Les services

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
<?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')
    );
  }
```


Exercice: 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.

```php
/**
 * 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.

```php
// 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.

```php
// PriceCalculator.php

namespace Drupal\price\Service;

class PriceCalculator
{
  public function __construct(){}

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

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

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


Exercice : Créer un service

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

Modifier le controlleur de la page /est-il-premium/{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');
  }
}
```


Exercice : 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 premium un utilisateur premium a accéder au site en mode maintenance.

drupal site:maintenance on

drush sset maintenance_mode 1

Décoration de service

Ne remplace pas tout le service d'origine.

Permet de modifier une fonction ou d'en rajouter.

```yaml
services:
  hello.service_override:
    class: Drupal\hello\ServiceHelloOverride
    decorates: hello.service.base
    decoration_priority: 9
    public: false
    arguments: ['@hello.service_override.inner']
```

decorates: service surchargé
decoration_priority: priorité de surcharge
public: accès direct au service
arguments: "*NEW_SERVICE_NAME.inner*". Les arguments du service d'origine doivent aussi être passés.

Décoration de service

```php
/**
 * Class OriginalServiceOverride
 */
class ServiceHelloOverride extends ServiceHello {
  /**
   * Original service object.
   *
   * @var \Drupal\hello\ServiceHello
   */
  protected $helloService;
  /**
   * ServiceHelloOverride constructor.
   *
   */
  public function __construct(ServiceHello $hello_service) {
    $this->$helloService = $hello_service;
    parent::__construct();
  }

}
```


Exercice : Décorer un service

Créer un service PremiumServiceOverride afin de décorer le service PremiumService

Ajouter une fonction isCurrentUserPremium qui retourne un boolean.

Modifier le controlleur de la page /suis-je-premium afin d'utiliser le service via Injection de dépendance

Base
de Données

Schema API

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()

```php
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()

```php
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()

```php
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->udpate();
$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)

```php
$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

Quelques exemples

```php
// Get single value:
$query = \Drupal::database()->select('node_field_data', 'nfd');
$query->addField('nfd', 'nid');
$query->condition('nfd.title', 'Potato');
$query->range(0, 1);
$nid = $query->execute()->fetchField();

// Get single row:
$query = \Drupal::database()->select('node_field_data', 'nfd');
$query->fields('nfd', ['nid', 'title']);
$query->condition('nfd.type', 'vegetable');
$query->range(0, 1);
$vegetable = $query->execute()->fetchAssoc();

// Using db like:
$query = \Drupal::database()->select('node_field_data', 'nfd');
$query->fields('nfd', ['nid', 'title']);
$query->condition('nfd.type', 'vegetable');
$query->condition('nfd.title', $query->escapeLike('ca') . '%', 'LIKE');
$vegetable = $query->execute()->fetchAllKeyed();

// Get several rows with JOIN:
$query = \Drupal::database()->select('node_field_data', 'nfd');
$query->fields('nfd', ['nid', 'title']);
$query->addField('ufd', 'name');
$query->join('users_field_data', 'ufd', 'ufd.uid = nfd.uid');
$query->condition('nfd.type', 'vegetable');
$vegetable = $query->execute()->fetchAllAssoc('nid');
```

```php
// Insert row into database.
$query = \Drupal::database()->insert('flood');
$query->fields([
  'event',
  'identifier'
]);
$query->values([
  'My event',
  'My indentifier'
]);
$query->execute();

// Update row in database.
$query = \Drupal::database()->update('flood');
$query->fields([
  'identifier' => 'My new identifier'
]);
$query->condition('event', 'My event');
$query->execute();

// Upsert (Update or insert if not exists)
$query = \Drupal::database()->upsert('flood');
$query->fields([
  'fid',
  'identifier',
]);
$query->values([
  1,
  'My indentifier for upsert'
]);
$query->key('fid');
$query->execute();

