Drupal

Présentation et 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

L'ÉQUIPETECH

Qu’est-ce que Drupal ?

L’ÉQUIPETECH

Drupal

- Plateforme de gestion de contenu (CMS) Open-Source en PHP
- Plusieurs centaines de modules
- Des milliers de contributeurs au core, et encore + pour les modules, partout dans le monde.
- Des milliers d'entreprises contributrices, et encore + utilisatrices.
- Une communauté avec un slogan: « Come for the code, stay for the community »
- Des événements locaux, nationaux et internationaux plusieurs fois par an.

Pour créer son petit site

- Sites institutionnels (Impots.gouv.fr, CNIL, White House, Tesla etc)
- Sites E-commerce (Mac Donald, Mk2, Effia, Interflora etc)
- Sites associatifs (Unicef, Croix rouge, Chiens guides etc)
- … et bien d’autres

Il était une fois

L'ÉQUIPETECH

Il était une fois…

- Créé en 2000 par Dries Buytaert en Belgique
- Devait s'appeler Dorp (Village en allemand) mais une faute de frappe lors de son enregistrement le change en drop.org.
- En 2001, renommé en Drupal (dérivé de la prononciation anglaise du mot allemand druppel qui veut dire Drop).

Il était une fois…

- Devient populaire à partir de sa version 4.x.
- Innovant dès la version 4.x grâce aux modules :
  - Views (4.6.x) : Interface pour créer des pages de listing
  - CCK (4.7.x) : Ajout de champs aux contenus
- Popularité croissante de la version 5.x à aujourd'hui
  - 5.x : 2007
  - 6.x : 2008
  - 7.x : 2011 (fin de vie en novembre 2021)
  - 8.x : 2015 (fin de vie en novembre 2021 lié à la fin de vie de Symfony 3)
  - 9.x : 2020

Ce qu'il faut retenir de Drupal 6.x

- Version ayant connu les premiers grosses popularités (white-house.org, sites de media).
- Nombreux modules dont les plus gros faisant sa popularité:
  - Views
  - CCK
  - Features: Export/import de configuration en fichier pour faciliter les déploiements

Ce qu'il faut retenir de Drupal 7.x

- Intégration de la notion de champs (ancien cck).
- Refonte du principe de nodes et création de la notion d'entités.
- Continue sur ses succès et augmente encore sa visibilité.
- Amélioration de la traduction: Granularité de la traduction par champ.

