



Drupal 8  
Développeurs

# Introduction à Drupal 8



# Introduction à Drupal 8

## Refonte Orienté Objet

Ce qu'il faut savoir :

- POO
- Composer
- Symfony
- Routage et contrôleurs
- Plugins
- Injection de dépendances et le conteneur de services



# Introduction à Drupal 8

- Drupal 8 est une solution "jeune"
- Tout a changé depuis Drupal 7
- Peu de retours d'expérience
- En phase de stabilisation sur les implémentations ou process



# Introduction à Drupal 8

## Les normes PHP

Viennent du PHP Framework Interoperability Group (FIG)

Différentes normes

- PSR-0 : autoloader standard
- PSR-1 : normes de codage de base (Drupal les suit presque)
- PSR-2 : normes de codage plus poussées (Drupal les suit presque)
- PSR-3 : interface du logger (pas implémentée dans Drupal)
- PSR-4 : autoloader amélioré (choisi par Drupal) : <https://www.drupal.org/node/2156625>



# Introduction à Drupal 8

## Composants Symfony

Le plus populaire des frameworks PHP aujourd'hui  
(<http://symfony.com/doc/current/index.html>)

Ensemble de composants réutilisables  
Certains sont réutilisés par Drupal

Symfony2 Framework	
Vital components	Helper components
<b>Libraries &amp; Assets</b> <i>3rd party libraries for Symfony.</i>	<b>Browser Kit</b>
<b>WebProfilerBundle</b>	<b>Config</b>
<b>FrameworkBundle</b>	<b>Form</b>
<b>SwiftMailerBundle</b>	<b>.. other helper components ..</b>
<b>Services configuration</b>	<b>Debug</b> <i>Nice debug messages.</i>
<b>Kernel</b> <i>Provides Symfony's bundle functionality.</i>	<b>Process</b> <i>API for shell executions (like shell_exec).</i>
<b>HttpKernel</b> <i>Building blocks for creating any web application.</i>	<b>Serializer</b> <i>XML/JSON serialization. Replaces drupal_json_decode.</i>
<b>Routing</b> <i>Maps URLs to code (previously done in hook_menu)</i>	<b>Validator</b> <i>Data validation framework and classes.</i>
<b>EventDispatcher</b> <i>Provides the functionality for event dispatching; this is used a lot by other components.</i>	<b>Translation</b> <i>Contains translation services and translator interface, which is used by the validation component.</i>
<b>DependencyInjection</b> <i>Manages the configuration of services and their dependencies.</i>	<b>Yaml</b> <i>YAML config file parsing.</i>
<b>Http foundation</b> <i>HTTP Request &amp; Response</i>	
<b>Legend</b>	
Not in Drupal 8	
In Drupal 8 (partially)	
In Drupal 8	

# Installation de l'environnement



# Les standards de codage

- Indentation, espaces : lisibilité du code
- Nommage, toujours commencer par le nom système : éviter les conflits
  - fonctions
  - constantes
  - variables persistantes
  - classes
  - fichiers
- Tags <?php non fermés : éviter l'envoi du buffer

[À connaître](#)





# Les modules Drupal utiles au développement

- *Devel* : debug et informations sur les données
- *Drupal Console* : générateur de code
- *Drush* : administration (DRUpal SHell) & téléchargement de modules et librairies
- *Coder* : revue de code
- *Masquerade* : changer d'utilisateur sans se déconnecter
- *Examples for developers* : démonstrations de l'utilisation de l'API
- Ne pas utiliser les caches durant le développement  
(<https://www.drupal.org/node/2598914> / <http://www.tothenew.com/blog/is-your-drupal-8-ready-for-writing-codes/>)

# Composer



- Gestionnaire de dépendances utilisé par la communauté PHP
- Installation uniquement locale au projet
- composer.json
- "composer install"
- composer.lock
- Contient un autoloader
- Pour les performances : <https://github.com/hirak/prestissimo>
- `curl -sS https://getcomposer.org/installer | php mv composer.phar /usr/local/bin/composer`
- [D8 Composer definitive introduction](#)
- [Composer install vs. composer update](#)



# Le serveur Web

xAMP (Apache, MySQL, PHP) conseillé

D'autres possibilités : Nginx / IIS, PostgreSQL

Liste des langages utilisés :

- SQL
- PHP
- Javascript
- HTML
- CSS



# L'éditeur de code

Le meilleur est celui que vous maîtrisez

Différence IDE/Editeur simple :

- Autocomplétion
- XDebug
- Refactoring
- Erreurs de syntaxe

Possibilité de télécharger des configurations

Exemples : PhpStorm / Eclipse / Netbeans / Atom / Vim



# TP : Drush

- Lancer Drush
- Regarder la liste des commandes
- Installer un module (features)
- Regarder à nouveau la liste des commandes
- Installer les modules utiles au développement : devel, masquerade, examples



# TP : Console

- Installer la console Drupal([Documentation](#))
- Vérifier que c'est correctement installé
- Regarder la liste des commandes (drupal list)

```
devel
  devel:dumper (dd)                commands.devel.dumper.messages.change-devel-dumper-pi
develop
  develop:contribute               Download Drupal + Drupal Console to contribute.
dotenv
  dotenv:init                     Dotenv initializer.
entity
  entity:delete (ed)              Delete an specific entity
field
  field:info (fii)               View information about fields.
generate
  generate:ajax:command (gac)     Generate & Register a custom ajax command
  generate:authentication:provider (gap) Generate an Authentication Provider
  generate:breakpoint (gb)        Generate breakpoint
  generate:cache:context (gcc)     Generate a cache context
  generate:command (gco)          Generate commands for the console.
  generate:controller (gcon)      Generate & Register a controller
  generate:entity:bundle (geb)     Generate a new content type (node / entity bundle)
  generate:entity:config (gec)     Generate a new config entity
  generate:entity:content (geco)   Generate a new content entity
```



# Exemples de commandes souvent utilisées

- drupal site:mode dev
- drupal generate:module
- drupal generate:plugin:block
- drupal generate:routessubscriber
- drupal generate:form:config



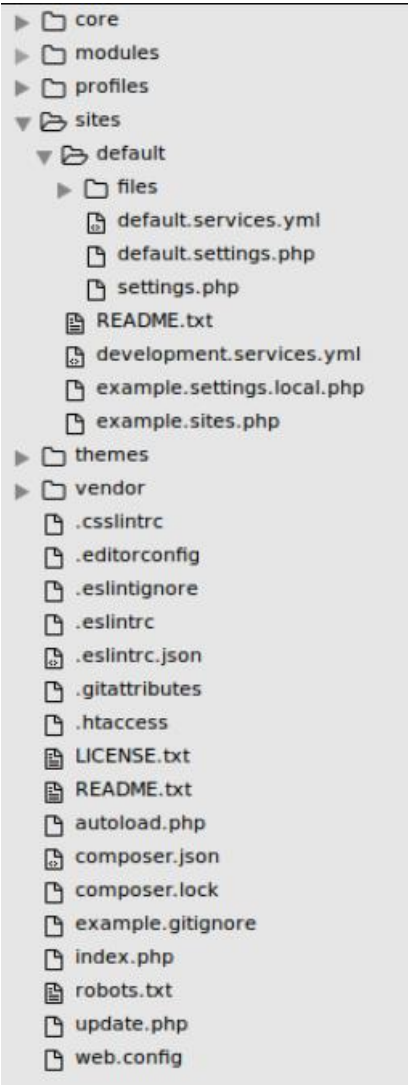
# Le meilleur moyen d'installer Drupal 8

- `composer create-project drupal-composer/drupal-project:8.x-dev some-dir --stability dev --no-interaction`
- `composer create-project drupal/drupal my_site_name` installe un nouveau site
- `composer require drupal/core ~8.5 --update-with-dependencies`` met à jour le cœur





# Organisations des répertoires



Cœur de Drupal  
Modules de tous les sites  
Profils d'installations  
Répertoire spécifique au site  
Répertoire d'upload par défaut  
Fichiers de configuration

Thèmes de tous les sites

Multisites

# Créer son premier module Drupal 8

- Fichiers .yaml
- Format de représentation de données par sérialisation
- Facilement modifiable et lisible
- Pas de tabulation (2 espaces)

```
key: 'value'  
tableau:  
  - valeur 1  
  - valeur 2
```



