Drupal

Bonnes pratiques

L'ÉQUIPETECH

Fabien Clément

L'ÉQUIPETECH

- Contributeur Drupal depuis + de 10 ans.
- Core contributeur Drupal 8.
- Core contributeur Drupal Commerce 1.x et 2.x.
- Contributeur de modules.
- Lead developer pendant 3 ans chez Commerce Guys.
- Directeur technique associé L'Équipe.tech

GozOo

Goz

L'ÉQUIPETECH

Sommaire

L'ÉQUIPETECH

Sommaire

- Les bonnes pratiques en Drupal 8
- Les bonnes pratiques en Drupal 8
- Les bonnes pratiques en Drupal 8
- Les bonnes pratiques en Drupal 8
- Les bonnes pratiques en Drupal 8

L'ÉQUIPETECH

Les bonnes pratiques en Drupal 8

L'ÉQUIPETECH

Règle n°1 - DON’T HACK

Si c’est du **core** ou de la **contrib**, on ne touche pas aux fichiers !

ET PIS C’EST TOUT!

Et si vous arrivez sur un projet existant: Module Hacked

Règle n°2 - DON’T HACK

ET PIS C’EST TOUT!

Règle n°2 (la vraie) - Arborescence

- Placer les modules contrib dans /modules/contrib
- Placer les modules custom dans /modules/custom
- Placer les thèmes contrib dans /themes/contrib
- Placer les thèmes custom dans /themes/custom

- Suivre PSR-4

Règle n°3 – Suivre les bonnes pratiques Symfony

- Suivre les bonnes pratiques de Symfony
- Suivre les normes PSR
- Utiliser l'injection de dépendance
- Étendre plutôt que surcharger si possible

Règle n°3bis – L'injection de dépendance

- Posez-vous des questions si vous voyez :

```
$monService = Drupal::service('entity_type.manager');
```

- À la place: déclarez votre injection de dépendance via le fichier services.yml si le service y est déclaré, ou sinon via la méthode `create()`.

3b

Règle n°3bis – L'injection de dépendance

```php
<?php

namespace Drupal\monmodule;

use Drupal\Core\Entity\EntityTypeManagerInterface;

class MaClass() {

  /**
   * The entity type manager.
   *
   * @var \Drupal\Core\Entity\EntityTypeManagerInterface
   */

  protected $entityTypeManager;


  /**
   * The constructor.
   *
   * @param \Drupal\Core\Entity\EntityTypeManagerInterface $entity_type_manager
   *   The entity type manager.
   */

  public __construct(EntityTypeManagerInterface $entity_type_manager) {

    $this->entityTypeManager = $entity_type_manager;

  }

}
```

L'ÉQUIPETECH

3b Règle n°3bis – L'injection de dépendance

```
# monmodule.services.yml

services:
  monmodule.maclass:
    class: Drupal\monmodule\MaClass
    arguments: ['@entity_type.manager']
```

OU

```
<?php

namespace Drupal\monmodule;

use Drupal\Core\Entity\EntityTypeManagerInterface;
use Drupal\Core\DependencyInjection\ContainerInjectionInterface;
use Symfony\Component\DependencyInjection\ContainerInterface;

class MaClass() {

  (…)


  /**
   * {@inheritdoc}
   */
  public static function create(ContainerInterface $container) implements
ContainerInjectionInterface {
    return new static(
      $container->get('entity.manager')
    );
  }


}
```

Règle n°4 – Au revoir .module

- (Quasiment) plus besoin de .module.
- Peu de hooks sont encore utiles, la majorité peut être réalisée via l'héritage Symfony et la POO.
- Posez-vous donc la question 3 fois si vous voulez mettre quelque chose dans le .module.

Règle n°5 – Penser automatisation

- Tout doit pouvoir être automatisé, aucun action manuelle ne doit être faite pour modifier les autres environnements (locaux, dev, preprod, prod etc)
- Utiliser Git
- Scripter les procédures
- S'il s'agit de contenu, utiliser des solutions d'import telles que:
  - Module Migrate
  - Module Content staging
  - Module Default content
  - Custom via `hook_update()`.
  - Custom via une commande drush/drupal-console

Règle n°5 – Penser automatisation

- Le fichier `settings.php` ne doit contenir que les informations communes à l'ensemble des environnements.
- Pour la configuration spécifique à chaque environnement, décommenter les lignes en bas du fichier `settings.php` et créer un fichier `sites/default/settings.local.php` qui **ne doit pas être versionné**.

```
# if (file_exists($app_root . '/' . $site_path . '/settings.local.php')) {
#   include $app_root . '/' . $site_path . '/settings.local.php';
# }
```

- Utiliser les variables d'environnement.