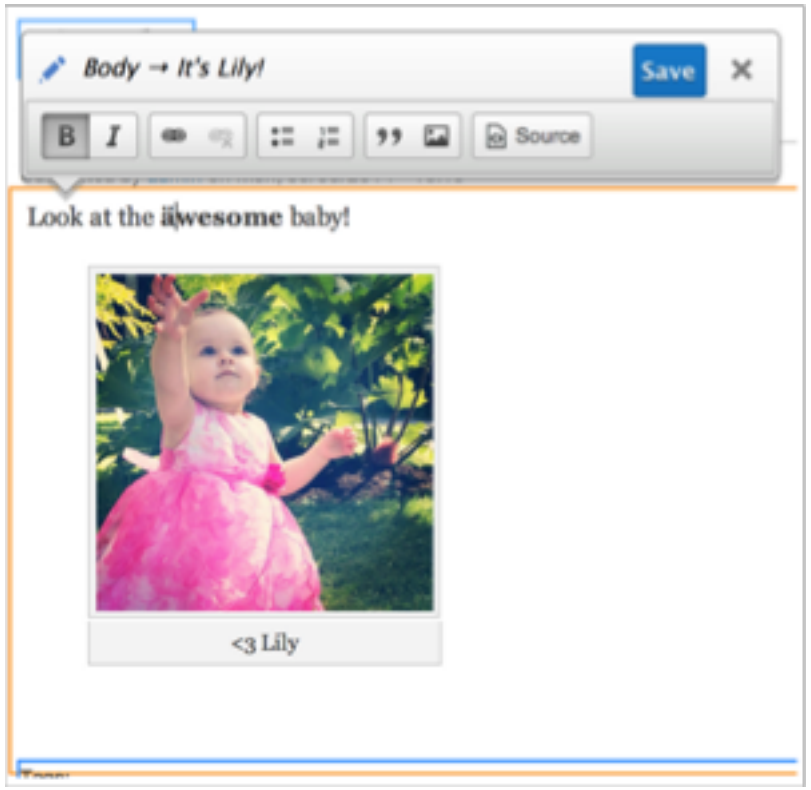
Ce qu'il faut retenir de Drupal 8.x

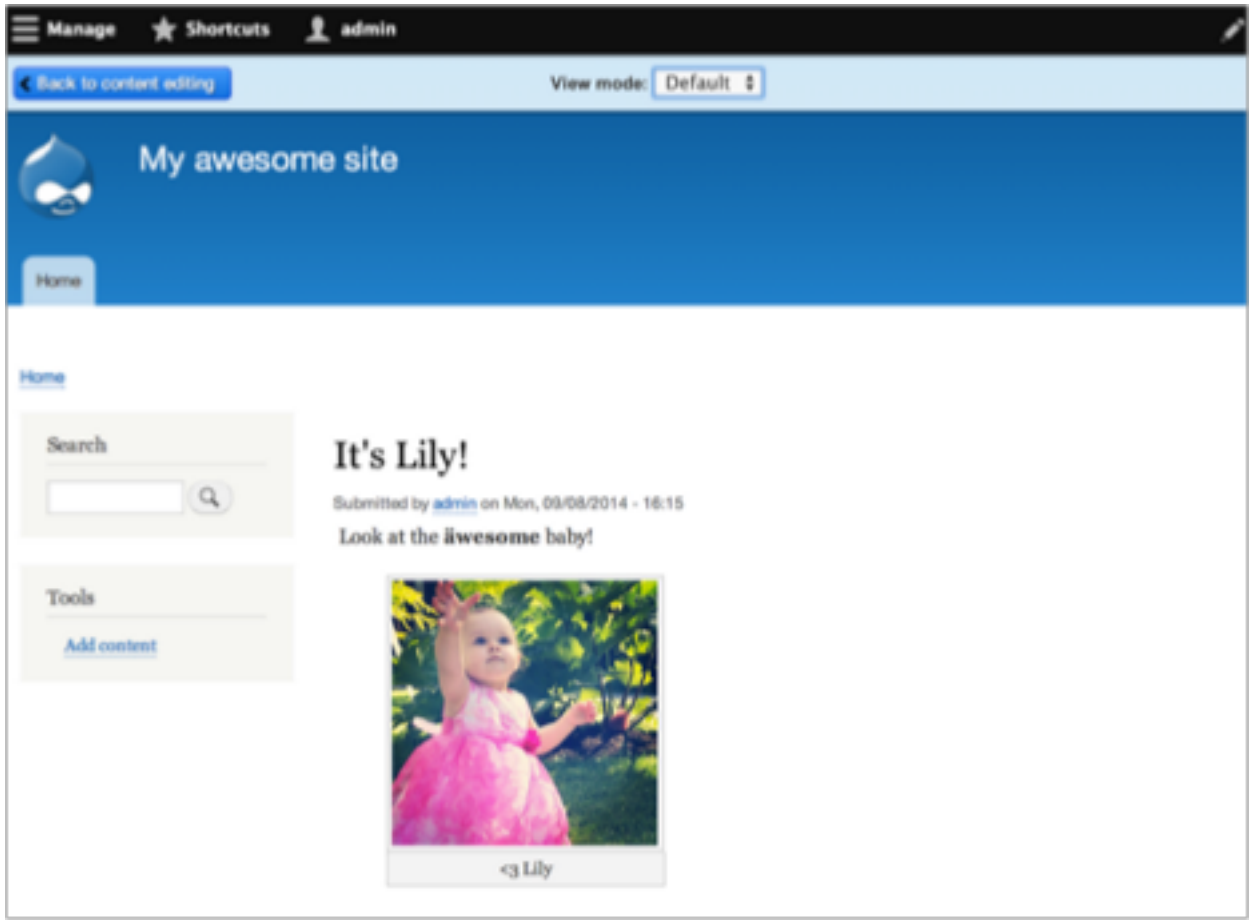
- Intégration de nombreux modules au cœur dont views.
- Refonte complète sur un modèle POO et utilisation de composants Symfony 3.x.
- Amélioration des entités et ajout de la notion de plugins.
- Ajout de nombreuses fonctionnalités pour améliorer l'expérience utilisateur et contributeur.
- Lancement des initiatives (UX, accessibilité, contribution etc).
- Configuration gérée via fichiers.

Des fonctionnalités de Drupal 8

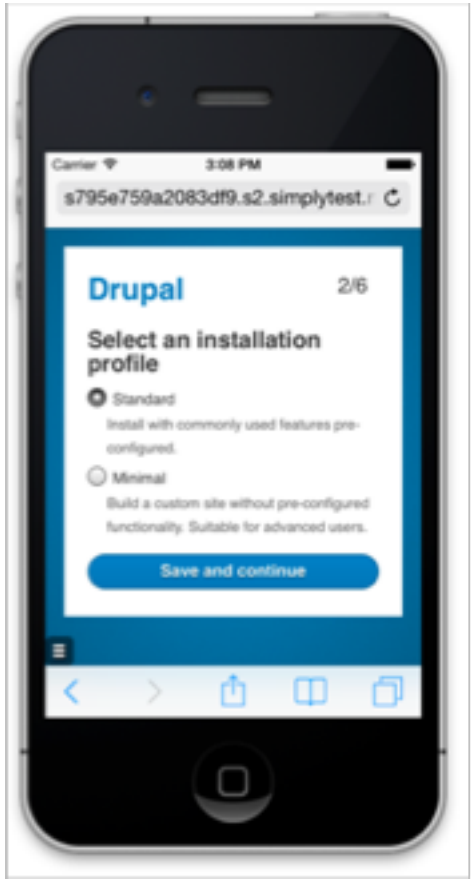
L'ÉQUIPETECH

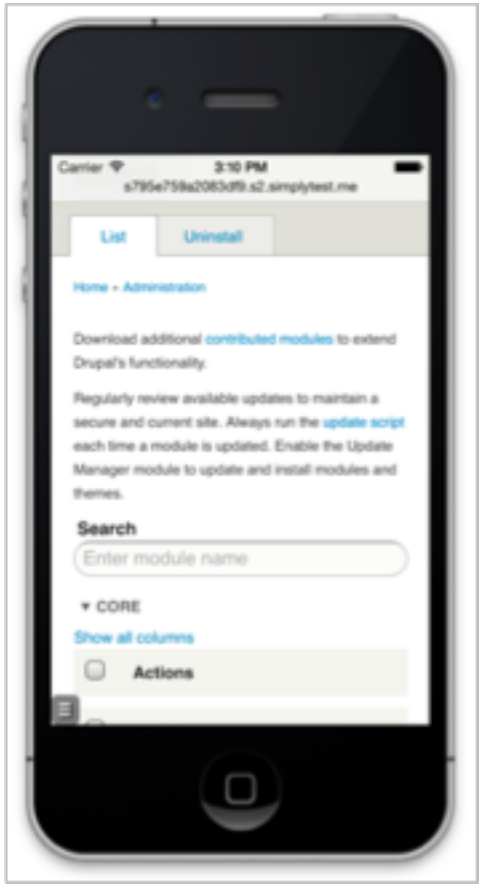
Édition « en ligne » ou prévisualisation