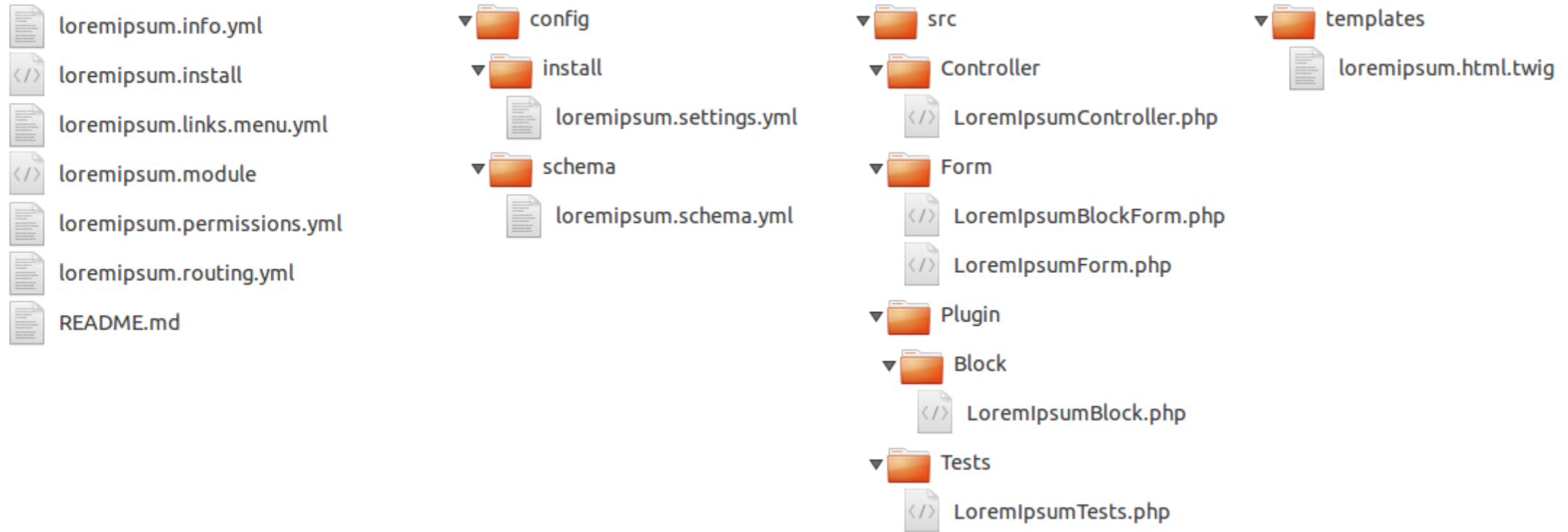
# Un module Drupal 8

- .info.yml (<https://www.drupal.org/node/2000204>)
  - name
  - core
  - type: module /!\
- .module (souvent vide en D8)
- .install facultatif (configuration désormais indépendante)
- répertoire "config/install" pour la configuration
- répertoire "src" pour les Plugins, Controller, Form, ...

```
name: Hello World
description: A Hello World example module
package: Hello World
type: module
core: 8.x
dependencies:
  - node
```



# Structure d'un module



<https://www.drupal.org/node/2560405>



# Les hooks

- Concept historique Drupal
- Implémenter `hook_form_alter()` donnera `mon_module_form_alter()`
- Poids des modules et altération
- Répondent à des déclencheurs
- Des hooks peuvent être déclarés par des modules contrib
- Rappel: on ne « hack » JAMAIS le core (sauf en cas de module bogué)
- Tend à disparaître avec Drupal 8 (Plugins, yml, events), mais existe encore...
- Les implémentations sont mise en cache
- Liste des hooks
  - <https://api.drupal.org/api/drupal/core%21core.api.php/group/hooks/8.5.x>



# Les hooks

## Exemple d'implémentation :

```
/**
 * Implements hook_form_alter().
 */
function mymodule_form_alter(&$form, \Drupal\Core\Form\FormStateInterface $form_state, $form_id) {
  // Custom code and comments go here ...
}
```

```
/**
 * Implements hook_form_alter().
 */
function hooks_example_form_alter(&$form, FormStateInterface $form_state, $form_id) {
  // This is an example of what is known as an alter hook. The $form parameter
  // in this case represents an already complete Form API array and our hook
  // implementation is being given the opportunity to make changes to the
  // existing data structure before it's used. Invoking and alter hooks is a
  // common pattern anytime lists or complex data structures are assembled.
  // hook_form_alter(), which allows you to manipulate any form, is one of the
  // most commonly implemented hooks.
  //
  // @see hook_form_alter()
  // @see hook_form_FORM_ID_alter()
  //
  // If this is the user login form, change the description text of the username
  // field.
  if ($form_id === 'user_login_form') {
    $form['name']['#description'] = t('This text has been altered by hooks_example_form_alter().');
  }
}
```

## Page de permissions (/admin/people/permissions)

Permissions ☆

List

Permissions

Roles

[Home » Administration » People](#)

Permissions let you control what users can do and see on your site. You can define a specific set of permissions for each role. (See the [Roles](#) page to create a role.) Any permissions granted to the Authenticated user role will be given to any user who is logged in to your site. From the [Account settings](#) page, you can make any role into an Administrator role for the site, meaning that role will be granted all new permissions automatically. You should be careful to ensure that only trusted users are given this access and level of control of your site.

[Hide descriptions](#)

PERMISSION	ANONYMOUS USER	AUTHENTICATED USER	ADMINISTRATOR
Block			
Administer blocks	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Comment			
Administer comment types and settings <i>Warning: Give to trusted roles only; this permission has security implications.</i>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Administer comments and comment settings	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Edit own comments	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Post comments	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Skip comment approval	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
View comments	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
Configuration Manager			
Export configuration <i>Warning: Give to trusted roles only; this permission has security implications.</i>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Import configuration	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>







# Les permissions

Fichier .permissions.yml

Statiques :

```
'delete contact entity':  
  title: Delete entity content  
  description: Allows users to delete contacts.  
'add contact entity':  
  title: Add entity content  
'view contact entity':  
  title: View entity content  
'edit contact entity':  
  title: Edit entity content  
  description: Allows users to edit contacts.  
'administer contact entity':  
  title: Administer settings  
  description: Allows users to alter the settings related to contacts.  
  restrict access: TRUE
```

Dynamique :

```
permission_callbacks:  
  - \Drupal\node\NodePermissions::nodeTypePermissions
```



# Système de routage / menus

Quelques définitions :

- routage : faire pointer une route (node/{node}) à une action (afficher un noeud)
- chemin (ou path) : route dont les arguments sont définis (ex: node/123 est un chemin, pointant vers la route node/{node} où l'argument est 123)
- lien de menu : Texte (ou titre) pointant vers un chemin
- alias : associe un chemin système (node/123) vers un chemin arbitraire renseigné par le contributeur (mon-noeud)

Les propriétés d'une route

- `_Permissions`
- Les arguments sont nommés ({node}) et peuvent être chargés dans le Controller (en les typant avec une classe)
- Vous pouvez passer des paramètres fixes au controller en les indiquant dans la route



# Les controllers

## Example (src/Controller/HelloController.php)

```
<?php

namespace Drupal\hello_world\Controller;

use Drupal\Core\Controller\ControllerBase;

class HelloController extends ControllerBase {

    /**
     * Display the markup.
     *
     * @return array
     */
    public function content() {
        return [
            '#type' => 'markup',
            '#markup' => $this->t('Hello, World!'),
        ];
    }
}
```



# Les menus

## Type d'élément de menu

- \*.routing.yml -> définit une URL
- \*.links.menu.yml -> lien de menu dans l'arborescence
- \*.links.task.yml -> onglet
- \*.links.action.yml -> "action" (back-office)

Paramètres de routage : <https://www.drupal.org/node/2092643>



# Exemples de routage

```
#.routing.yml
my_module.content:
  path: '/hello'
  defaults:
    _title: "HellWorld"
    _controller: "\Drupal\module\Controller\HelloController::content"
  requirements:
    _permission: 'access my module'

# , = ET
requirements:
  _permission: 'access my module,access content'

# + = OU
requirements:
  _permission: 'access my module+access content'

# Personnalisé
requirements:
  _custom_access: '\Drupal\my_module\MyClass::my_function'
```

La fonction renvoie alors `AccessResult::allowed()` ou `AccessResult::forbidden()`



# TP : le module Premium

- Créer le module Premium
- Créer deux permissions pour les rôles, une pouvant affecter le statut premium aux contenus et l'autre le voir
- Créer un rôle premium avec les permissions créées précédemment.
- Créer une page Suis-je Premium ? Qui affiche « Vous pouvez voir les contenus premium » ou « Vous ne pouvez pas voir les contenus premium »
  - url: suis-je-gold
- Créer une page *Est-il Premium ?* affichant la même chose, mais avec en argument l'uid de l'utilisateur
  - url d'exemple : est-il-gold/2
- Créer une page *Page Premium* avec du contenu "Lorem Ipsum" et ne s'affichant que si l'utilisateur courant a la permission de voir le contenu premium
  - url : page-gold

`\Drupal::currentUser()->hasPermission();` pour vérifier les permissions

```
\Drupal\user\Entity\User::load($uid)->hasPermission()
```



# Gestion des nodes et des users

Quelques fonctions de l'API à connaître :

