different ways as we need. We can then switch between implementations of the interface without impact to our code because the interface defines how we will use it regardless of how it actually works. - <https://secure.php.net/manual/en/language.oop5.interfaces.php>

class Advertiser extends ContentEntityBase implements ContentEntityInterface {

**how to create rest api in drupal 8?**

**src\Plugin\rest\resource\ExampleGetRestResource.php**

|  |
| --- |
|  |
| class ExampleGetRestResource extends ResourceBase { |
|  |

**POST RESt api**

|  |
| --- |
| <?php |
|  |  |
|  | /\*\* |
|  | \* @file |
|  | \* Contains Drupal\custom\_rest\Plugin\rest\resource\custom\_rest. |
|  | \*/ |
|  |  |
|  | namespace Drupal\custom\_rest\Plugin\rest\resource; |
|  |  |
|  | use Drupal\Core\Session\AccountProxyInterface; |
|  | use Drupal\rest\Plugin\ResourceBase; |
|  | use Drupal\rest\ResourceResponse; |
|  | use Symfony\Component\DependencyInjection\ContainerInterface; |
|  | use Symfony\Component\HttpKernel\Exception\AccessDeniedHttpException; |
|  | use Symfony\Component\HttpKernel\Exception\HttpException; |
|  | use Psr\Log\LoggerInterface; |
|  |  |
|  | /\*\* |
|  | \* Provides a resource to get view modes by entity and bundle. |
|  | \* |
|  | \* @RestResource( |
|  | \* id = "custom\_rest\_resource", |
|  | \* label = @Translation("Custom rest resource"), |
|  | \* uri\_paths = { |
|  | \* "canonical" = "//api/custom" |
|  | \* } |
|  | \* ) |
|  | \*/ |
|  | class CustomRestResource extends ResourceBase { |
|  | /\*\* |
|  | \* A current user instance. |
|  | \* |
|  | \* @var \Drupal\Core\Session\AccountProxyInterface |
|  | \*/ |
|  | protected $currentUser; |
|  |  |
|  | /\*\* |
|  | \* Constructs a Drupal\rest\Plugin\ResourceBase object. |
|  | \* |
|  | \* @param array $configuration |
|  | \* A configuration array containing information about the plugin instance. |
|  | \* @param string $plugin\_id |
|  | \* The plugin\_id for the plugin instance. |
|  | \* @param mixed $plugin\_definition |
|  | \* The plugin implementation definition. |
|  | \* @param array $serializer\_formats |
|  | \* The available serialization formats. |
|  | \* @param \Psr\Log\LoggerInterface $logger |
|  | \* A logger instance. |
|  | \* @param \Drupal\Core\Session\AccountProxyInterface $current\_user |
|  | \* A current user instance. |
|  | \*/ |
|  | public function \_\_construct( |
|  | array $configuration, |
|  | $plugin\_id, |
|  | $plugin\_definition, |
|  | array $serializer\_formats, |
|  | LoggerInterface $logger, |
|  | AccountProxyInterface $current\_user) { |
|  | parent::\_\_construct($configuration, $plugin\_id, $plugin\_definition, $serializer\_formats, $logger); |
|  |  |
|  | $this->currentUser = $current\_user; |
|  | } |
|  |  |
|  | /\*\* |
|  | \* {@inheritdoc} |
|  | \*/ |
|  | public static function create(ContainerInterface $container, array $configuration, $plugin\_id, $plugin\_definition) { |
|  | return new static( |
|  | $configuration, |
|  | $plugin\_id, |
|  | $plugin\_definition, |
|  | $container->getParameter('serializer.formats'), |
|  | $container->get('logger.factory')->get('rest'), |
|  | $container->get('current\_user') |
|  | ); |
|  | } |
|  | /\*\* |
|  | \* Responds to POST requests. |
|  | \* |
|  | \* Returns a list of bundles for specified entity. |
|  | \* |
|  | \* @throws \Symfony\Component\HttpKernel\Exception\HttpException |
|  | \* Throws exception expected. |
|  | \*/ |
|  | public function post() { |
|  |  |
|  | // You must to implement the logic of your REST Resource here. |
|  | // Use current user after pass authentication to validate access. |
|  |  |
|  | /\* |
|  | if(!$this->currentUser->hasPermission($permission)) { |
|  | throw new AccessDeniedHttpException(); |
|  | } |
|  | \*/ |
|  |  |
|  | // Throw an exception if it is required. |
|  | // throw new HttpException(t('Throw an exception if it is required.')); |
|  | return new ResourceResponse("Implement REST State POST!"); |
|  | } |
|  |  |
|  | } |