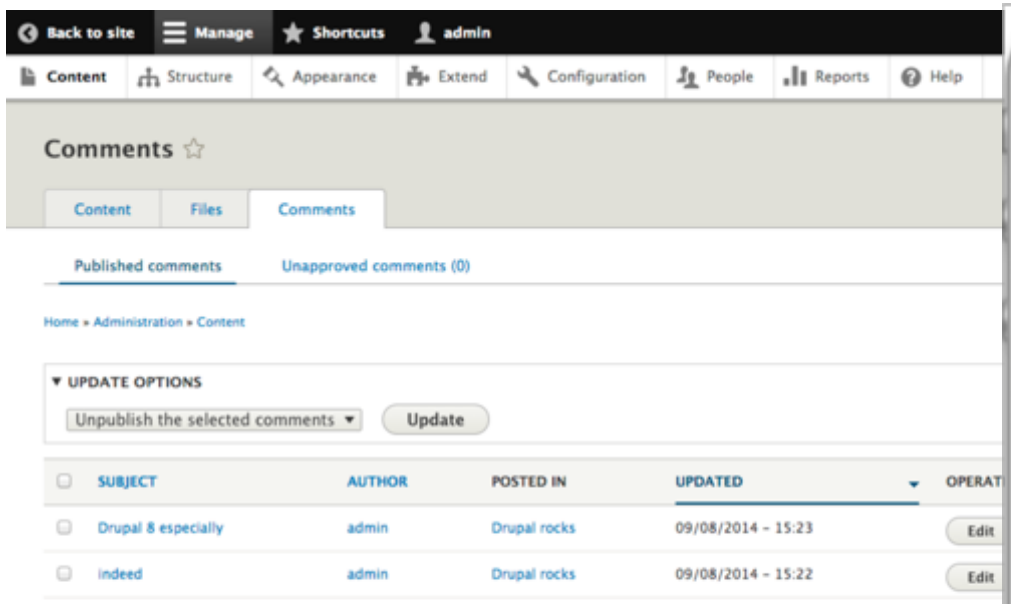


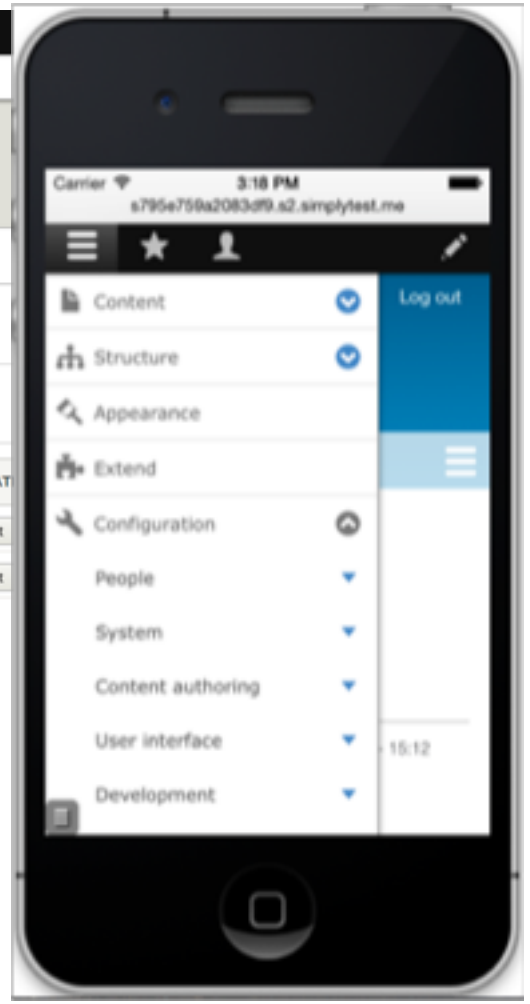
Mobile first

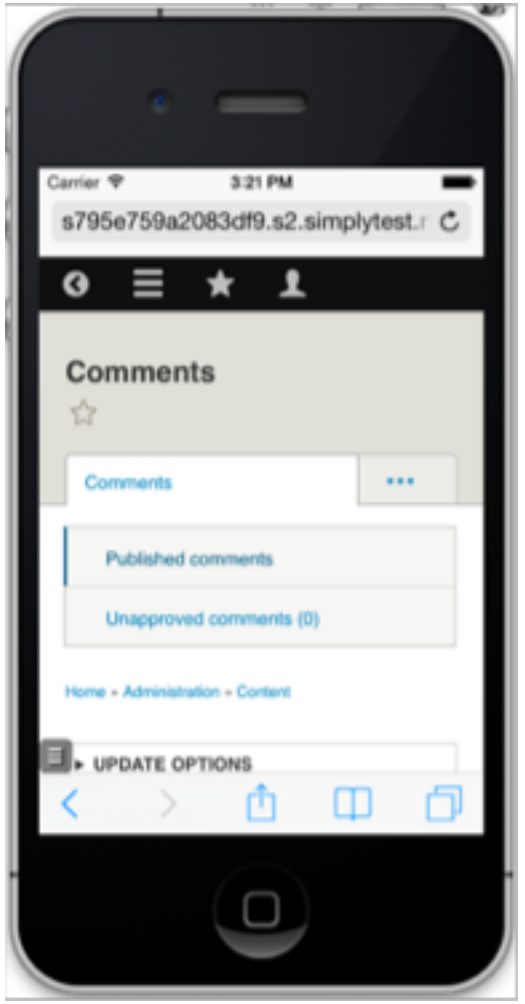




Administration disponible en version mobile





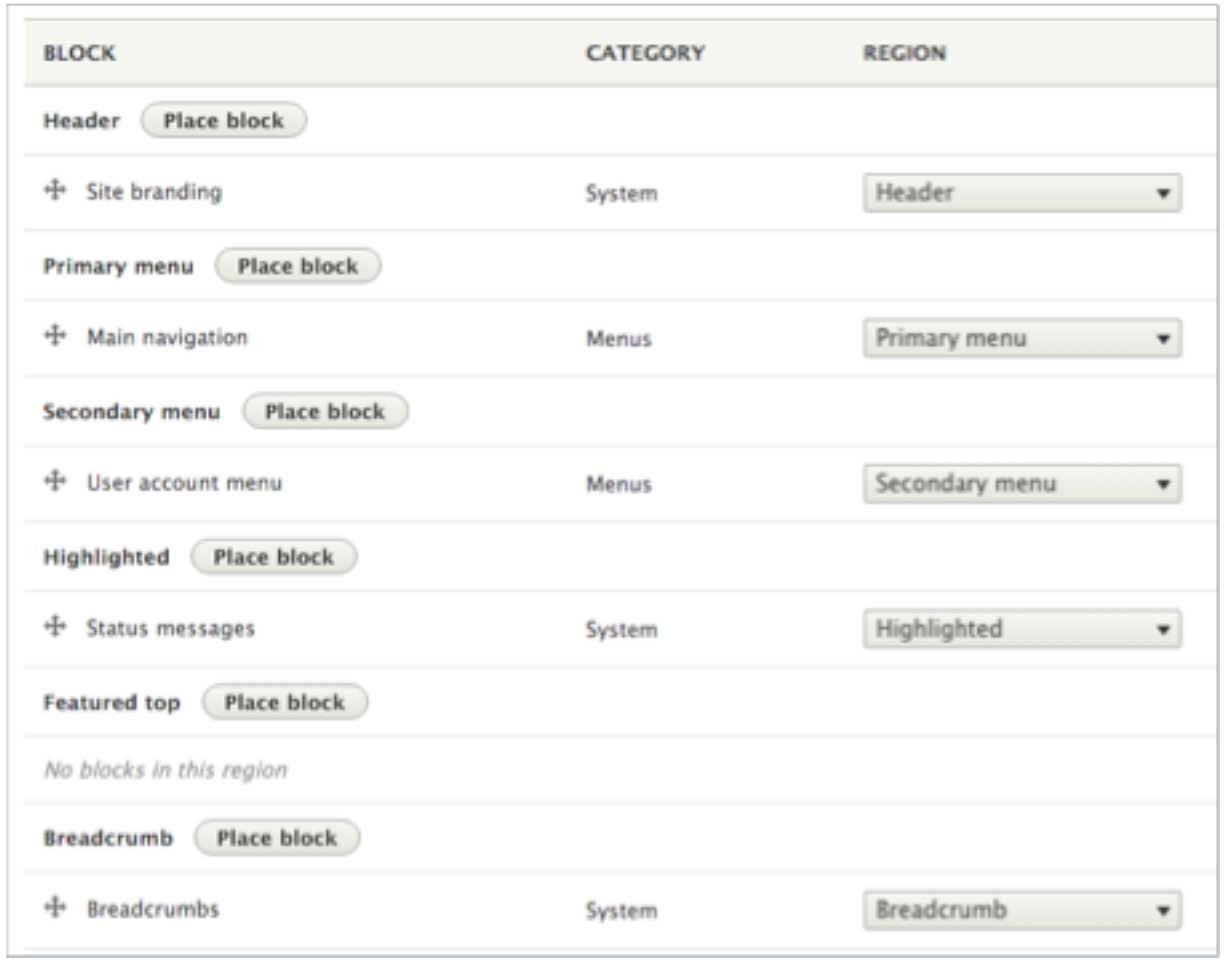


Des fonctionnalités de Drupal 8

L'ÉQUIPETECH