Règle n°6 – Multilingue

- Drupal est Multilingue by design.
- Toujours passer une chaine de caractère dans le service de traduction (même pour un site d'une seule langue).
  Cela permettra au client de modifier la chaine via l'interface de traduction si besoin.

```
// Si la méthode n'est pas déjà implémentée, import du trait.
use Drupal\Core\StringTranslation\StringTranslation;

class MaClass() {
  trait StringTranslation;

  public customFunction() {
    $string = $this->t('Drupal c'est de la balle');
  }
}
```

7

Règle n°7 – Gestion des droits

- Utiliser la gestion de droits de Drupal.
- Ajouter ses propres droits plutôt que de conditionner sur les rôles :

```
# monmodule.permissions.yml

acces my functionality:

  title: "Accéder à ma fonctionnalité"
```

- Utiliser les permissions lors du routage :

```
# monmodule.routing.yml

monmodule.myfunctionality:
  path: '/my-functionality'
  defaults:
    _controller: '\Drupal\monmodule\Controller\MyFunctionalityController'
    _title: 'Faire le café'
  requirements:
    _permission: 'acces my functionality'
```

Règle n°7 – Gestion des droits

- Si plus de complexité, gérer via une gestion d'accès personnalisée :

```
# monmodule.routing.yml

monmodule.myfunctionality:
  path: '/my-functionality'
  defaults:
    _controller: '\Drupal\monmodule\Controller\MyFunctionalityController'
    _title: 'Faire le café'
  requirements:
    _custom_access: '\Drupal\monmodule\Controller\MyFunctionalityController:checkAccess'
```

- S'il s'agit de permissions propres à des actions d'entité, surcharger la définition de celle-ci pour compléter sa gestion des droits.

Règle n°8 – Routing

- Utiliser des chemins avec paramètres plutôt que des arguments :

`/my-functionality/1` `/my-functionality?node=1`

```
# monmodule.routing.yml

monmodule.myfunctionality.node:
  path: '/my-functionality/{node}'
  defaults:
    _controller: '\Drupal\monmodule\Controller\MyFunctionalityController'
    _title: 'Faire le café'
  requirements:
    node: \d+
```

- Bonus: en ajoutant l'entité node à "requirements", cette entité sera directement chargée en paramètre du controller plutôt qu'un simple identifiant.

Règle n°9 – Liens de menu d'administration

- Pour les pages d'administration, utiliser le fichier `monmodule.links.menu.yml` pour gérer les éléments de menu et leur hiérarchie.

```
# monmodule.links.menu.yml

monmodule.admin_dashboard:
  title: "Tableau de bord personnalisé"
  description: "Tableau de bord personnalisé pour mes fonctionnalités de café"
  parent: system.admin
  route_name: monmodule.admin_dashboard
```

10 Règle n°10 – Entités

- Ajouter des propriétés links aux entités (custom ou existantes)

```
/**
 * Defines the node entity class.
 *
 * @ContentEntityType(
 *   id = "node",
 *   (…)
 *   links = {
 *     "canonical" = "/node/{node}",
 *     "delete-form" = "/node/{node}/delete",
 *     "delete-multiple-form" = "/admin/content/node/delete",
 *     "edit-form" = "/node/{node}/edit",
 *     "version-history" = "/node/{node}/revisions",
 *     "revision" = "/node/{node}/revisions/{node_revision}/view",
 *     "create" = "/node",
 *   }
 * )
 */
class Node extends EditorialContentEntityBase implements NodeInterface {}
```

10 Règle n°10 – Entités

- Ce qui permet de les appeler simplement :

```
/** @var \Drupal\node\NodeInterface $node */
$url = $node->toUrl('canonical');
```

- Pour étendre les fonctionnalités d'une entité (storage, access, form handlers etc), ajouter ou surcharger des définitions via *hook_entity_info_alter()*. (Exception à la règle n°4).
- Plutôt que de créer des tables custom, Privilégiez les entités pour profiter :
  - Utilisation de views pour les pages d'administration / listing.
  - CRUD opérationnel
  - Intégration automatique avec les fonctionnalités core et contrib (ex: REST).