- `\Drupal::currentUser()` : utilisateur actuellement connecté
- `Node::load()` et `Node::loadMultiple()` pour charger des nœuds
- `User::load()` et `User::loadMultiple()` pour charger des utilisateurs
- `\Drupal::entityTypeManager()->getStorage('node')->loadMultiple($nids);`
- `$entity->save()` pour enregistrer un nœud, un utilisateur, ...
- `$user->getDisplayName()` pour afficher un nom d'utilisateur
- `Node::create(['type' => 'article']->save();`
- `$node->set('body' => ['value' => 'My body']); $node->save();`



# Quelques routes spéciales

- <front>
- <nolink>

Pour voir les routes, utilisez la console : `drupal router:debug`



# Menu





# Les liens

- Menu links
- Action links
- Local tasks
- Contextual links

The screenshot displays a web application interface with the following elements:

- Search:** A search bar with a magnifying glass icon.
- Tools:** A section containing a link labeled "Content Entity Example: Contacts Listing". A blue arrow labeled "Menu link" points to this link.
- Contact Record:** A card for "Blake Hall" with the following details:
  - Name:** Blake Hall
  - Buttons:** "View", "Edit", and "Delete" buttons are highlighted with a red box and a red arrow labeled "Local tasks".
  - + Add Contact:** A button below the record card, highlighted with a blue arrow labeled "Action".
  - First Name:** Blake
  - Gender:** male
  - User Name:** admin



# Menu links

## Fichier .links.menu.yml

```
entity.node_type.collection:  
  title: 'Content types'  
  parent: system.admin_structure  
  description: 'Create and manage fields, forms, and display settings for your  
content.'  
  route_name: entity.node_type.collection  
node.add_page:  
  title: 'Add content'  
  route_name: node.add_page
```



# Action links

## Fichier .links.action.yml

```
node.type_add:  
  route_name: node.type_add  
  title: 'Add content type'  
  appears_on:  
    - entity.node_type.collection  
node.add_page:  
  route_name: node.add_page  
  title: 'Add content'  
  appears_on:  
    - system.admin_content
```



# Local task links

Fichier .links.task.yml

Rendu par défaut en onglets

```
entity.node.canonical:
  route_name: entity.node.canonical
  base_route: entity.node.canonical
  title: 'View'
entity.node.edit_form:
  route_name: entity.node.edit_form
  base_route: entity.node.canonical
  title: Edit
entity.node.delete_form:
  route_name: entity.node.delete_form
  base_route: entity.node.canonical
  title: Delete
  weight: 10
entity.node.version_history:
  route_name: entity.node.version_history
  base_route: entity.node.canonical
  title: 'Revisions'
  weight: 20
entity.node_type.edit_form:
  title: 'Edit'
  route_name: entity.node_type.edit_form
  base_route: entity.node_type.edit_form
entity.node_type.collection:
  title: List
  route_name: entity.node_type.collection
  base_route: entity.node_type.collection
```

```
entity.node.edit_form:
  route_name: entity.node.edit_form
  group: node
  title: Edit

entity.node.delete_form:
  route_name: entity.node.delete_form
  group: node
  title: Delete
  weight: 10
```

Aliquip distineo gemino lucidus ludus occuro quadrum qui tego tum. Augue feugiat jugis neo vereor. Caecus commodo consequat cui du  
pagus utrum vereor. Abdo dolore modo persto quis valde vero. Blandit nutus praemitto quidem turpis. Abdo brevitae ea importunus iu  
populus quibus tamen utrum.

Aliquip conventio jumentum refero ulciscor ullamcorper utrum. Ex gemino laoreet tincidunt vero vicis. Acsi camur esca ille meus pagus pertineo tamen tation ut. Diam esca et ex ratis ullamcorper vulputate. Bene molior ullamcorper. Abluo augue cui hos iaceo loquor luptatum pecus te. Cogo defui genitus ideo luctus. Abigo commoveo eros ex haero minim modo saepius wisi. Esca neo saluto veniam.

Decet eum gemino gravis pertineo premo. Acsi magna nutus tation tincidunt vel verto. Aliquip cui exputo ille natu quis ut vindico wisi. Damnum sed

# Les Plugins





# Les plugins

Le principe de plugins dans Drupal est de permettre au système de fournir une fonctionnalité extensible et remplaçable de manière simple.

Deux concepts clés sont liés aux Plugins :

- Les Plugins
- Les types de Plugins (Plugins Type)

Les blocs sont par exemple des Plugins, chaque bloc en est un. Ils sont du Plugin Type Block.

Les Plugins sont utiles quand il est nécessaire de pouvoir facilement étendre une fonctionnalité générique mais que les implémentations possibles ne partagent que peu de code commun.





# Les types de plugins

Liste des types de plugin :  
drupal debug:plugin

Plugin type	Plugin manager class
action	Drupal\Core\Action\ActionManager
archiver	Drupal\Core\Archiver\ArchiverManager
block	Drupal\Core\Block\BlockManager
ckeditor.plugin	Drupal\ckeditor\CKEditorPluginManager
condition	Drupal\Core\Condition\ConditionManager
display_variant	Drupal\Core\Display\VariantManager
editor	Drupal\editor\Plugin\EditorManager
element_info	Drupal\Core\Render\ElementInfoManager
entity_reference_selection	Drupal\Core\Entity\EntityReferenceSelection\SelectionPluginManager
field.field_type	Drupal\Core\Field\FieldTypePluginManager
field.formatter	Drupal\Core\Field\FormatterPluginManager
field.widget	Drupal\Core\Field\WidgetPluginManager
filter	Drupal\filter\FilterPluginManager
help_section	Drupal\help\HelpSectionManager
image.effect	Drupal\image\ImageEffectManager
language_negotiation_method	Drupal\language\LanguageNegotiationMethodManager
link_relation_type	\Drupal\Core\Http\LinkRelationTypeManager
mail	Drupal\Core\Mail\MailManager



# Les annotations

Très utilisé dans le cœur : pour tous les plugins

<https://www.drupal.org/node/1882526>

<https://api.drupal.org/api/drupal/core%21core.api.php/group/annotation/8.1.x>

# Les blocs





# Les blocs

Fichier src/Pluglin/Block/HelloBlock.php  
Le nom de la classe et du fichier doivent être indentiques.

```
namespace Drupal\hello_world\Plugin\Block;

use Drupal\Core\Block\BlockBase;

/**
 * Provides a 'Hello' Block.
 *
 * @Block(
 *   id = "hello_block",
 *   admin_label = @Translation("Hello block"),
 *   category = @Translation("Hello World"),
 * )
 */
class HelloBlock extends BlockBase {

  /**
   * {@inheritdoc}
   */
  public function build() {
    return array(
      '#markup' => $this->t('Hello, World!'),
    );
  }

}
```



# Ajouter un formulaire au bloc

```
use Drupal\Core\Block\BlockPluginInterface;  
use Drupal\Core\Form\FormStateInterface;
```

```
extends BlockBase implements BlockPluginInterface {
```

```
/**  
 * {@inheritdoc}  
 */  
public function blockForm($form, FormStateInterface $form_state) {  
  $form = parent::blockForm($form, $form_state);  
  
  $config = $this->getConfiguration();  
  
  $form['hello_block_settings'] = array (  
    '#type' => 'textfield',  
    '#title' => $this->t('Who'),  
    '#description' => $this->t('Who do you want to say hello to?'),  
    '#default_value' => isset($config['hello_block_settings']) ? $config['hello_block_settings'] : ''  
  );  
  
  return $form;  
}  
  
public function blockSubmit($form, FormStateInterface $form_state) {  
  parent::blockSubmit($form, $form_state);  
  $values = $form_state->getValues();  
  $this->configuration['hello_block_name'] = $values['hello_block_name'];  
}
```



# Ajouter un formulaire au bloc

```
/**
 * {@inheritdoc}
 */
public function build() {
    $config = $this->getConfiguration();

    if (!empty($config['hello_block_name'])) {
        $name = $config['hello_block_name'];
    }
    else {
        $name = $this->t('to no one');
    }
    return array(
        '#markup' => $this->t('Hello @name!', array(
            '@name' => $name,
        )),
    );
}
```



# Configuration par défaut

Configuration : fichier .yml

config/install/hello\_word.settings.yml

```
hello:  
  name: 'Hank Williams'
```

```
/**  
 * {@inheritdoc}  
 */  
public function defaultConfiguration() {  
  $default_config = \Drupal::config('hello_world.settings');  
  return [  
    'hello_block_name' => $default_config->get('hello.name'),  
  ];  
}
```

Valeur utilisée à l'installation du module (nécessite donc de désinstaller/installer le module après ajout si le module est déjà activé)



# TP : Créer un bloc

Créer un bloc :

- dont le titre côté administration est "Statut premium de l'utilisateur"
- dont l'id (nom machine) est premium\_status
- qui affiche "Vous pouvez voir les contenus premium" ou "Vous ne pouvez pas voir les contenus premium«
- qui affiche un second message configuration (exemple : « Bienvenue »)

Rappel: `\Drupal::currentUser()->hasPermission();` pour vérifier les permissions



# Theming





# Les Render Arrays

Les render arrays sont les blocs constituant une page Drupal. Ce sont des arrays PHP qui définissent des données (c-a-d la structure) ; On est obligés de produire des render arrays. Ceci afin qu'ils puissent être modifiés via les hooks d'altérations ou par la couche de theming.