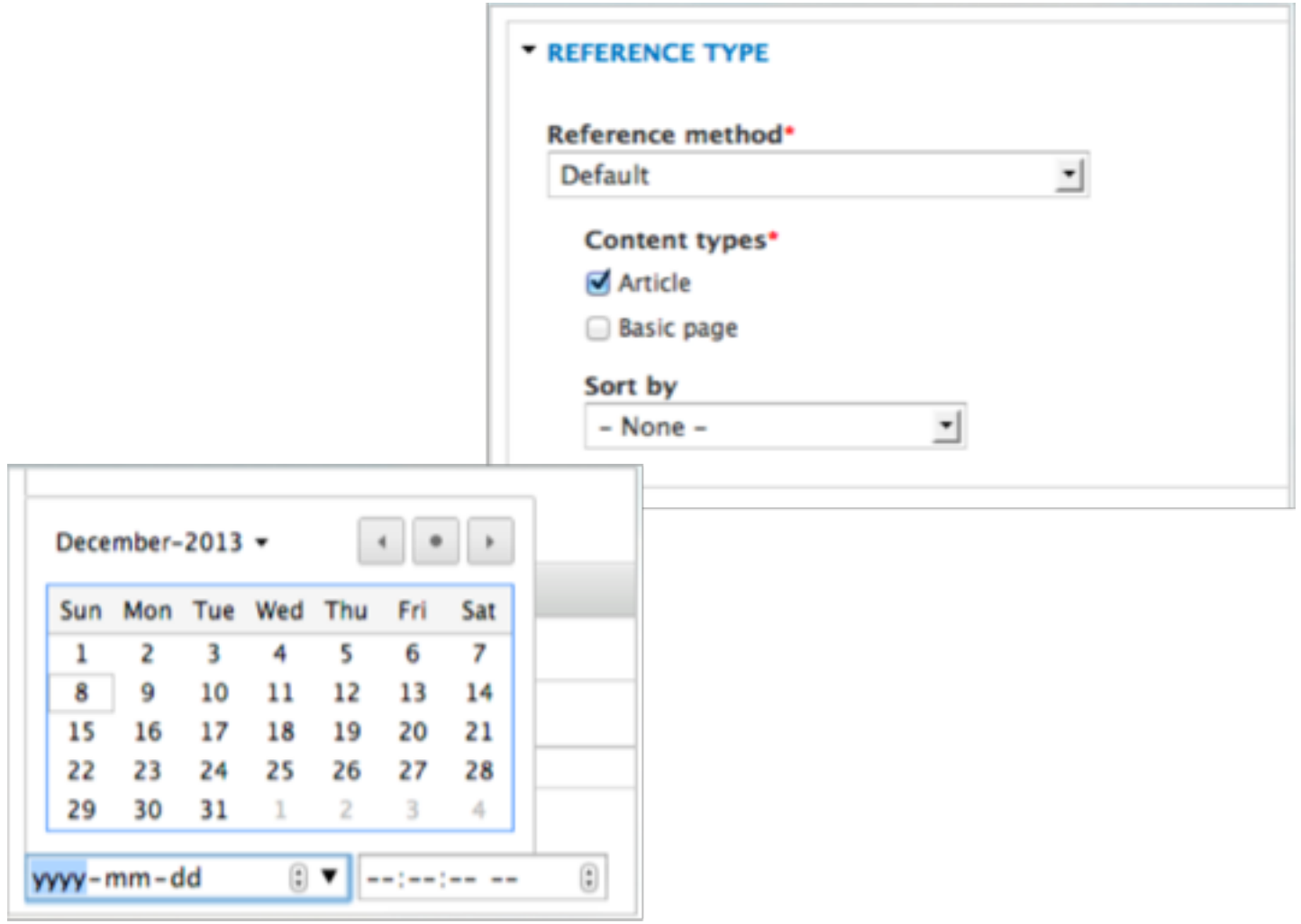
Tout est block

- Et un block peut être placé dans plusieurs régions avec des règles d'affichages suivant:
  - Le rôle de l'utilisateur
  - Les pages affichées
  - La langue
  - Ce que vous voulez



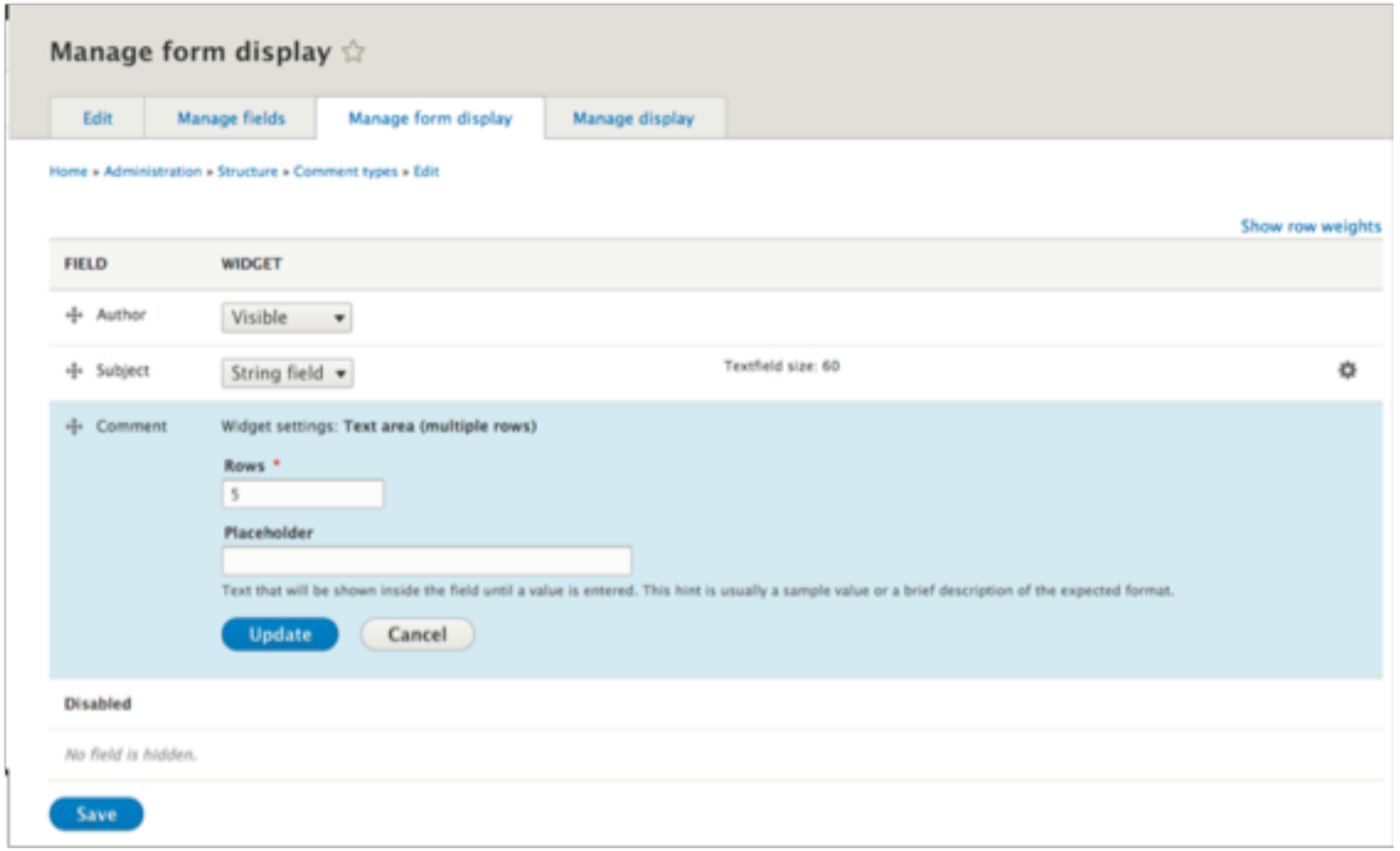
Nouveaux types de champs

- Référence à d'autres entités
- Date / Datetime
- Téléphone
- E-mail
- Champ ou zone de texte avec ou sans wysiwyg



Plusieurs types d'affichage de formulaire possibles

- On le faisait pour l'affichage, on en rêvait pour les formulaires.