**Get rest api**

|  |
| --- |
| <?php |
|  |  |
|  | namespace Drupal\example\_rest\Plugin\rest\resource; |
|  |  |
|  | use Drupal\Core\Session\AccountProxyInterface; |
|  | use Drupal\rest\Plugin\ResourceBase; |
|  | use Drupal\rest\ResourceResponse; |
|  | use Symfony\Component\DependencyInjection\ContainerInterface; |
|  | use Symfony\Component\HttpKernel\Exception\AccessDeniedHttpException; |
|  | use Psr\Log\LoggerInterface; |
|  |  |
|  | /\*\* |
|  | \* Provides a resource to get view modes by entity and bundle. |
|  | \* |
|  | \* @RestResource( |
|  | \* id = "example\_get\_rest\_resource", |
|  | \* label = @Translation("Example get rest resource"), |
|  | \* uri\_paths = { |
|  | \* "canonical" = "/example-rest" |
|  | \* } |
|  | \* ) |
|  | \*/ |
|  | class ExampleGetRestResource extends ResourceBase { |
|  |  |
|  | /\*\* |
|  | \* A current user instance. |
|  | \* |
|  | \* @var \Drupal\Core\Session\AccountProxyInterface |
|  | \*/ |
|  | protected $currentUser; |
|  |  |
|  | /\*\* |
|  | \* Constructs a Drupal\rest\Plugin\ResourceBase object. |
|  | \* |
|  | \* @param array $configuration |
|  | \* A configuration array containing information about the plugin instance. |
|  | \* @param string $plugin\_id |
|  | \* The plugin\_id for the plugin instance. |
|  | \* @param mixed $plugin\_definition |
|  | \* The plugin implementation definition. |
|  | \* @param array $serializer\_formats |
|  | \* The available serialization formats. |
|  | \* @param \Psr\Log\LoggerInterface $logger |
|  | \* A logger instance. |
|  | \* @param \Drupal\Core\Session\AccountProxyInterface $current\_user |
|  | \* A current user instance. |
|  | \*/ |
|  | public function \_\_construct( |
|  | array $configuration, |
|  | $plugin\_id, |
|  | $plugin\_definition, |
|  | array $serializer\_formats, |
|  | LoggerInterface $logger, |
|  | AccountProxyInterface $current\_user) { |
|  | parent::\_\_construct($configuration, $plugin\_id, $plugin\_definition, $serializer\_formats, $logger); |
|  |  |
|  | $this->currentUser = $current\_user; |
|  | } |
|  |  |
|  | /\*\* |
|  | \* {@inheritdoc} |
|  | \*/ |
|  | public static function create(ContainerInterface $container, array $configuration, $plugin\_id, $plugin\_definition) { |
|  | return new static( |
|  | $configuration, |
|  | $plugin\_id, |
|  | $plugin\_definition, |
|  | $container->getParameter('serializer.formats'), |
|  | $container->get('logger.factory')->get('example\_rest'), |
|  | $container->get('current\_user') |
|  | ); |
|  | } |
|  |  |
|  | /\*\* |
|  | \* Responds to GET requests. |
|  | \* |
|  | \* Returns a list of bundles for specified entity. |
|  | \* |
|  | \* @throws \Symfony\Component\HttpKernel\Exception\HttpException |
|  | \* Throws exception expected. |
|  | \*/ |
|  | public function get() { |
|  |  |
|  | // You must to implement the logic of your REST Resource here. |
|  | // Use current user after pass authentication to validate access. |
|  | if (!$this->currentUser->hasPermission('access content')) { |
|  | throw new AccessDeniedHttpException(); |
|  | } |
|  | $entities = \Drupal::entityTypeManager() |
|  | ->getStorage('node') |
|  | ->loadMultiple(); |
|  | foreach ($entities as $entity) { |
|  | $result[$entity->id()] = $entity->title->value; |
|  | } |
|  |  |
|  | $response = new ResourceResponse($result); |
|  | $response->addCacheableDependency($result); |
|  | return $response; |
|  | } |
|  |  |
|  | } |

**what is drupal queue / drupal batch**

<http://karimboudjema.com/en/drupal/20180807/create-queue-controller-drupal8>

Queues are particularly important when we need to stash some tasks for later processing. To do so, we are going to put some tasks or data in a queue (create the queue) and later we will process those tasks with a QueueWorker plugin (process the queue), usually triggered by cron.

There are several ways to create a queue:  
- With a form class  
- With a controller class  
- With a hook\_cron() function  
   
To process the queue, we also have different options:  
- As a cron process with a QueueWorker plugin  
- As a batch process also with QueueWorker plugin but extending a base plugin  
- As a batch process claiming each item of the queue in a service or in a controller

Acquia DAM uses drupal queue.

Drupal queue maintain a table.

**What is final keyword?**

PHP 5 introduces the final keyword, which prevents child classes from overriding a method by prefixing the definition with *final*. If the class itself is being defined final then it cannot be extended.