Règle n°11 – Requêtes

- Drupal fournit des outils pour générer les requêtes en s'affranchissant du type de base utilisé. Pour une requête simple sur une entité :

```
$entity_ids = \Drupal::entityQuery('node')->condition('uuid', $uuids, 'IN')->execute();
```

- Ne pas faire de requête en dur mais utiliser le générateur de requêtes.

```
$query = db_query('SELECT n.nid FROM {node} n WHERE n.type = @type', ['article']);


/** @var \Drupal\Core\Database\Connection $connection */
$query = $connection->select('node', 'n');
$query->fields('n', ['nid']);
$query->condition('n.type', 'article', '=');
```

Règle n°11 – Requêtes

- Utiliser si possible les méthodes de stockage des entités pour récupérer les données plutôt que des requêtes.

```
$entities = $this->entityTypeManager->getStorage('node')->loadByProperties('type' => 'article');
```

- Dans le cas où une requête est nécessaire, ne sélectionner que l'id de l'entité puis charger les entités complètes plutôt que de récupérer une ou plusieurs colonnes spécifiques. Drupal cache les entités complètes et récupèrera donc les informations utiles d'un coup.

```
$entities = $this->entityTypeManager->getStorage('node')->loadMultiple($entity_ids);
```

- Préférer un `->loadMultiple()` plutôt que plusieurs `->load()`.

Règle n°11 – Requêtes

- Inutile de faire une requête pour obtenir des entités référencées

```
$referenced_entity = $entity->get('field_reference')->
first()->get('entity')->getTarget()->getValue();
```

- Utiliser `->first()` plutôt que `->get(0)`.

Règle n°12 – Templates

- Aucun code markup ne doit être présent dans une classe.
- Passer par la création de template et l'utilisation de template twig pour générer toute sortie d'affichage.

```
// monmodule.module (deuxième exception à la règle n°4).
/**
 * Implements hook_theme().
 */
function monmodule_theme() {
  return [
    'monmodule_custom_display' => [
      'variables' => ['content' => NULL],
    ],
  ];
}

# templates/monmodule-custom-display.html.twig
<div>{content}</div>
```

Règle n°12 – Templates

- Ne pas faire de métier dans les templates.
- Aucune requête dans les templates.
- Aucune requête dans les preprocess.
- Un preprocess ne sert qu'à filtrer/ordonner/modifier les données à envoyer au template.

Règle n°13 – Charger un asset

- Définir les assets dans un fichier `monmodule.libraries.yml`.

```
custom_display:
  version: 1.x
  css:
    base:
      css/custom_display.css: {}
  js:
    js/custom_display.js: {}
```

- Attacher cet asset au tableau de rendu à l'endroit où on en a besoin.

```
$render_array['#attached'] = ['library' => ['monmodule/custom_display']];
```

Règle n°14 – Gestion des caches

- Ne jamais désactiver les caches.
- Caches activés par défaut sur Drupal 8.
- Utiliser les caches tags pour gérer l'invalidation des caches.
- Créer ses propres cache tags et exécuter leur invalidation si besoin.

Règle n°15 – Utilisateurs de test

- Éviter au maximum d'être connecté en admin lors du développement.
- Tester les fonctionnalités avec un utilisateur ayant le rôle qui utilisera réellement cette fonctionnalité.
- Tester en anonyme.
- Rappel: le super admin à tous les droits et passe certaines couches de résolution des droits.

Règle n°16 – Nommage des champs

- Inutile de préfixer systématiquement le nom des champs par l'entité ou le bundle.
- Pratique sous Drupal 7, inutile sous Drupal 8.

Sous D8, les champs sont vus tels que:

- Stockage : field.storage.[entity].field_[nom_champ]
- Instance de bundle : field.field.[entity].[bundle].field_[nom_champ]

Règle n°16bis – Nommage des champs

- Comme les variables, le nom des champs doit renseigner son utilité.
- Utiliser le pluriel s'il s'agit d'un champ multiple, et le singulier s'il s'agit d'un champ simple.

Règle n°17 – Composer

- `Composer update`: met à jour le fichier composer.lock avec les dernières versions correspondant aux versions spécifiées dans le composer.json
- `Composer install` : Télécharge les modules/librairies avec la version définie dans le composer.lock.

Lors du développement, utiliser `composer update` pour se mettre à jour.

Versionner le fichier composer.json ET composer.lock

Sur tous les autres environnements, utiliser uniquement `composer install` pour être sûr de n'avoir que la version testée et approuvée auparavant.

Règle n°18 – Contrib Vs Custom

- Vérifier qu'un module contrib ne fait pas déjà ce que vous voulez faire avant de vous lancer.
- S'il existe:
  - Vérifier le nombre d'utilisation
  - S'il est maintenu / Le nombre et le types des problèmes remontés dans les issues
  - Regarder le code (si vous êtes développeur)
  - Le module vaut-il vraiment le coup au regard du nombre de fonctionnalités qu'il propose (usine a gaz) ?
- Plus vous avez de modules, plus votre site s'allourdi

Règle n°19 – Validation des données

- En cas de Javascript, valider également les données côté serveur.
- Utiliser la Form-API
- Utiliser les fonctionnalités Ajax de Drupal.
- Utiliser les méthodes de nettoyage de Drupal :
  - Html::escape() : Plain text
  - Xss::filter() : Pour du texte qui autorise le contenu HTML
  - Xss::filterAdmin() : Texte soumis par un admin qui peut utiliser plus de HTML.