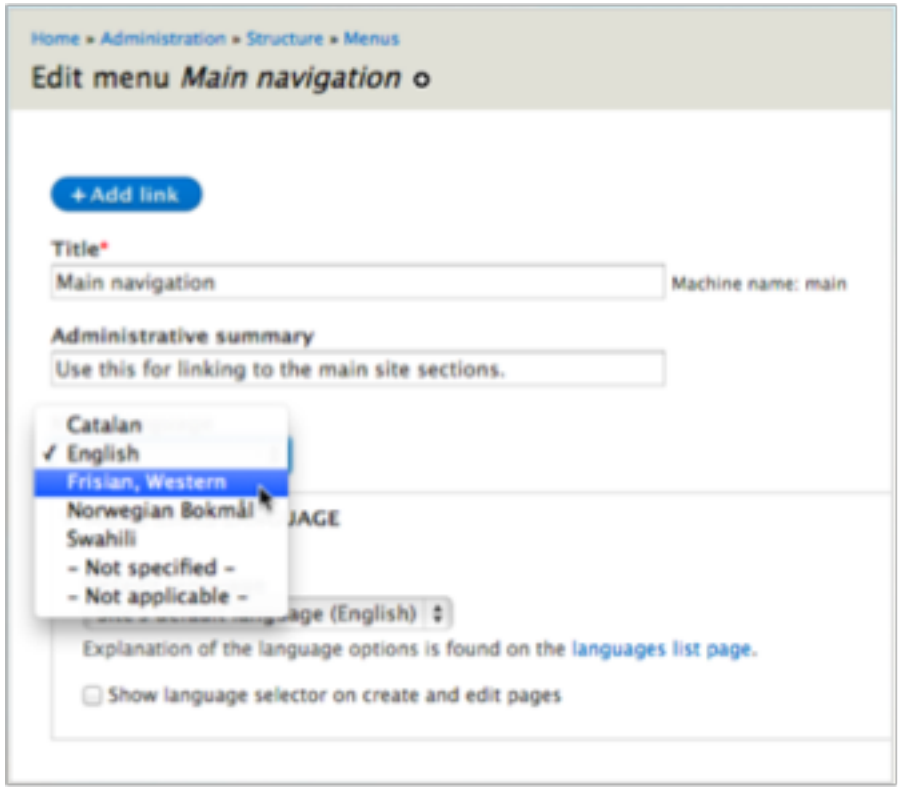
Views intégré au core

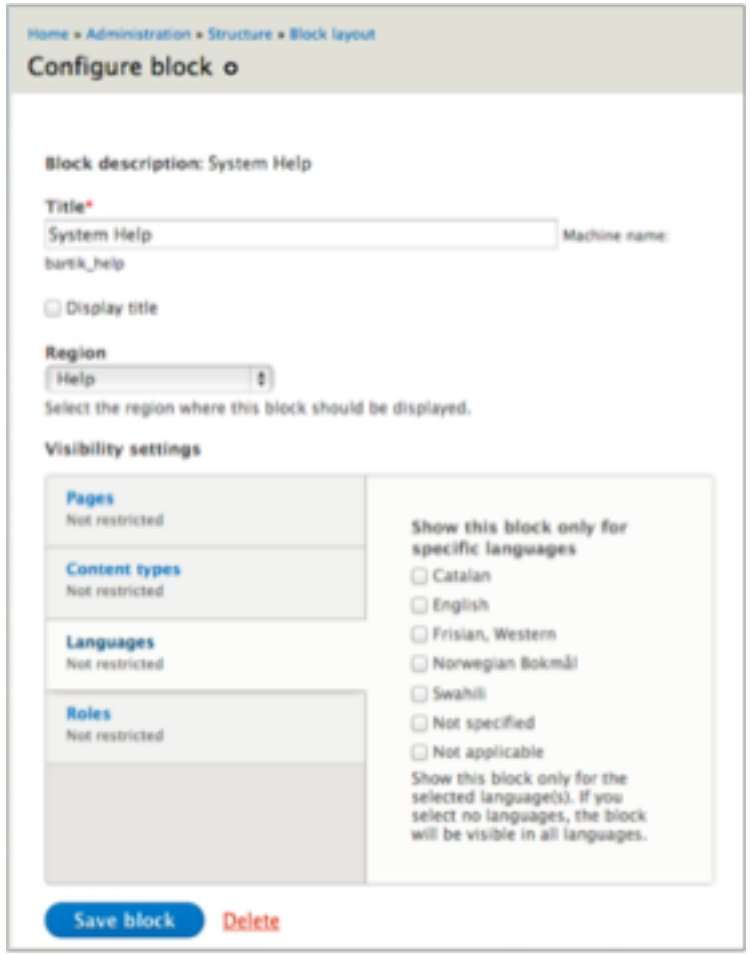
Permet:

- Créer des listes filtrables/triables dans l'administration
- Créer du contenu transverse
- Créer des galeries d'images
- Créer des diaporama
- Afficher une sortie REST

… Avec **0** ligne de code!

