

# PROJET DEPOT CAMP

Introduction.....	3
Mission 1 .....	4
Tâche 1.1 : Installer le module Megaimporter sur le backoffice Prestashop de la boutique.....	4
Tâche 1.2 : Interfacer le module du backoffice avec l'EDI Grimport .....	4
Mission 2.....	6
Tâche 2.1 : Gérer la phase d'authentification sur le site fournisseur .....	6
Tâche 2.2 : Récupérer les informations constituant un produit .....	7
Tâche 2.3 : Empêcher le crawl si le produit n'appartient pas à la catégorie accessoire .....	8
Tâche 2.4 : Gérer le cache des catégories .....	8
Tâche 2.5 : Gérer les descriptions avec l'API ChatGPT : .....	10
Tâche 2.6 : Supprimer du backoffice les produits qui n'ont pas été crawlé .....	10
Tâche 2.7 : Ajouter le produit au backoffice .....	11
Mission 3.....	12
Tâche 3.1 : Tester le script sur un ou plusieurs produits .....	12
Bilan .....	13

## Introduction :

Depot Camp est un distributeur d'équipements pour camping-cars et vans. Son catalogue actuel lui est fourni par le fournisseur Sunroad Equipment. Toutefois, Depot Camp a constaté que ce catalogue est incomplet, et que Sunroad Equipment ne prévoit pas de lui en fournir une version plus exhaustive. Pour surmonter cette limitation, Depot Camp fait appel à la société idIA Tech, spécialisée dans le web mining. En tant que développeur chez idIA Tech, ma mission consiste à concevoir un script d'importation automatisée du catalogue disponible sur le site de Sunroad Equipment, afin d'alimenter la boutique e-commerce de Depot Camp.

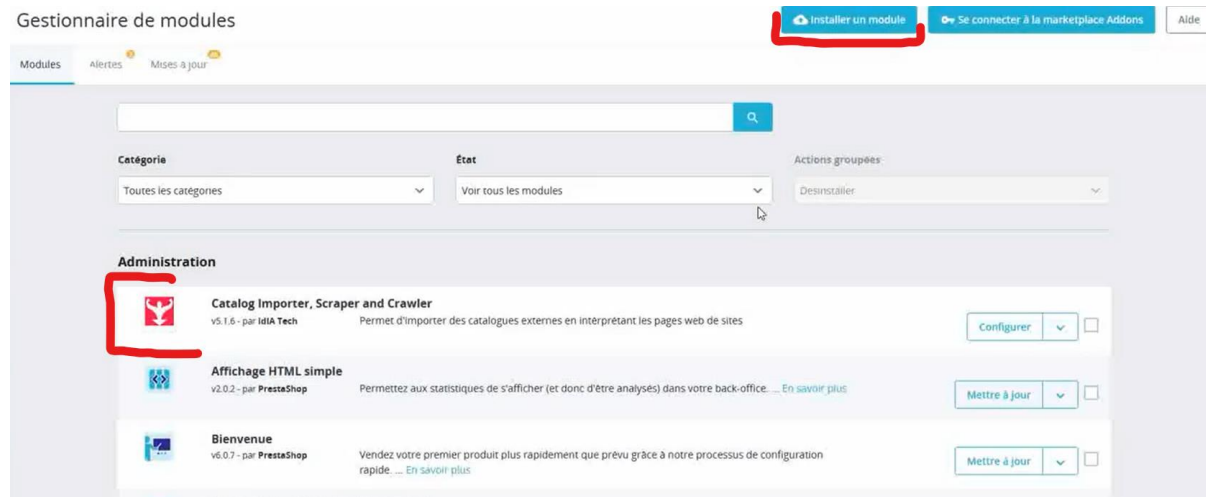
Les logiciels et technologies utilisés pour ce projet sont les suivantes :

- Grimport (EDI)
- Wordpress / Prestashop
- Fiddler
- Langage Grimport (proche de Java)
- HTML/CSS
- XAMPP (serveur local pour les tests)

# Mission 1


## Tâche 1.1 : Installer le module Megaimporter sur le backoffice Prestashop de la boutique

Pour cette tâche il faut commencer par se rendre sur le backoffice Prestashop et installer Megaimporter via le gestionnaire de module :



## Tâche 1.2 : Interfacer le module du backoffice avec l'EDI Grimport

Ensuite il faut ouvrir l'EDI Grimport qui et interfacier le site en rentrant les informations tels que le nom du site, le lien de connexion avec le module du backoffice, ainsi que le lien du backoffice :

 Interfacez votre site

Type :