Example #1 Final methods example

<?php  
class BaseClass {  
   public function test() {  
       echo "BaseClass::test() called\n";  
   }  
     
   final public function moreTesting() {  
       echo "BaseClass::moreTesting() called\n";  
   }  
}  
  
class ChildClass extends BaseClass {  
   public function moreTesting() {  
       echo "ChildClass::moreTesting() called\n";  
   }  
}  
// Results in Fatal error: Cannot override final method BaseClass::moreTesting()  
?>

Example #2 Final class example

<?php  
final class BaseClass {  
   public function test() {  
       echo "BaseClass::test() called\n";  
   }  
  
   // Here it doesn't matter if you specify the function as final or not  
   final public function moreTesting() {  
       echo "BaseClass::moreTesting() called\n";  
   }  
}  
  
class ChildClass extends BaseClass {  
}  
// Results in Fatal error: Class ChildClass may not inherit from final class (BaseClass)  
?>

**what is private and protected**

**what is abstract and interface?**

An **interface** is a contract. An **interface** defines what a class can do, without saying anything about how the class will do it. All the methods in an **interface** is public and **abstract**. An **abstract** class is used to define blueprint for a child classes. The **abstract** class is only created for an inheritance.

**how to do route alter in drupal 8?**

<https://www.drupal.org/docs/8/api/routing-system/altering-existing-routes-and-adding-new-routes-based-on-dynamic-ones>

You can alter existing routes by implementing the [alterRoutes(RouteCollection $collection)](https://api.drupal.org/api/drupal/core%21lib%21Drupal%21Core%21Routing%21RouteSubscriberBase.php/function/RouteSubscriberBase%3A%3AalterRoutes/8) method of this class.

**class RouteSubscriber extends RouteSubscriberBase {**

class RouteSubscriber extends RouteSubscriberBase {

/\*\*

\* {@inheritdoc}

\*/

protected function alterRoutes(RouteCollection $collection) {

// Define custom access for '/user/login'.

if ($route = $collection->get('user.login')) {

$route->setRequirement('\_custom\_access', 'Drupal\example\Access\StandardAccessCheck::access');

}

// Define custom access for '/user/logout'.

if ($route = $collection->get('user.logout')) {

$route->setRequirement('\_custom\_access', 'example.services\_access\_checker::access');

}

}

**how to pass variables from drupal to JS?**

<https://docs.acquia.com/tutorials/fast-track-drupal-8-coding/add-custom-variable-drupalsettings/>

## Drupal 8 method

In the lotus.libraries.yml file, you must declare a dependency to drupalSettings:

lotus-js:

version: 1.x

js:

js/lotus.js: {}

dependencies:

- core/jquery

- core/drupalSettings

In the lotus.module file:

function lotus\_preprocess\_html(&$variables) {

$lotus\_height = '300px';

//Add a JS library

$variables['#attached']['library'][] = 'lotus/lotus-js';

$variables['#attached']['drupalSettings']['lotus']['lotusJS']['lotus\_height'] = $lotus\_height;

}

In the lotus.js file:

(function ($, Drupal, drupalSettings) {

Drupal.behaviors.LotusBehavior = {

attach: function (context, settings) {

// can access setting from 'drupalSettings';

var lotusHeight = drupalSettings.lotus.lotusJS.lotus\_height;

$('lotusElement').css('height', lotusHeight);

}

};

})(jQuery, Drupal, drupalSettings);

**based on the value of the field in node we need to call page template?**

**how to write service in drupal 8?**

In **Drupal 8** speak, a **service** is any object managed by the **services** container. **Drupal 8** introduces the concept of **services** to decouple reusable functionality and makes these **services** pluggable and replaceable by registering them with a **service** container.

 As of **Drupal** 8, **dependency injection** is the preferred method for accessing and using services and should be used whenever possible.

**git fetch and pull difference?**

**how you can bring changes from 1 branch to another?**

**what are the ports of http and https?**

**do you know what is PO files?  
  
what is t function?  
  
how to add div inside a div?  
  
how do you add field in existing table by using hooks?  
  
what is scope in drupal add js?  
  
how to export configuraton in git?  
  
which file is called first when calling a page in drupal?  
  
what is hook update N?**  
You implement hook\_update\_N(), and inside that function, you can add db\_add\_field(). You should also add the field definition to your implementation of hook\_schema().  
  
  
  
difference between drupal 7 and drupal 8?  
  
what is migrate path UI drupal 8?  
  
what is the order of template content type, node type and field type?  
  
how you will transfer a feature which is not transferred by migrate UI path?  
  
how to create a form in drupal 8?

what are mandatory fiels in info.yml file?  
name and type

what s difference between git fetch and git pull?  
  
module\_invoke\_all Invokes a hook in all enabled modules that implement it.  
  
module\_invoke Invokes a hook in a particular module.  
  
Invokes a hook in a particular module.  
  
Invokes a hook in a particular module.  
  
WHAT IS THE T() FUNCTION?