Amélioration de la gestion multilingue

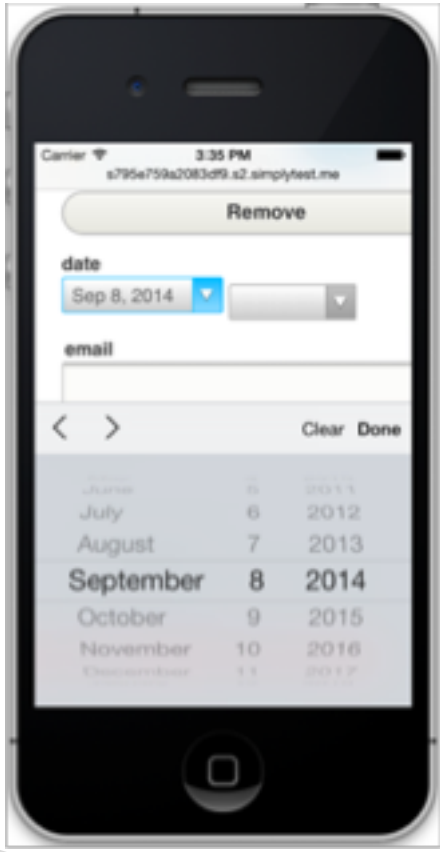


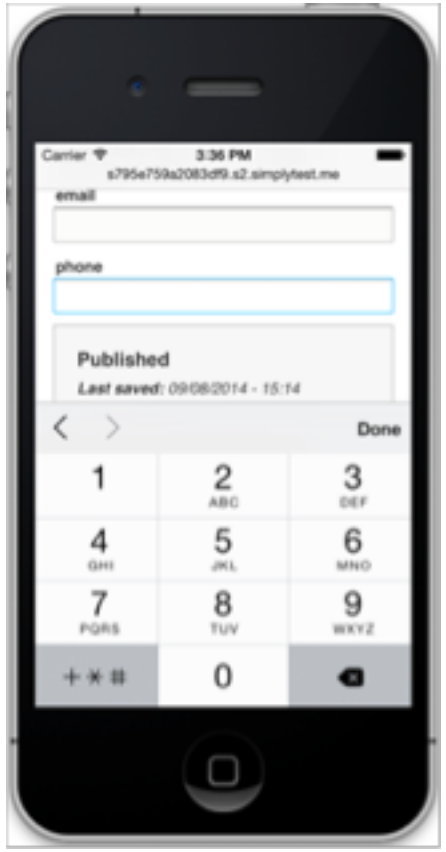


L'ÉQUIPETECH

Des fonctionnalités de Drupal 8

Éléments de formulaire HTML5

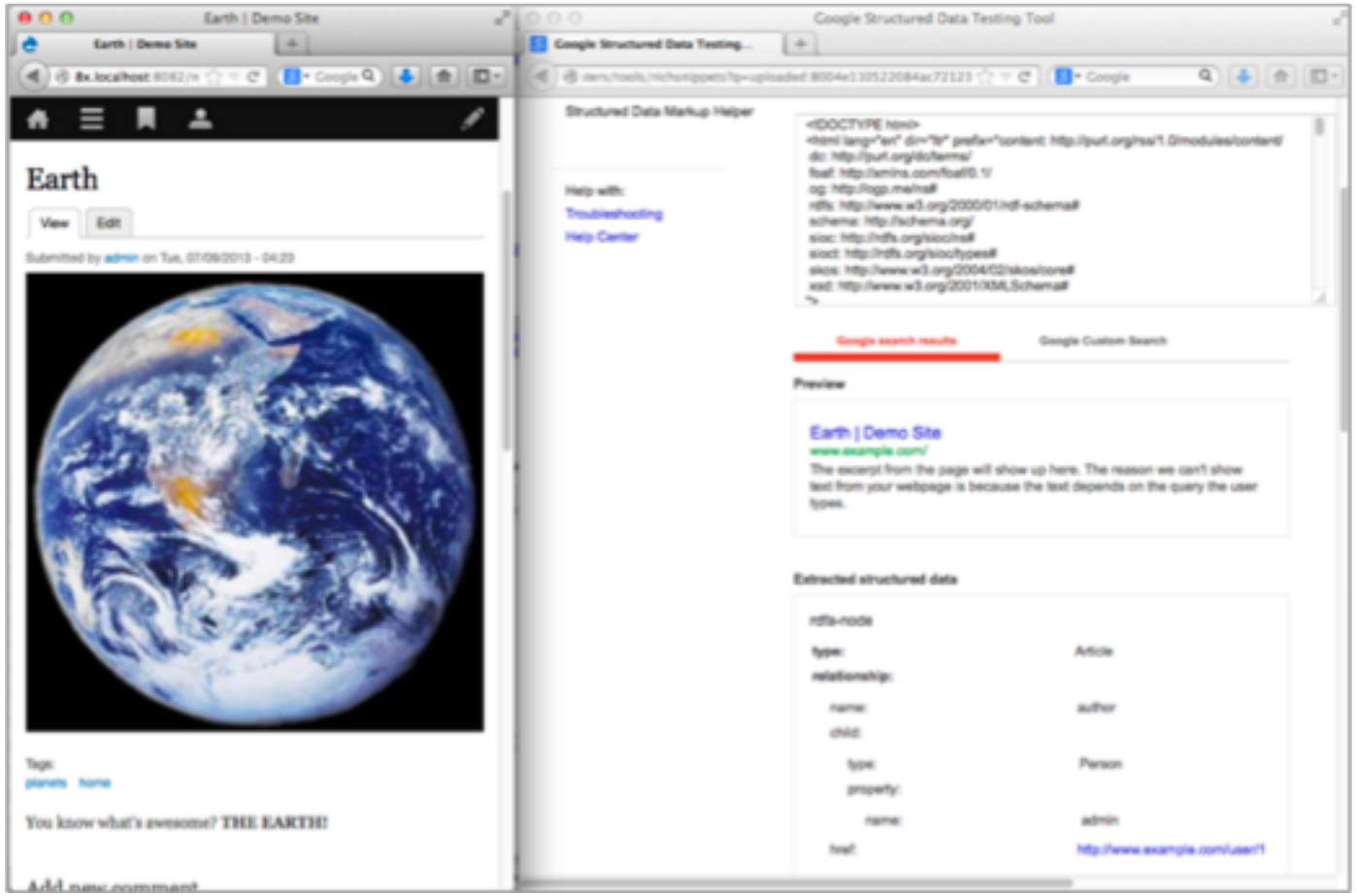




Utilisation de nouvelles librairies Front

- Backbone.js
- Modernizr
- Twig
- Normalize.css

Sortie Schema.org native



Gestionnaire de configuration

Deux outils en ligne de commande

- Drush: outil historique à Drupal pour gérer la vie du site
- Drupal console: Similaire à la console de symfony, d'avantage utilisé pour de l'aide au développement.

