TWIG IN DRUPAL

Reference link:- https://www.youtube.com/watch?v=S0oJGy4a65Q

Twig is a template engine for PHP and it is part of the [Symfony2 framework](http://symfony.com/).

In Drupal 8 [Twig replaces PHP Template as the default templating engine](https://www.drupal.org/node/1831138). One of the results of this change is that all of the theme\_\* functions and PHP Template based \*.tpl.php files have been replaced by \*.html.twig  templating files.

## **Overriding templates**

You can override Drupal core templates by adding templates to your theme folder that [follow a specific naming convention](https://www.drupal.org/node/2354645).

To override templates you need to:

1. Locate the template you wish to override.
2. Copy the template file from its base location into your theme folder.
3. (optionally) Rename the template according to the naming conventions in order to target a more specific subset of areas where the template is used.
4. Modify the template to your liking.

Once you copy a template file into your theme and clear the cache, Drupal will start using your instance of the template file instead of the base version.

# **Comparison of PHPTemplate and Twig theming paradigms**

Twig is a PHP-based compiled templating language. When your web page renders, the Twig engine takes the template and converts it into a 'compiled' PHP template which is stored in a protected directory in sites/default/files/php/twig. The compilation is done once, template files are cached for reuse and are recompiled on clearing the Twig cache.

# **Create custom twig templates for custom module**

Reference link:- <https://www.youtube.com/watch?v=SbsZwkPYEdM>

## **Step 1: Define hook\_theme in .module file**

Create a [module].module file if it doesn't already exist in your module, and add code that defines each of your twig templates. The key of each item in the array is what you will need to call the template later. Do not use dashes in the file name.

\*\*

\* Implements hook\_theme().

\*/

function [module]\_theme($existing, $type, $theme, $path) {

return [

'my\_template' => [

'variables' => ['test\_var' => NULL],

],

];

}

## **Step 2: Create Twig Template**

In your module, inside of the templates folder, create your twig template. The name of the file has to match what you put into hook\_theme() (**make sure replace underscores with dashes**). In this case, the file name would be my-template.html.twig.

{# [module]/templates/my-template.html.twig #}

<p>Test twig template!</p>

<p>test\_var: {{ test\_var }}</p>

## **Step 3: Call the Template**

### **Step 3.1: Call from controller**

In the place where you are returning your render array (whether from a controller method that is called from your router yml file, or wherever), make a call to your twig template. Below is an example from a testing module that is called from the routing yml file in the module

<?php

/\*\*

\* @file

\* Contains \Drupal\test\_twig\Controller\TestTwigController.

\*/

namespace Drupal\test\_twig\Controller;

use Drupal\Core\Controller\ControllerBase;

class TestTwigController extends ControllerBase {

public function content() {

return [

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

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

];

}

}

### **Step 3.2: Render as HTML**

You can also use render service method to build the output if you need to use this as part of a different workflow in your code: -

$renderable = [

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

'#test\_var' => 'test variable',

];

$rendered = \Drupal::service('renderer')->renderPlain($renderable);

### **Step 3.3: Render as part of another plugin (such as block).**

You can also use render array as output of custom plugin such as block:

<?php

namespace Drupal\[module]\Plugin\Block;

use Drupal\Core\Block\BlockBase;

/\*\*

\* Provides a 'My Template' block.

\*

\* @Block(

\* id = "my\_template\_block",

\* admin\_label = @Translation("My Template")

\* )

\*/

class MyTemplateBlock extends BlockBase {

/\*\*

\* {@inheritdoc}

\*/

public function defaultConfiguration() {

return ['label\_display' => FALSE];

}

/\*\*

\* {@inheritdoc}

\*/

public function build() {

$renderable = [

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

'#test\_var' => 'test variable',

];

return $renderable;

}

}

## 

However in our case, we have used method 3.1 i.e call from controller