CMS - PrestaShop, Magento, WordPress

Nom du site :

depotcamp

Lien de connexion :

np.fr/?megaimporter\_communication=1&def=724226114

Lien du backoffice (optionnel) :

https://depotcamp.fr/wp-admin/

Code Cloud idIA Tech (optionnel) :

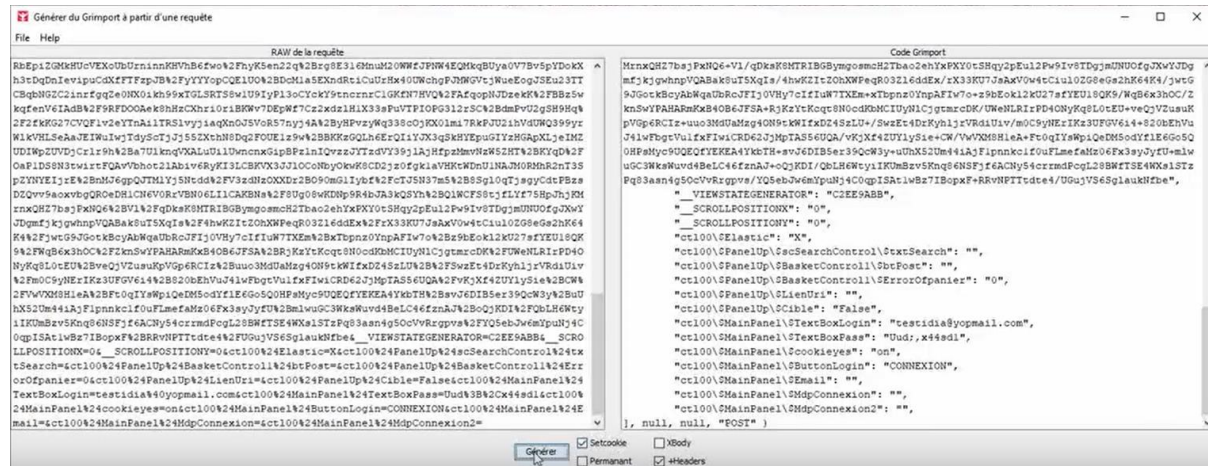
ID employé (optionnel) :

Enregistrer

# Mission 2

## Tâche 2.1 : Gérer la phase d'authentification sur le site fournisseur

Pour la phase d'authentification il est nécessaire de s'authentifier sur le site fournisseur et intercepter la requête brute via Fiddler. Cette requête brute va nous permettre de générer la requête en code Grimpot grâce à un générateur présent sur l'EDI :



Après un peu de nettoyage, voici à quoi ressemble la requête en code Grimpot qui va permettre authentification :

```
// IDENTIFICATION
setCookie("https://www.sunroad-equipment.com/connexion.php")
httpHeader("User-Agent", "Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:132.0) Gecko/20100101 Firefox/132.0", true)
httpHeader("Accept", "text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8", true)
httpHeader("Accept-Language", "fr,fr-FR;q=0.8,en-US;q=0.5,en;q=0.3", true)
httpHeader("Accept-Encoding", "gzip, deflate, br, zstd", true)
httpHeader("Content-Type", "application/x-www-form-urlencoded", true)
httpHeader("Origin", "https://www.sunroad-equipment.com", true)
httpHeader("Connection", "keep-alive", true)
httpHeader("Referer", "https://www.sunroad-equipment.com/connexion.php", true)
httpHeader("Upgrade-Insecure-Requests", "1", true)
httpHeader("Sec-Fetch-Dest", "document", true)
httpHeader("Sec-Fetch-Mode", "navigate", true)
httpHeader("Sec-Fetch-Site", "same-origin", true)
httpHeader("Sec-Fetch-User", "?1", true)
httpHeader("Priority", "u=0, i", true)
post("https://www.sunroad-equipment.com/connexion.php", [
  "login": " ",
  "mdp": " ",
  "user_locale": "auto"
])
```

## Tâche 2.2 : Récupérer les informations constituant un produit

Je commence par analyser le code HTML d'une fiche produit sur le site du fournisseur, afin d'identifier les éléments pertinents qui décrivent un produit, tels que le nom, le prix ou encore le numéro de référence. Cette étape permet de repérer les balises HTML et les classes CSS associées à ces informations, ce qui est essentiel pour leur extraction automatisée.