Les propriétés sont toujours préfixées par un # et la propriété par défaut est #markup, elle permet d'indiquer du balisage simple. Un render array est converti en HTML avec la fonction render();

```
// Un render array simple
'ma_cle1' => array(
  '#markup' => "<h2>Du texte basique</h2>",
),

// Des propriétés utiles
'ma_cle2' => array(
  '#markup' => "Du texte basique",
  '#prefix' => '<h2>',
  '#suffix' => '</h2>',
),
```



# Paramètres du render array et propriétés

Une fonction de #theme peut être renseignée ainsi que ses paramètres (<https://www.drupal.org/developing/api/8/render/arrays>)

```
// Un render array qui produit un tableau HTML
'ma_cle1' => array(
  '#theme' => 'table',
  '#header' => $header,
  '#rows' => $rows,
  '#empty' => "Aucune donnée pour ce tableau",
),
```

Des propriétés utiles :

- #type: Le type d'élément
- #cache: contexts, tags, ... /!\
- #markup: Pour fournir directement de l'HTML
- #pre\_render / #post\_render: agit sur le tableau
- #prefix / #suffix, #weight, #attached, #access, ...



# TP : Manipuler les render arrays

Ajouter un `<h3>` autour d'un block précédent



# TP: Altérer les render arrays

Altérer le bloc dans un `hook_block_view_BASE_ID_alter()` afin de changer le `<h3>` en `<h4>`

-> Impossible à faire directement... Il faut ajouter un `#pre_render` dans le `hook_block_view_alter()`, et la fonction appelée dans le `pre_render` aura accès au `$build` du contenu du bloc.

Exemple : <http://www.drupal8.ovh/en/tutoriels/150/change-drupal-8-powered-by-block>



# Hook\_theme()

Le `hook_theme()` définit des hooks/clés qui pourront ensuite être utilisés via la propriété `#theme` des render arrays. Ces theme hooks généreront ensuite le markup HTML via un template.

On peut fournir des variables à ces theme hooks. Des fonctions preprocess peuvent ajouter ou modifier des variables, et également ajouter des suggestions de templates.



# Déclaration et appel

```
function forum_theme() {  
    return array(  
        'forums' => array(  
            'template' => 'forums',  
            'variables' => array(  
                'forums' => NULL,  
                'topics' => NULL,  
            ),  
        ),  
    );  
}  
  
// Mais toujours privilégier les render arrays, car altérables  
$build['forums'] = array(  
    '#theme' => 'forums',  
    '#forums' => $forums,  
    '#topics' => $topics,  
);
```



Exemples d'utilisations de theme() :

- table
- item\_list
- pager
- links
- Image

[Liste complete des implementation de theme du cœur](#)

```
$image = array(  
  '#theme' => 'image',  
  '#path' => drupal_get_path('module', 'monmodule') . '/monimage.png',  
);
```

<https://www.drupal.org/developing/api/8/render/pipeline#html-main-content-renderer-pipeline>



Un template engine pour PHP  
Créé par Fabien Potencier (Symfony)

[Documentation](#)

Très sécurisé

Extensible



# Exemple de syntaxe

```
<!DOCTYPE html>
<html lang="{{ language.language }}" dir="{{ language.dir }}">
<head>
  <meta charset="utf-8">
  {{ head }}
  <title>{{ head_title }}</title>
  {{ styles }}
</head>
<body class="{{ classes }}" {{ attributes }}>
{{ page_top }}
{{ page }}
{{ scripts }}
{{ page_bottom }}
</body>
</html>
```



# 3 syntaxes à connaître

- `{{ afficher }}`
- `{# commenter #}`
- `{% "programmer" %}`

## TWIG VARIABLE RESOLUTION

---

<p>{{ user.name }}</p>

---

`$user['name']`

`$user->name`

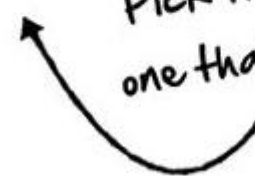
`$user->name()`

`$user->getName()`

`$user->isName()`

**NULL**

*Pick the first  
one that exists*



# Programmer



- `{% if expr %} {% else %} {% endif %}`
- `{% for item in items %} ({% else %}) {% endfor %}`



# Les filtres

- `{{ messages | join(', ') }}`
- `{{ "now" | date('d/m/Y H:i') }}`
- `{{ 'Home' | t }}` (ajouté par Drupal)
- `{{ content.field_date | format_date('short') }}` (ajouté par Drupal)
- Voir `core/lib/Drupal/Core/Template/TwigExtension.php` pour la liste des rajouts Drupal
- Sur toute une section : `{% filter upper %} Texte {% endfilter %}`
- `abs, batch, capitalize, convert_encoding, date, date_modify, default, escape, first, format, join, json_encode, keys, last, length, lower, merge, nl2br, number_format, raw, replace, reverse, round, slice, sort, split, striptags, title, trim, upper, url_encode`



# Les fonctions

`dump(node)`

`{{ max(1, 3, 2) }}`

`{{ random(['pomme', 'orange', 'citron']) }}`

+ celles [fournies par Drupal](#) :

- `link()`
- `path()`
- `url()`
- `attach_library()`



# Inclusion de template

- `{% include 'page.html.twig' with {'foo': 'bar', 'baz': 'bat'} %}`





# Héritage de template

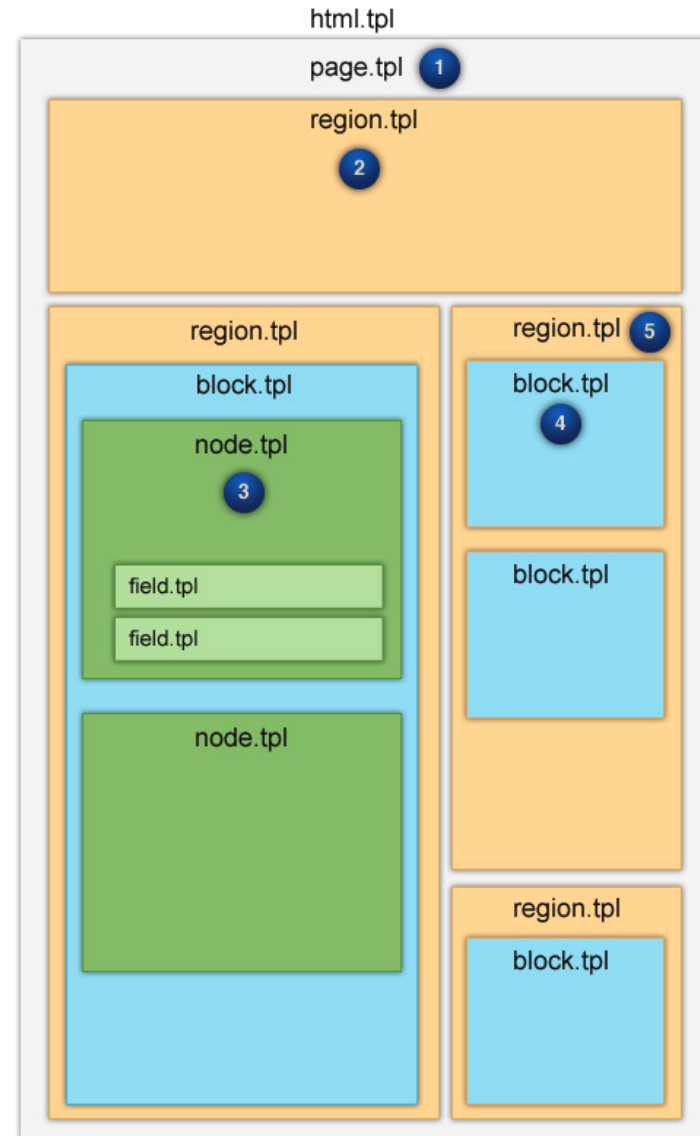
```
{# Template A #}
{% block foo %}
    <p>Mon premier paragraphe</p>
{% endblock %}

{# Template B #}
{% extends 'TemplateA' %}
{% block foo %}
    {{ parent() }}
    <p>Mon deuxième paragraphe</p>
{% endblock %}

{# Rendu Template B #}
<p>Mon premier paragraphe</p>
<p>Mon deuxième paragraphe</p>
```