Composer

- Intégration de composer pour ajouter/mettre à jour des modules.
- Dépendances et gestionnaire de packages gérés par Drupal.org.

Intégration de composants Symfony 3.x

- HttpFoundation
- HttpKernel
- DependencyInjection
- EventDispatcher
- Routing
- Yaml

Amélioration des caches via les Cache tags

- Le cache est permanent.
- Il faut invalider un cache pour qu'il soit mis à jour.
- Les caches tags permettent d'invalider tous les caches qui référencent ce tag.

~~Not invented here~~
"Proudly Found Elsewhere"

- D’avantage de code moderne et orienté objet (classes, inheritance, interfaces, etc.)
- Embrasse les derniers standards PHP (c.f. PSR-0, namespaces, traits)
- Utilise de nombreuse librairies externes éprouvées: Composer, PHPUnit, Guzzle, Zend Feed Component...

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!

Règle n°2 - 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()`.

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

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 réalisé 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()`.

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';
# }
```

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');
  }
}
```

Règle n°6 – 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°6 – 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°7 – 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°8 – 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
```

Règle n°9 – 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 {}
```

Règle n°9 – 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°10 – 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°10 – 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 – 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°11 – 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°12 – 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°13 – 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°14 – Utilisateurs de test

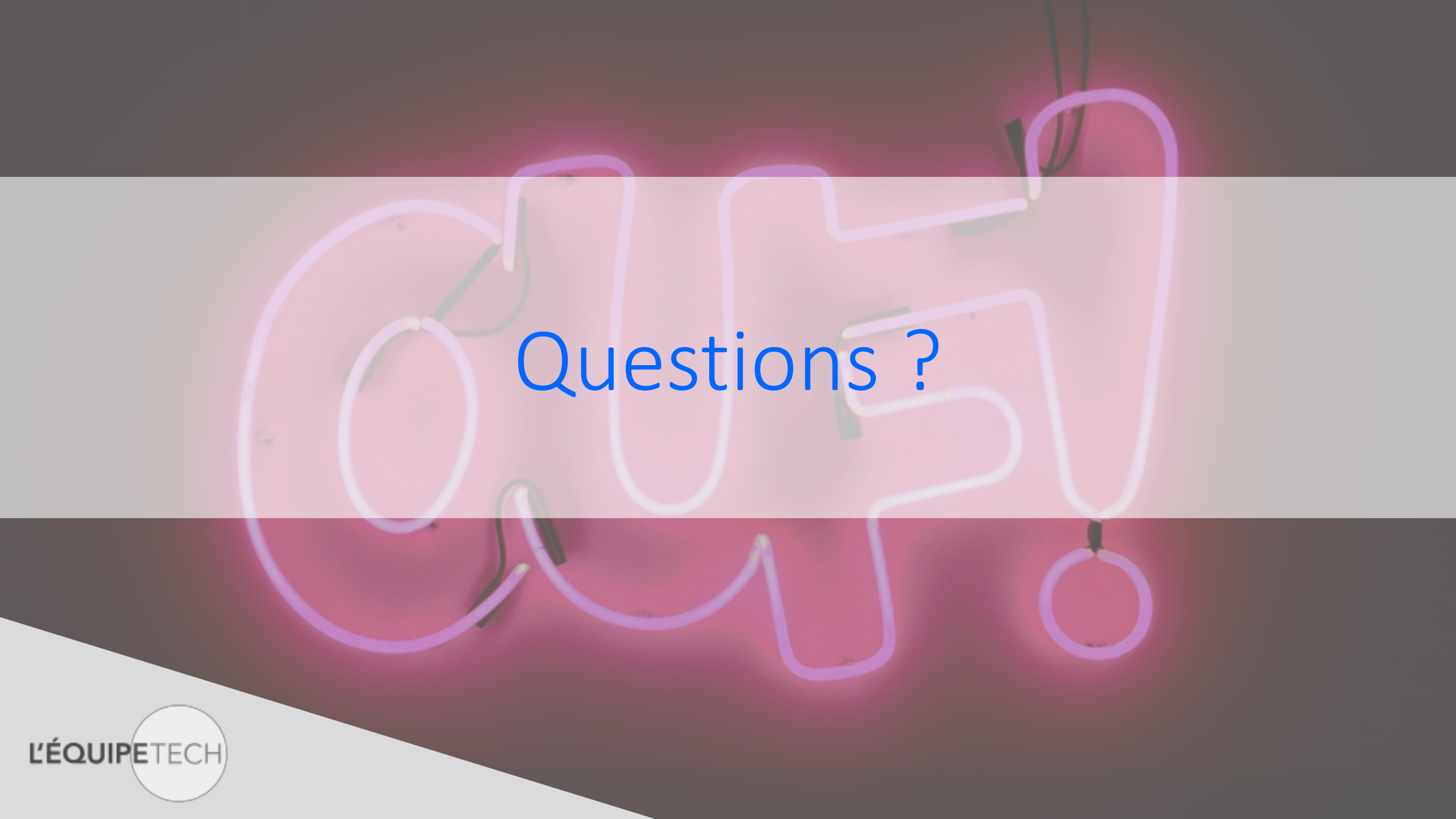
- Éviter au maximum d'être connecté en admin lors du développement.
- Tester les fonctionnalités avec un utilisateur ayant le rôle qui utilisera réellement cette fonctionnalité.
- Tester en anonyme.
- Rappel: le super admin à tous les droits et passe certaines couches de résolution des droits.

Drupal 9

L'ÉQUIPETECH

Drupal 9

- Sortie prévue en 2020.
- Suppression du code déprécié.
- Passage aux composants Symfony 4.x et/ou 5.x.
- Poursuite des initiatives.
- Un Drupal de plus en plus découplé (Objectif découplage total).
- Chemins de migration Drupal 7 et 8 vers Drupal 9 assurée.



Questions ?

L'ÉQUIPETECH