Ensuite, dans le script *FORPAGE* qui est exécuté sur chaque page produit du site du fournisseur j'utilise des sélecteurs CSS ciblés pour extraire précisément ces données. Ces sélecteurs me permettent de naviguer efficacement dans la structure du DOM (Document Object Model) et de récupérer uniquement les informations utiles pour enrichir le catalogue du distributeur :

```
nom = cleanSelect(".product_base_info h1")
prix = select(".prix_visible > tbody > tr > td:nth-child(3)");
prix = removeElement(removeElement(prix, "br"), "span")
prix = htmlToPrice(trim(standardizeText(prix)))
reference = cleanRegex(/(?s)id\s*=\s*"quantity_([^\>"]+)"$/)
numEAN13 = removeElement(select(".pb_tab_content.logistique > .row > .col-lg-8 > div"), "strong")
numEAN13 = trim(stripTags(standardizeText(numEAN13)))
titrePage = nom
stock_info = cleanSelect(".statut_stock_product")
conditionnementValuesTr = get(selectAll(".tableau_donnees_logistique tbody tr"), 0)
conditionnementValuesTr = "<table><tr> $conditionnementValuesTr </tr></table>"
conditionnementValues = cleanSelectAll("td", conditionnementValuesTr)
poids = number(get(conditionnementValues, 2))
longueur = number(get(conditionnementValues, 3))
largeur = number(get(conditionnementValues, 4))
hauteur = number(get(conditionnementValues, 5))
```

Nous ajoutons une condition afin de nous assurer que le script ne poursuit son exécution que si la page analysée est bien une page d'un produit. Cette vérification est cruciale pour éviter de traiter des pages non pertinentes, comme les pages de catégorie, d'accueil ou les résultats de recherche. Généralement, cette condition repose sur la présence d'éléments spécifiques au modèle de page produit, tels qu'un bloc contenant le nom du produit ou un identifiant HTML caractéristique. Cela garantit que les opérations de scraping sont réalisées uniquement sur les pages contenant des données exploitables :

```
// IGNORER LA PAGE SI CE N'EST PAS UNE PAGE PRODUIT
isProductPage = nom && prix && reference
if(!isProductPage) return
```

## Tâche 2.3 : Empêcher le crawl si le produit n'appartient pas à la catégorie accessoire

Je récupère notamment le fil d'Ariane présent sur chaque page produit, car il me permet d'identifier les catégories auxquelles appartient l'article. Cette information est essentielle pour organiser correctement les produits dans la boutique en ligne de Depot Camp. Afin de limiter l'import uniquement aux produits réellement pertinents, j'ajoute une condition dans le script : je ne poursuis l'extraction que si le produit appartient à la catégorie "Nos accessoires". Pour cela, je vérifie la structure du fil d'Ariane et je m'assure que cette catégorie y figure, ce qui me permet d'éviter de traiter des produits issus d'autres sections du site qui ne nous concernent pas :

```
// IGNORER LA PAGE PRODUIT SI LE PRODUIT N'APPARTIENT PAS A LA CATEGORIE "Nos Accessoires"
categories = cleanSelectAll(".breadcrumbs a:not(:first-child)")
if(!equals(categories[0], "Nos accessoires")) return
```

## Tâche 2.4 : Gérer le cache des catégories

Je parcours les éléments du fil d'Ariane à l'aide d'une boucle, afin d'identifier chaque catégorie associée au produit. Pour optimiser le traitement et éviter les doublons, je vérifie si chaque catégorie est déjà présente dans le cache. Si ce n'est pas le cas, je l'ajoute au cache et je le mets à jour en conséquence. Cette logique me permet de construire dynamiquement une liste de catégories uniques tout en minimisant les opérations redondantes :

```
// ADD CATEGORY
ids_cat_asso = []
id_category = null;
if(!id_product_exist)
{
    id_parent = idCategoryRacine
    majCategoryCache = false;
    for(category in categories)
    {
        id_category = getCategoryIn(category, id_parent)

        if(!id_category)
        {
            id_category = functionNow("add_category_product",[category, id_parent])
            majCategoryCache = true
        }

        ids_cat_asso.add(id_category)
        id_parent = id_category;
    }

    if(majCategoryCache) updateCategories()
}
```





## Tâche 2.5 : Gérer les descriptions avec l'API ChatGPT :

Je récupère la description longue du produit directement depuis la page du fournisseur. À partir de cette description détaillée, j'utilise l'API d'OpenAI (ChatGPT) pour générer automatiquement une description courte, plus concise et adaptée à la fiche produit de la boutique. Je configure la requête avec des instructions précises afin que la description générée soit claire, attrayante et conforme au ton et au style souhaités par Depot Camp :

```
// GPT QUESTION
if(descriptionSource)
{
    question = "Créer une description courte de produit en français. Longueur maximale : 300 caractères. Inclure les informations importantes :
descriptionSource. Utiliser un langage clair, concis et professionnel. Éviter les répétitions et répondre uniquement avec le texte final "
    gptDescriptionCourteVariation = gptQuestion(question, chatGPT_api_key, "gpt-4o")

    if(gptDescriptionCourteVariation) descriptionCourte = gptDescriptionCourteVariation
}
```