# Hiérarchie des templates Drupal





# Le nommage des templates (surchage)

- <https://www.drupal.org/node/2354645>
- <https://www.drupal.org/docs/8/theming/twig/working-with-twig-templates>
- Pour Views : <http://redcrackle.com/blog/drupal-8/theme-views-templates>



# TP : Créer un template

Créer un template twig pour la page Suis-je Premium ? avec une couleur de texte différente en fonction du résultat



# Ajouter du contenu auto-généré

Activer le module Devel generate

2 Vocabulaire: 20 caractères

10 Termes : 15 caractères

25 Articles – 10 mots par titre

10 Utilisateurs

1 Menu – 5 liens / 12 Caractères



# Installer un thème contrib

Installer le theme bootstrap



# Structure d'un theme

- **MYTHEME.info.yml** Definition du theme. metadata, style sheets, block regions. requis.
- **MYTHEME.libraries.yml** - JavaScript and CSS librairies.
- **MYTHEME.breakpoints.yml** - Breakpoints - Responsive
- **MYTHEME.theme** - Fonctions de preprocess.
- **screenshot.png** -
- **css/** or **styles/** -
- **js/** or **script/**
- **img/** or **images/**
- **templates/** -



# Le fichier .info.yml

```
core: 8.x
type: theme
base theme: bootstrap

name: 'THEMETITLE'
description: 'Uses the jsDelivr CDN for all CSS and JavaScript. No source files or compiling is necessary and is recommended for simple sites or beginners.'
package: 'Bootstrap'

regions:
  navigation: 'Navigation'
  navigation_collapsible: 'Navigation (Collapsible)'
  header: 'Top Bar'
  highlighted: 'Highlighted'
  help: 'Help'
  content: 'Content'
  sidebar_first: 'Primary'
  sidebar_second: 'Secondary'
  footer: 'Footer'
  page_top: 'Page top'
  page_bottom: 'Page bottom'

libraries:
  - 'THEMENAME/global-styling'
```





# TP : Créer un thème basé sur bootstrap

Créer un répertoire premium à partir du starterkit CDN

Activé le thème premium

Définies dans le fichier montheme.libraries.yml

```
css-stuff:  
  version: VERSION  
  css:  
    theme:  
      css/css-stuff.css: {}  
      css/css-stuff-print.css: { media: print }  
      //fonts.googleapis.com/css?family=Abhaya+Libre|Open+Sans: {  
type: external }
```

Possible « d'attacher » une librairie directement dans le template

```
{{ attach_library('premium/malibjs') }}
```



# TP : Ajout de librairies

- Ajouter une première librairie premium1 au theme qui contient :
  - `css-premium.css` : modifie le background color du body
  - `css-premium-print.css`: modifie la taille du texte et la couleur au media print
  - `js-premium.js`: affiche un message via `console.log()`
- Ajouter une seconde librairie premium2 attaché au template `node.html.twig`:
  - `css-premium2.css` : modifie le background color et la couleur du texte
  - `js-premium2.js`: affiche un message via `console.log()`



# TP : Ajouter une nouvelle region

- Modifier le fichier .info.yml pour ajouter une region footer\_top
- Naviguer dans le module bootstrap et trouver le template page.html.twig
- Copier le dans notre theme (/templates/page/)
- Editer notre page.html.twig pour ajouter notre nouvelle region
- Ajouter un bloc dans la region footer\_top



# Surcharge de template

Exemple avec le block Propulsé par drupal :

```
<!-- THEME DEBUG -->  
<!-- THEME HOOK: 'block' -->  
<!-- FILE NAME SUGGESTIONS:  
  * block--bartik-footer.html.twig  
  * block--system-menu-block--footer.html.twig  
  x block--system-menu-block.html.twig  
  * block--system.html.twig  
  * block.html.twig  
-->
```



# TP : Surcharger le block Premium

Surcharger seulement le template du block Premium

- Ajouter des balises strong autour du titre
- Modifier le h2 du titre en h1



# Fichier .theme et fonction preprocess

function hook\_preprocess\_HOOK()

Permettre de manipuler les données avant le rendu twig

```
/**
 * Implements hook_preprocess_HOOK() for menu.html.twig.
 */
function bartik_preprocess_menu(&$variables) {
  $variables['attributes']['class'][] = 'clearfix';
}
```

```
/**
 * Implements hook_preprocess_HOOK() for maintenance-page.html.twig.
 */
function bartik_preprocess_maintenance_page(&$variables) {
  // By default, site_name is set to Drupal if no db connection is available
  // or during site installation. Setting site_name to an empty string makes
  // the site and update pages look cleaner.
  // @see template_preprocess_maintenance_page
  if (!$variables['db_is_active']) {
    $variables['site_name'] = '';
  }

  // Bartik has custom styling for the maintenance page.
  $variables['#attached']['library'][] = 'bartik/maintenance_page';
}
```



# TP : Ajouter un copyright

- Ajouter une fonction de preprocess afin de manipuler le template page.html.twig
- Ajouter une variable 'copyright' au template.

```
t("Copyright @date", array('@date' => date('Y')))
```

- Modifier le template page.html.twig afin d'ajouter le copyright dans une div avant la balise fermante </footer >





# Les suggestions de template

2 fonctions :

- hook\_theme\_suggestions\_alter()
- hook\_theme\_suggestions\_HOOK\_alter()

HOOK permet de cibler un element par exemple node, page, menu\_link.

```
/**
 * Implements hook_theme_suggestions_HOOK().
 */
function node_theme_suggestions_node(array $variables) {
  $suggestions = [];
  $node = $variables['elements']['#node'];
  $sanitized_view_mode = strstr($variables['elements']['#view_mode'], '.', '_');

  $suggestions[] = 'node__' . $sanitized_view_mode;
  $suggestions[] = 'node__' . $node->bundle();
  $suggestions[] = 'node__' . $node->bundle() . '__' . $sanitized_view_mode;
  $suggestions[] = 'node__' . $node->id();
  $suggestions[] = 'node__' . $node->id() . '__' . $sanitized_view_mode;

  return $suggestions;
}
```



# TP : Ajouter une suggestion

Ajouter 2 suggestion de template pour les node si l'utilisateur est connecté :

```
$suggestions[] = 'node__' . $bundle . '__logged_in';  
$suggestions[] = 'node__' . $view_mode . '__logged_in';
```



# Ajouter des paramètres au theme

.theme :

```
function hook_form_system_theme_settings_alter(&$form, \Drupal\Core\Form\FormStateInterface $form_state)
{
    // Add a checkbox to toggle the breadcrumb trail.
    $form['breadcrumb'] = array(
        '#type' => 'checkbox',
        '#title' => t('Display the breadcrumb'),
        '#default_value' => theme_get_setting('breadcrumb'),
        '#description' => t('Show a trail of links from the homepage to the current page.'),
    );
}
```

theme/config/install/theme.settings.yml

```
breadcrumb: value
```

Variables récupérables ensuite dans les fichiers php du theme :

```
$breadcrumb = theme_get_setting('breadcrumb');
```



# TP : Ajouter un paramètre

Ajouter un paramètre `'copyright_holder'` au thème.

Vérifier que le champs apparait sur la page de configuration du thème.

Modifier le `preprocess_page` pour afficher la valeur de `copyright_holder` apres la date du copyright

# Créer des formulaires





Un formulaire est une structure déclarative composée d'éléments de la form API. La majeure partie des traitements est effectuée par celle-ci, rendant la création ou la modification de formulaire rapide et sécurisée.

Form API : <https://www.drupal.org/node/2117411>



# Type de formulaire

Formulaire générique : FormBase

Formulaire de configuration : ConfigFormBase

Formulaire de Confirmation : ConfirmFormBase



# Les méthodes

```
// Déclaration
public function getId() {
    return 'my_module_form';
}

public function buildForm(array $form, FormStateInterface $form_state) {
    $form['submit'] = array(
        '#type' => 'submit',
        '#value' => t('Submit'),
    );
    return $form;
}

public function validateForm(array &$form, FormStateInterface $form_state) {
    // Logique de validation.
    $form_state->setErrorByName('form_field', 'message');
}

public function submitForm(array &$form, FormStateInterface $form_state) {
    // Traitement des données soumises.
}
```





# Traitement des données

Les données soumises et validées sont contenues dans `$form_state->getValue('key')`.

Après exécution du `_submit()`, l'utilisateur est redirigé vers le formulaire vidé de ses valeurs, ou bien vers une route définie par `$form_state->setRedirectUrl($url)`

Chaque formulaire a un identifiant unique qui permet de l'altérer facilement par les autres modules.