// Delete
$query = \Drupal::database()->delete('flood');
$query->condition('event', 'My event');
$query->execute();
```

Exercice : Création d’une table

- Créer un module « personnes ».
- Utiliser la fonction hook_schema afin de créer une table personnes (nom, prénom, age).
- Utiliser la fonction hook_install afin d’initialiser 3 entrées dans la table.
- Créer une page (Controller + route) /personnes afin d’afficher les 3 entrées dans un tableau (#theme table).
- Ajouter un hook_update pour insérer une autre entrée et vérifier l’ajout sur la page /personnes.

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

```php
// 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
```


Events

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

Exercice : Créer un EventSubscriber

Afficher un message (drupal_set_message) « Kernel events occured » en s’abonnant à KernelEvents::REQUEST.

Déclencher un évènement

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


Créer un évènement

```php
/**
 * 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’evenement

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


Exercice : Créer un évènement

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

Créer un formulaire avec un champ textfield qui déclenchera l'évènement PremiumMessage 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

Utilisé pour le contenu editorial

2 types de champs :

- Configurable Fields (field_*)
- Base Fields (title, author, state, ...)

Révision et traduction possible.

Schéma de base de données générés automatiquement.

Les Nodes sont des Content Entities.

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.

Bundles

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 ».

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.

Entity API

```php
// Make sure that an object is an entity.
if ($object instanceof \Drupal\Core\Entity\EntityInterface) {
}

// Make sure it's a content entity.
if ($entity instanceof \Drupal\Core\Entity\ContentEntityInterface) {
}

// Get the entity type.
$entity->getEntityTypeId();

// Make sure it's a node.
if ($entity instanceof \Drupal\node\NodeInterface) {
}

// Using entityType() works better when the needed entity type is dynamic.
$needed_type = 'node';
if ($entity->getEntityTypeId() == $needed_type) {
}

// Get the ID.
$entity->id();

// Get the bundle.
$entity->bundle();

// Check if the entity is new.
$entity->isNew();

// Get the label of an entity. Replacement for entity_label().
$entity->label().
```


Entity API

```php
// Create a duplicate that can be saved as a new entity.
$duplicate = $entity->createDuplicate();

// Use the procedural wrapper.
$node = entity_create('node', array(
  'title' => 'My node',
  'body' => 'The body content. This just works like this due to the new Entity Field
          API. It will be assigned as the value of the first field item in the
          default language.',
));

// Or you can use the static create() method if you know
// the entity class::
$node = Node::create(array('title' => 'The node title'));

// Use the entity manager.
$node = \Drupal::entityTypeManager()->getStorage('node')->create(array('type' => 'article', 'title' => 'Another node'));

// To save an entity.
$entity->save();

// The following will attempt to insert a new node with the ID 5, this will fail if that node already exists.
$node->nid->value = 5;
$node->enforceIsNew(TRUE);
$node->save();
```


Entity API

```php
// Delete a single entity.
$entity = \Drupal::entityTypeManager()->getStorage('node')->load(1);
$entity->delete();

// Delete multiple entities at once.
\Drupal::entityTypeManager()->getStorage($entity_type)->delete(array($id1 =>
$entity1, $id2 => $entity2));

// Check view access of an entity.
// This defaults to check access for the currently logged in user.
if ($entity->access('view')) {

}

// Check if a given user can delete an entity.
if ($entity->access('delete', $account)) {

}

// Update a field value in a ContentEntity.
$custom_field_value = $Custom_Entity->custom_field->value;
// Perform some kind of data manipulation
$Custom_Entity->custom_field->value = $custom_field_value;
$Custom_Entity->save();
```


Exercice : Création de page utilisateurs premium

- Créer un module exercice_utilisateurs
- Créer 3 utilisateurs à l'installation (hook_install) du module 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*.

Exercice : Créer une page articles premium

- Créer un module exercice_articles
- Créer 5 articles à l'installation (hook_install) du module avec le terme « premium »
- Créer une page articles premium (/articles-premium) qui liste les articles avec l'étiquette premium
- Afficher les articles avec le #theme item_list

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

```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

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


Exercice : 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:config

drupal generate:entity:content --module monmodule

https://hechoendrupal.gitbooks.io/drupal-console/en/commands/generate-entity-config.html

```
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
```


Exercice : 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)