Règle n°presque 20 – Performances

Utiliser les outils disponibles (ou équivalents) :

- Memcache/Redis : Sortir la gestion des caches de la BDD.
- Varnish
- Opcode
- Solr/Elasticsearch pour la recherche
- CDN
- ~~Module Entity Cache~~

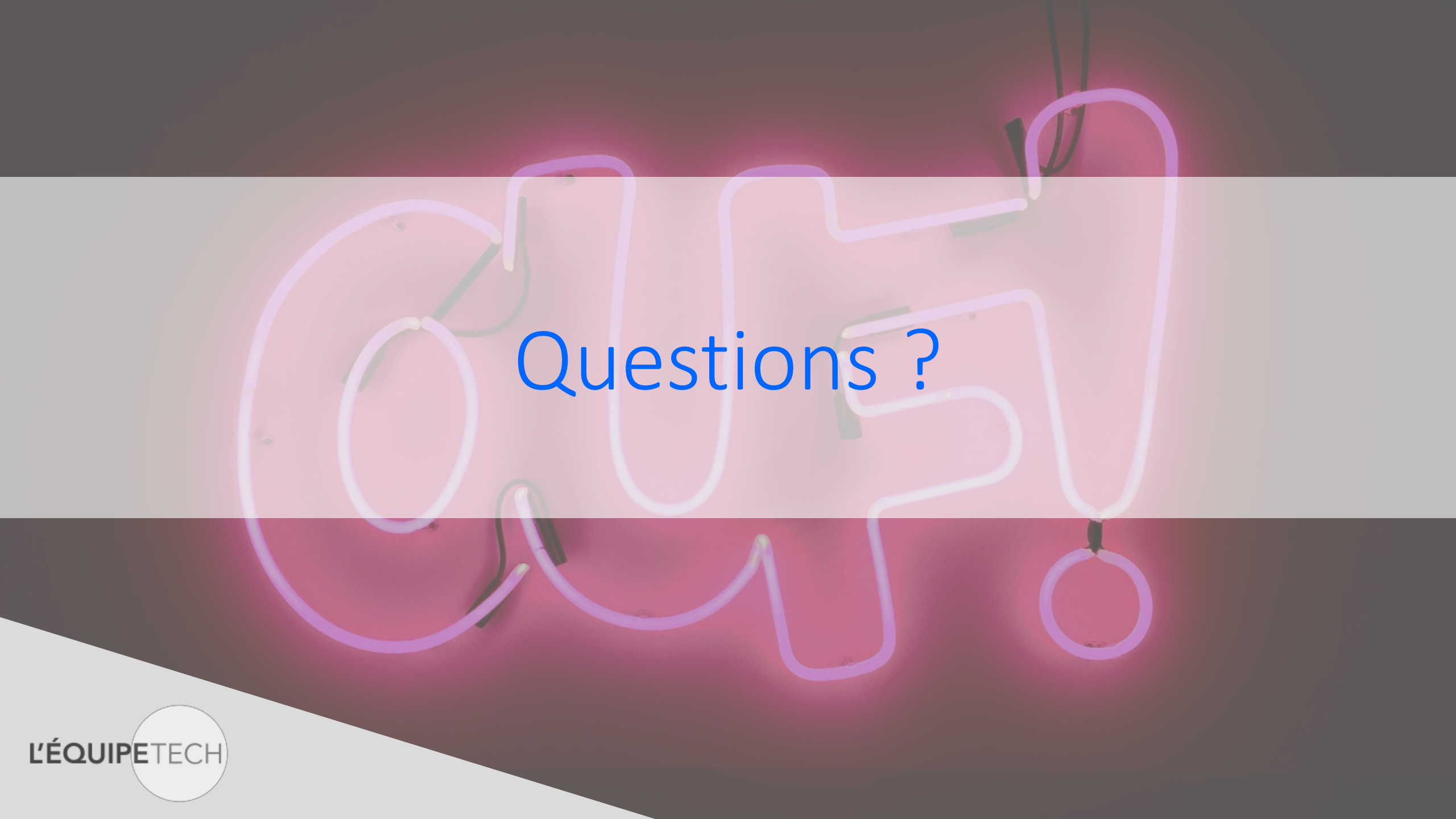
Règle n°20 – Clear cache

KEEP CALM
AND
CLEAR CACHE

L'ÉQUIPETECH

Gibbs-Slap

Si tu ne le fais pas pour toi, rappelles-toi qu'il y a toujours un Gibbs derrière toi !



Questions ?

L'ÉQUIPETECH