## Appeler un formulaire directement depuis le routage

```
example.form:  
  path: '/example-form'  
  defaults:  
    _title: 'Example form'  
    _form: '\Drupal\mymodule\Form\ExampleForm'
```



# SubmitForm()

```
public function submitForm(array &$form, FormStateInterface $form_state) {  
    // Retrieve the configuration  
    $this->configFactory->getEditable('example.settings')  
    // Set the submitted configuration setting  
    ->set('example_thing', $form_state->getValue('example_thing'))  
    // You can set multiple configurations at once by making  
    // multiple calls to set()  
    ->set('other_things', $form_state->getValue('other_things'))  
    ->save();  
  
    parent::submitForm($form, $form_state);  
}
```



# TP: Créer un formulaire

- Créer un formulaire contenant 3 champs avec leur type correspondant et une description:
  - Nom
  - Telephone
  - Email
- Créer une route pour accéder au formulaire (/premium/monform)
- Vérifier que le numéro de téléphone comporte 10 chiffres et que l'adresse email est bien au format [xxxx@xxx.com](#). (sinon afficher une erreur)
- Afficher les informations renseignées via un `drupal_set_message` dans le submit



# TP: hook\_form\_alter

- Modifier la description du champ email.
- Modifier le titre du champ telephone.

# ConfigFormBase



```
namespace Drupal\example\Form;

use Drupal\Core\Form\ConfigFormBase;
use Drupal\Core\Form\FormStateInterface;

/**
 * Configure example settings for this site.
 */
class exampleSettingsForm extends ConfigFormBase {
    /**
     * {@inheritdoc}
     */
    public function getFormId() {
        return 'example_admin_settings';
    }

    /**
     * {@inheritdoc}
     */
    protected function getEditableConfigNames() {
        return [
            'example.settings',
        ];
    }
}
```



# Gestion de la configuration

- `$config = \Drupal::service('config.factory')->getEditable('monmodule.settings');`
- `$config->set('name', value)->save();` pour définir
- `$config = \Drupal::config('monmodule.settings');`
- `$config->get('name');` pour récupérer
- `$config = \Drupal::service('config.factory')->getEditable('monmodule.settings');`
- `$config->clear('name')->save();` pour supprimer une valeur
- `$config = \Drupal::service('config.factory')->getEditable('monmodule.settings')->delete();`

[Documentation de l'API](#)



# TP : Créer un formulaire d'administration

- Créer un formulaire d'administration qui contient un champ message de type textarea.
- Sauvegarder la valeur du champ au submit dans une configuration
- Ajouter une route (/admin/premiumconfig/) et lien de menu vers le formulaire
- Afficher un message (celui du formulaire) de connexion au utilisateur

Utiliser le theme admin (routing) :

options: \_admin\_route: TRUE

Utiliser hook\_user\_login() et drupal\_set\_message()

Récuper une configuration :

```
$config = \Drupal::config(module.adminsettings);
```

```
$config->get('mavariabile');
```



# Les services





Un service est une classe qui propose des fonctionnalités spécifiques et globales à toute l'application. L'exemple le plus couramment utilisé est un service permettant l'envoi de mail depuis n'importe quel endroit de l'application ou l'accès à la base de donnée.

Méthode d'instantiation par un *Service Container* (Composant Symfony).

Un service peut avoir besoin de paramètres ou de l'injection d'autres services qui sont alors définis dans son fichier YAML de déclaration.

Le *core* de Drupal 8 expose lui-même de nombreux services. (`core.service.yml`)

```
# drupal debug:container
```



# Exemple d'utilisation

Exemple d'utilisation d'une classe pour encoder une variable en json :

```
# Controller.php
use Drupal\Component\Serialization\Json;

$json_serializer = new Json();
$json_serializer->encode(...);
```

Définition du service :

```
# core.services.yml
services:
  serialization.json:
    class: Drupal\Component\Serialization\Json
```

Utilisation du service :

```
# Controller.php
$service = \Drupal::service('serialization.json');
$service->encode(...);
```



# Injection de dépendance

*Service Container* permet d'accéder à tous les services de Drupal.

Injection de dépendance : méthode privilégiée pour accéder et utiliser les services.

Plutôt qu'utiliser le Service Container directement, les services sont passé en arguments dans le constructeur ou via setters.



# Exemple d'injection de dépendance

```
<?php
/**
 * @file
 * Contains \Drupal\example\Form\ExampleForm.
 */
namespace Drupal\example\Form;

use Drupal\Core\Form\FormBase;
use Drupal\Core\Form\FormStateInterface;
use Drupal\Core\Session\AccountInterface;
use Symfony\Component\DependencyInjection\ContainerInterface;

/**
 * Implements an example form.
 */
class ExampleForm extends FormBase {
    /**
     * @var AccountInterface $account
     */
    protected $account;

    /**
     * Class constructor.
     */
    public function __construct(AccountInterface $account) {
        $this->account = $account;
    }

    /**
     * {@inheritdoc}
     */
    public static function create(ContainerInterface $container) {
        // Instantiates this form class.
        return new static(
            // Load the service required to construct this class.
            $container->get('current_user')
        );
    }
}
```



# TP: Injection de dépendance

Créer un formulaire RoleUserForm avec champ « Id utilisateur » de type number.  
Récupérer les services entity\_type.manager et current\_user via injection de dépendance.

```
/**
 * The current user account object.
 *
 * @var AccountInterface $account
 */
protected $account;
/**
 * The entity manager. Can be used to load user entities.
 *
 * @var EntityTypeManagerInterface $entityManager
 */
protected $entityManager;
```

Dans la fonction submit :

- récupérer via \$account l'id et le DisplayName de l'utilisateur courant et l'afficher en drupal\_set\_message
- Récupérer l'utilisateur via la valeur du champ « Id utilisateur » en utilisateur \$entityManager et afficher son Username et ses roles.

```
// Charger un utilisateur via son uid
$user = $this->entityManager->getStorage('user')->load($uid)
```



# Créer son service

Il est possible de définir ses propres services via le fichier `monmodule.services.yml`.  
Ce fichier utilise la même structure que le fichier `core.services.yml`.

```
// PriceCalculator.php

namespace Drupal\price\Service;

class PriceCalculator
{
    public function __construct(){}

    public function getFinalPriceHT($products) {}
}
```

```
# price.services.yml
services:
  price.calculator:
    class: Drupal\price\Service\PriceCalculator
    arguments: []
```

```
$priceCalculator = \Drupal::service('price.calculator');
```



# TP : Créer un service

Créer un Service « PremiumService » avec une fonction `isUserPremium($uid)` qui retourne un boolean.

Modifier le contrôleur de la page `/est-il-gold/{uid}` afin d'utiliser le service via Injection de dépendance





# Modifier un service

Il faut implémenter une classe qui étend ServiceProviderBase et qui possède une méthode alter().  
Il suffit ensuite de modifier la Classe du service (setClass) par notre classe.

```
# src/MyModuleServiceProvider.php
namespace Drupal\my_module;

use Drupal\Core\DependencyInjection\ContainerBuilder;
use Drupal\Core\DependencyInjection\ServiceProviderBase;

class MyModuleServiceProvider extends ServiceProviderBase {
  /**
   * {@inheritdoc}
   */
  public function alter(ContainerBuilder $container) {
    // Remplace la classe qui implémente le service "language_manager".
    $definition = $container->getDefinition('language_manager');
    $definition->setClass('Drupal\language_test\LanguageTestManager');
  }
}
```



# TP : Modifier un service

Déclarer une classe PremiumMaintenanceMode qui hérite MaintenanceMode  
Implémenter les fonctions suivante :

```
public function __construct(StateInterface $state) {  
    parent::__construct($state);  
}  
  
public function exempt(AccountInterface $account) {  
    return $account->hasPermission('access site in maintenance mode');  
}
```

Déclarer une classe PremiumServiceProvider qui hérite ServiceProviderBase

Modifier le service maintenance\_mode afin d'utiliser la classe PremiumMaintenanceMode

Modifier la fonction exempt afin d'autoriser un utilisateur au site en mode maintenance (par exemple si UID = 2)

Passer ensuite le site en mode maintenance et vérifier que l'utilisateur a accès au site.

*drupal site:maintenance on*

*drush sset maintenance\_mode 1*

# Base de Données





Schema API permet déclarer des tables dans la base de données sous la forme d'un tableau.

L'API fournit également des fonctions permettant de créer, supprimer modifier des tables, des colonnes, clés ou indexes.



# Créer une table

Fichier \*.install du module.