## Tâche 2.6 : Supprimer du backoffice les produits qui n'ont pas été crawlé

Dans le script initial, je déclare un tableau destiné à stocker tous les produits extraits au cours du crawling, via le script *FORPAGE*. Chaque produit analysé y est ajouté progressivement. Une fois le crawling terminé, dans le script final, j'exécute une fonction PHP qui supprime l'ensemble des produits actuellement présents dans le back-office du distributeur, à l'exception de ceux spécifiquement listés dans les paramètres d'exclusion. Cette étape permet de synchroniser proprement la base produits en ne conservant que les articles encore valides ou explicitement protégés :

```
// TABLEAU DES REFERENCES A EXCLURE DE LA SUPPRESSION DES PRODUITS SUR LE BACKOFFICE A LA FIN DU CRAWL
referencesExclues = []
setGlobal("referencesExclues")

//AJOUT DANS LES REFERENCES A EXCLURE DE LA SUPPRESSION A LA FIN DU CRAWL
referencesExclues.add(reference)

// SUPPRESSION SUR LE BACKOFFICE DES PRODUITS DU FOURNISSEUR NON CRAWLE
phpBigCommandsByLine(functionNow("delete_all_products_test", [referencesExclues, [fournisseurAttributeID]]))
```

## Tâche 2.7 : Ajouter le produit au backoffice

Pour ajouter un produit, j'utilise simplement la fonction PHP

`update_or_add_product`, à laquelle je transmets les différentes caractéristiques du produit récupérées à l'aide des sélecteurs CSS (nom, prix, référence, description, etc.).

Il est important de ne pas oublier d'importer également les informations de stock, en appelant la fonction PHP `update_stock`, afin d'assurer que la disponibilité affichée sur la boutique soit à jour et cohérente avec celle du fournisseur :

```
// ADD/UPDATE PRODUCT
name = id_product_exist ? null : nom
short_description = useDescriptionCourte ? descriptionCourte : null
long_description = useDescriptionLongue ? descriptionLongue : null
function("update_or_add_product",[name, prix, reference, short_description, long_description, null, poids, hauteur, largeur, longueur])
if(!id_product_exist) function("associate_categories",[ids_cat_asso])
is_stock_dispo = equals(stock_info, "Disponible")
stock_quantite = is_stock_dispo ? 1000 : 0
stock_statut = is_stock_dispo ? "instock" : "outofstock"
function("update_stock",[stock_quantite, stock_statut])
```

## Mission 3

### Tâche 3.1 : Tester le script sur un ou plusieurs produits

Le script étant finalisé, j'ai procédé à une série de tests en important manuellement quelques produits dans le back-office. Cela m'a permis de vérifier que l'extraction des données, leur formatage et l'importation fonctionnaient correctement de bout en bout. Ces tests ont également servi à valider le bon comportement du filtrage par catégorie et la génération des descriptions courtes via l'API d'OpenAI.

### Tâche 3.2 : Importer un produit test et informer le client pour qu'il valide le script

Il ne reste plus qu'à envoyer un mail au client avec le lien des produit test importé sur son back office pour qu'il valide ou non le script

## Bilan

Le développement du script d'importation des produits a été mené à bien et a permis de mettre en place un flux automatisé entre le site du fournisseur Sunroad Equipment et la boutique en ligne de Depot Camp. Le processus a commencé par l'analyse du code HTML des pages produits, afin de repérer et extraire les informations pertinentes telles que le nom, le prix, la référence et la description du produit. À partir de ces données, j'ai utilisé l'API OpenAI pour générer des descriptions courtes et adaptées, optimisant ainsi la présentation des produits sur la boutique.

Après avoir finalisé le script d'importation, j'ai procédé à des tests pratiques en important manuellement quelques produits dans le back-office de Depot Camp. Ces tests ont permis de vérifier que les données étaient correctement extraites, formatées et intégrées, et que les stocks étaient bien mis à jour à l'aide de la fonction PHP `update_stock`.

Le script a été conçu pour gérer efficacement l'ajout de nouveaux produits tout en excluant ceux qui ne répondent pas à des critères spécifiques, grâce à des filtres appliqués sur le fil d'Ariane et les catégories. De plus, une logique de mise à jour du cache a été intégrée afin d'éviter les doublons et d'optimiser les performances du processus d'importation.

En somme, l'intégration est désormais opérationnelle, et le système est capable de maintenir le catalogue de produits à jour, tout en assurant la bonne gestion des stocks. Il ne reste alors plus que le retour de la part du client.