Implémentation de  
la méthode hook\_schema()

```
function monmodule_schema() {
  $schema['matable'] = array(
    'description' => 'Stores example person entries for demonstration purposes.',
    'fields' => array(
      'pid' => array(
        'type' => 'serial',
        'not null' => TRUE,
        'description' => 'Primary Key: Unique person ID.',
      ),
      'name' => array(
        'type' => 'varchar',
        'length' => 255,
        'not null' => TRUE,
        'default' => '',
        'description' => 'Name of the person.',
      ),
      'age' => array(
        'type' => 'int',
        'not null' => TRUE,
        'default' => 0,
        'size' => 'tiny',
        'description' => 'The age of the person in years.',
      ),
    ),
    'primary key' => array('pid'),
    'indexes' => array(
      'name' => array('name'),
      'age' => array('age'),
    ),
  );
};

return $schema;
}
```



# Ajout de données par défaut

hook\_install()

```
function monmodule_install() {  
  $database = \Drupal::database();  
  // Add a default entry.  
  $fields = array(  
    'name' => 'John',  
    'age' => 0,  
  );  
  $database->insert('matable')  
    ->fields($fields)  
    ->execute();  
  
  // Add another entry.  
  $fields = array(  
    'name' => 'John',  
    'age' => 100,  
  );  
  $database->insert('matable')  
    ->fields($fields)  
    ->execute();  
}
```



# Gérer les montées de versions

hook\_update\_N()

```
use Drupal\Core\Database\Database;
function monmodule_update_8001(&$sandbox) {
  $spec = [
    'type' => 'varchar',
    'description' => "New Col",
    'length' => 20,
    'not null' => FALSE,
  ];
  $schema = Database::getConnection()->schema();
  $schema->addField('matable', 'newcol', $spec);
  $schema->addIndex('matable', ['newcol']);
}
```

drupal update:execute [module|all]  
drush updb



# Database API

## Instancier l'objet Connection

```
$connection = \Drupal::database();
```

## Requête statique

```
$connection = \Drupal::database();  
$query = $connection->query("SELECT id, example FROM {mytable}");  
$result = $query->fetchAll();
```

## Requête avec placehodler

```
$result = $connection->query("SELECT example FROM {mytable} WHERE id = :id", [  
    ':id' => 1234,  
]);
```





# Database API

## Requête en base de données

```
$db = \Drupal::database();  
$result = $db->select();  
$result = $db->insert();  
$result = $db->delete();  
$result = $db->update();  
$result = $db->merge();
```

```
$query = $connection->select('users', 'u');  
  
// Add extra detail to this query object: a condition, fields and a range  
$query->condition('u.uid', 0, '<>');  
$query->fields('u', ['uid', 'name', 'status', 'created', 'access']);  
$query->range(0, 50);
```

```
$result = $query->execute();  
foreach ($result as $record) {  
    // Do something with each $record  
}
```

## Requête sur le modèle objet

```
$ids = \Drupal::entityQuery('user')->condition('name', 'test')->execute();  
$users = User::loadMultiple($ids);
```

<https://www.drupal.org/docs/8/api/database-api/dynamic-queries>

# DBTNG : DataBase The Next Generation (issu de Drupal 7)



```
$results = $db->select('contact', 'c')
->fields('c')
->condition('created', REQUEST_TIME)
->execute() // ->fetch*()
;
foreach ($results as $result) {
  // faire qqch
}
```

## Récupération de résultats :

- `fetchField()` : la première colonne du premier résultat
- `fetchCol()` : la première colonne sous forme d'array
- `fetchAssoc()` : le premier résultat sous forme d'objet
- `fetchAllAssoc()` : tous les résultats sous forme d'objet
- `fetchAllKeyed()` : tous les résultats sous forme de tableau indexé par la 1ere colonne avec pour valeur la 2e



# TP: Création de page utilisateurs premium

Créer préalablement plusieurs utilisateurs avec le rôle Premium.

Créer une page *Utilisateurs premium* (*/utilisateurs-premium*) listant les utilisateurs du site ayant un rôle premium.

Récupérer les utilisateurs ayant un rôle permettant de voir le contenu premium (->condition('roles', '...'))

Les afficher dans un tableau ('#theme' => 'table') avec *Identifiant*, *Nom*, *Uid*.



# TP : Créer une page articles premium

- Modifier plusieurs articles afin d'ajouter une étiquette « Premium »
- Créer une page articles premium (/articles-premium) qui liste les articles avec l'étiquette premium
  - Récupérer les articles publiés avec un terme « Premium »
  - Afficher les articles avec le `#theme item_list`

# Les Evenements





# Les évènements

On "s'inscrit" à un événement (via un service) pour que le système nous appelle automatiquement et qu'on puisse *réagir*.

[EventDispatcher component](#) de Symfony

# Les évènements



## KernelEvents

```
// vendor/symfony/http-kernel/KernelEvents.php

// when a request is beginning to be dispatched
KernelEvents::REQUEST
// when an uncaught exception appears
KernelEvents::EXCEPTION
// when a controller has returned anything that is not a Response
KernelEvents::VIEW
// when a controller has been found for handling a request
KernelEvents::CONTROLLER
// when a response has been created for replying to a request
KernelEvents::RESPONSE
// when a response has been sent
KernelEvents::TERMINATE
// when a response has been generated for a request
KernelEvents::FINISH_REQUEST
```

## Core Events

```
// core/lib/Drupal/Core/Config/ConfigEvents.php
// when information on all config collections is collected
ConfigEvents::COLLECTION_INFO
// when deleting a configuration object
ConfigEvents::DELETE
// when importing configuration to target storage
ConfigEvents::IMPORT
// when validating imported configuration
ConfigEvents::IMPORT_VALIDATE
// when renaming a configuration object
ConfigEvents::RENAME
// when saving a configuration object
ConfigEvents::SAVE

// core/lib/Drupal/Core/Entity/EntityTypeEvents.php
// when a new entity type is created
EntityTypeEvents::CREATE
// when an existing entity type is deleted
EntityTypeEvents::DELETE
// when an existing entity type is updated
EntityTypeEvents::UPDATE

// core/modules/locale/src/LocaleEvents.php
// when saving a translated string
LocaleEvents::SAVE_TRANSLATION

// core/lib/Drupal/Core/Routing/RoutingEvents.php
// when collecting routes to allow changes to them
RoutingEvents::ALTER
// when collecting routes to allow to allow new ones
RoutingEvents::DYNAMIC
// when route building has finished
RoutingEvents::FINISHED
```



# Concrètement



## Une classe pour la réponse à l'évènement

```
namespace Drupal\my_module\EventSubscriber;
use Symfony\Component\EventDispatcher\EventSubscriberInterface;
use Symfony\Component\HttpKernel\KernelEvents;
class MyModuleSubscriber implements EventSubscriberInterface {
    static function getSubscribedEvents() {
        $events[KernelEvents::REQUEST][] = array('my_function');
        return $events;
    }
    function my_function(GetResponseEvent $event) {
        $event->setResponse(new RedirectResponse('http://example.com/'));
    }
}
```

## Un service (fichier my\_module.services.yml)

```
services:
  my_module.redirect_all:
    class: Drupal\my_module\EventSubscriber\MyModuleSubscriber
    tags:
      - {name: event_subscriber}
```



# Que se passe-t-il sur Kernel.request ?

- AuthenticationSubscriber : Charge la session et initialize currentUser().
- LanguageRequestSubscriber : Détecte la langue courante
- PathSubscriber : Convertit l'URL en chemin système
- LegacyRequestSubscriber : Permet de définir un thème par défaut
- MaintenanceModeSubscriber : Affiche la page de maintenance si besoin
- RouteListener : Récupère le router chargé entièrement
- AccessSubscriber : Vérifie que le visiteur a accès à la route



# TP : Créer un EventSubscriber

Afficher un message (`drupal_set_message`) « Kernel events occurred » en s'abonnant à `KernelEvents::REQUEST`.



# Déclencher un évènement

```
$dispatcher = \Drupal::service('event_dispatcher');  
$dispatcher->dispatch('my_object.my_event', $params);
```

# Créer un évènement



```
/**
 * Wraps a node insertion demo event for event listeners.
 */
class NodeInsertExampleEvent extends Event {

    const EXAMPLE_NODE_INSERT = 'event_subscriber_example.node.insert';

    /**
     * Node entity.
     *
     * @var \Drupal\Core\Entity\EntityInterface
     */
    protected $entity;

    /**
     * Constructs a node insertion example event object.
     *
     * @param \Drupal\Core\Entity\EntityInterface $entity
     */
    public function __construct(EntityInterface $entity) {
        $this->entity = $entity;
    }

    /**
     * Get the inserted entity.
     *
     * @return \Drupal\Core\Entity\EntityInterface
     */
    public function getEntity() {
        return $this->entity;
    }
}
```

# Créer un évènement



Event listener class :

```
public function onExampleNodeInsert(NodeInsertExampleEvent $event) {
    $entity = $event->getEntity();
}

/**
 * {@inheritdoc}
 */
public static function getSubscribedEvents() {
    $events[NodeInsertExampleEvent::EXAMPLE_NODE_INSERT][] = ['onExampleNodeInsert'];
    return $events;
}
```

# Créer un évènement



## Déclencher l'évènement

```
\Drupal::service('event_dispatcher')->dispatch(ExampleInsertDemoEvent::EXAMPLE_NODE_INSERT,  
new ExampleInsertDemoEvent($entity));
```



# TP : Créer un évènement

Créer un évènement PremiumMessage avec une propriété message (string).

Créer un formulaire avec un champ textfield qui déclenchera l'évènement à l'envoi en lui passant la valeur du champ en paramètre.

Modifier ensuite l'EventSubscriber pour s'abonner à l'évènement et afficher le message du formulaire (drupal\_set\_message).



# Les entités



# Les entités



Tout dans Drupal 8 est une Entité : Nodes, Comments, Users, and Blocks

Il existe 2 types d'entités :

- Content Entity
- Configuration Entity

# Content Entity



Content Entities sont sauvegardé dans leur propre table de basse de données et sont généralement affichées.

Les Nodes, Blocks sont des Content Entities.

L'exemple le plus simple d'une Content Entity est Node, qui dispose de sa table « node » dans la base de donnée.

# Configuration Entity



Les Configuration Entities sont basées sur la Config API.

Les données des Config Entity sont stockées dans la table « config » de la base de données.

Les Configuration Entities permettent de stocké des données qui ne sont généralement pas affichés comme par exemple le nom du site.



Un Bundle est une variante d'une Configuration Entity.

Ce sont des entités qui améliorent une Content Entity en fournissant des mécanisme pour avoir différents « types » de Content Type.

L'exemple le plus parlant de Bundles dans Drupal sont les Node Types (Content Type).

Les Content Type du module Node sont des Configuration Entity qui stocke différents champs et paramètre pour chaque type de contenu.



# Definition d'une entité

## Exemple de Content :

```
/**
 * Defines the Most Simple entity.
 *
 * @ContentEntityType(
 *   id = "most_simple",
 *   label = @Translation("Most Simple"),
 *   base_table = "most_simple",
 *   entity_keys = {
 *     "id" = "id",
 *     "bundle" = "bundle",
 *   },
 *   fieldable = TRUE,
 *   .....

```

## Exemple de Bundle :

```
/*
 * @ConfigEntityType(
 *   id = "most_simple_type",
 *   label = @Translation("Most Simple Type"),
 *   bundle_of = "most_simple",
 *   entity_keys = {
 *     "id" = "id",
 *     "label" = "label",
 *     "uuid" = "uuid",
 *   },
 *   config_prefix = "most_simple_type",
 *   config_export = {
 *     "id",
 *     "label",
 *   },

```

Exemple complet : <https://www.drupal.org/docs/8/api/entity-api/entity-types>

# Créer une ConfigEntity



## example/src/Entity/Example.php

```
namespace Drupal\example\Entity;
use Drupal\Core\Config\Entity\ConfigEntityBase;

/**
 * Defines the Exampleentity.
 *
 * @ConfigEntityType(
 *   id = "example",
 *   label = @Translation("Example"),
 *   config_prefix = "Example",
 *   entity_keys = {
 *     "id" = "name",
 *     "label" = "label",
 *   }
 * )
 */
class Example extends ConfigEntityBase {

  /**
   * The Example label.
   *
   * @var string
   */
  public $label;

  /**
   * The Example machine readable name.
   *
   * @var string
   */
  public $id;
}
```



# Créer une ConfigEntity

## Configuration schema file

**example/config/schema/example.schema.yml**

```
example.example.*:  
  type: config_entity  
  label: 'Example config'  
  mapping:  
    id:  
      type: string  
      label: 'ID'  
  label:  
    type: label  
    label: 'Label'
```





# TP : Créer une ConfigEntity

- Créer un nouveau module premium\_partner.
- Créer dans ce module une ConfigEntity « Partner » (schema + Entity) qui dispose de 3 champs :
  - label : string
  - id : string
  - url : string
- Créer ensuite un formulaire (/premium/partner) qui :
  - Affiche dans un tableau (#type => table) les sites partenaire (label, id, url).
  - Permet d'ajouter un site partenaire

drush entity-updates / drupal update:entities : Mettre à jour le schéma de base de données



# Créer une entité via la Drupal Console

drupal generate:entity:content --module monmodule

```
1 - modules/custom/cars/cars.permissions.yml
2 - modules/custom/cars/cars.links.menu.yml
3 - modules/custom/cars/cars.links.task.yml
4 - modules/custom/cars/cars.links.action.yml
5 - modules/custom/cars/src/CarsAccessControlHandler.php
6 - modules/custom/cars/src/CarsTranslationHandler.php
7 - modules/custom/cars/src/Entity/CarsInterface.php
8 - modules/custom/cars/src/Entity/Cars.php
9 - modules/custom/cars/src/CarsHtmlRouteProvider.php
10 - modules/custom/cars/src/Entity/CarsViewsData.php
11 - modules/custom/cars/src/CarsListBuilder.php
12 - modules/custom/cars/src/Form/CarsSettingsForm.php
13 - modules/custom/cars/src/Form/CarsForm.php
14 - modules/custom/cars/src/Form/CarsDeleteForm.php
15 - modules/custom/cars/cars.page.inc
16 - modules/custom/cars/templates/cars.html.twig
17 - modules/custom/cars/src/Form/CarsRevisionDeleteForm.php
18 - modules/custom/cars/src/Form/CarsRevisionRevertTranslationForm.php
19 - modules/custom/cars/src/Form/CarsRevisionRevertForm.php
20 - modules/custom/cars/src/CarsStorage.php
21 - modules/custom/cars/src/CarsStorageInterface.php
22 - modules/custom/cars/src/Controller/CarsController.php
23 - modules/custom/cars/templates//cars-content-add-list.html.twig
24 - modules/custom/cars/cars.module
25 - modules/custom/cars/cars.module
26 - modules/custom/cars/config/schema/cars_type.schema.yml
```



# TP : Créer une entité via la Drupal Console

- Créer une entité Dossier via la commande drupal.
- Rajouter ensuite 2 champs :
  - Description
  - Articles (Reference vers des contenus Articles)



## Créer des entités

```
$node = entity_create('node', array(  
  'title' => 'New Article',  
  'body' => 'Article body',  
  'type' => 'article',  
));
```

```
$node = Node::create(array(  
  'title' => 'New Article',  
  'body' => 'Article body',  
  'type' => 'article',  
));
```

```
$node = Node::create([  
  'type' => 'article',  
  'title' => 'A new article',  
]);  
$node->save();
```

## Charger des entités

```
$node = entity_load('node', $id);  
$node = Node::load($id);
```

## Créer des entités

```
$node = entity_create('node', array(  
  'title' => 'New Article',  
  'body' => 'Article body',  
  'type' => 'article',  
));
```

```
$node = Node::create(array(  
  'title' => 'New Article',  
  'body' => 'Article body',  
  'type' => 'article',  
));
```

```
$node = Node::create([  
  'type' => 'article',  
  'title' => 'A new article',  
]);  
$node->save();
```

## Charger des entités

```
$node = entity_load('node', $id);  
$node = Node::load($id);
```



## Lire les champs d'une entité

```
// text field
$node = Node::load(4);
$txt = $node->field_my_text->value;

// entity reference
$node = Node::load(3);
$tags = $node->field_tags->referencedEntities();
```

## Mettre à jour une entité

```
$node = Node::load(4);
$node->field_my_text = "updated text";
$node->save();
```



# Drupal::httpClient

# httpClient



## Récupérer le client

```
$client = \Drupal::httpClient();
```

## Effectuer une requête

```
$client->request('GET', 'http://demo.ckan.org/api/3/action/package_list');
```

## Requête GET + réponse

```
$request = $client->get('http://demo.ckan.org/api/3/action/package_list');  
$response = json_decode($request->getBody());
```

## POST + réponse

```
$client = \Drupal::httpClient();  
$request = $client->post('http://demo.ckan.org/api/3/action/group_list', [  
    'json' => [  
        'id' => 'data-explorer'  
    ]  
]);  
$response = json_decode($request->getBody());
```



# TP : Création d'entité Contact via httpClient – 1



- Créer un nouveau module contacts.
- Créer une entité Contact.
- Ajouter ensuite à l'entité 3 champs via l'interface admin :
  - Username : champ text
  - Email : champ text
  - website : champ text



# TP : Création d'entité Contact via httpClient – 2

- Créer un formulaire :
  - Avec un tableau listant les entités Contact comprenant 4 colonnes :
    - Name
    - Username
    - Email
    - Website
  - Un champ ID (number) avec comme valeur possible mini 1 et max 10.
- Créer une entité via l'interface d'administration et vérifier son affichage dans le formulaire



# TP : Création d'entité Contact via httpClient – 3

- Dans la fonction Submit du formulaire :
  - Récupérer la valeur du champ ID
  - Faire une requête GET sur l'url :  
[https://jsonplaceholder.typicode.com/users/\[ID\]](https://jsonplaceholder.typicode.com/users/[ID])
  - Avec la réponse de la requête précédente créer une nouvelle entité Contact en remplissant les différents champs.
- Vérifier ensuite en soumettant un id au formulaire la création d'une nouvelle entité.

Exemple de réponse : <https://jsonplaceholder.typicode.com/users/1>