Plan de formation Magento - Scénario pratique : création d'un site de produits via une API (Amazon)

Scénario :
Vous êtes une agence en charge de développer un site e-commerce multimarques qui regroupe et héberge des **produits récupérés via l'API Amazon** (Amazon Product Advertising API). Les produits récupérés seront affichés avec leurs détails, et les utilisateurs pourront les ajouter à leur panier pour les acheter directement via votre plateforme.

Objectifs pédagogiques :

1. Comprendre et maîtriser les bases de Magento (modules, thèmes, etc.).
2. Développer des modules personnalisés pour importer et gérer les produits d'une API externe.
3. Intégrer ces produits au catalogue Magento.
4. Personnaliser le frontend pour afficher des produits récupérés par l'API.
5. Gérer les paniers et les commandes.
6. Optimiser le site (performance, cache, sécurité, API).

Plan en étapes détaillées :

Jour 1 : Introduction et mise en place du projet

La première journée de cette formation a pour objectif de poser les bases essentielles pour le développement avec Magento. Nous débuterons par une exploration des concepts fondamentaux de Magento, son architecture robuste et modulaire, ainsi que son écosystème riche. Ces notions théoriques permettront aux participants de mieux comprendre la philosophie de cette plateforme, et de se familiariser avec la manière dont elle structure et manipule ses fonctionnalités.

Ensuite, nous nous concentrerons sur l'installation et la configuration de l'environnement de développement. Cela inclut l'utilisation d'outils pratiques comme Docker ou DDEV pour simplifier et automatiser la préparation de notre infrastructure. Nous installerons une instance Magento fonctionnelle, prête à être personnalisée tout au long de cette formation.

Enfin, la journée s'achèvera par une introduction à l'**API Amazon Product Advertising**, que nous utiliserons tout au long de la formation pour alimenter notre site en produits externes. Nous explorerons rapidement son fonctionnement, son authentification et ses principales fonctionnalités pour établir les bases de son intégration future avec Magento.

À la fin de cette première journée, les développeurs auront installé et configuré un environnement fonctionnel, construit une base solide pour démarrer le développement de notre projet, et seront prêts à s'attaquer aux modules personnalisés dès le jour 2.

- **1.1. Présentation et préparation de Magento**

Objectifs de cette section :

- Comprendre les concepts fondamentaux et l'architecture de Magento.
- Apprendre la structure modulaire de Magento et ses fichiers clefs.
- Installer Magento grâce à **Composer**, puis mettre en place un environnement local optimisé avec des technologies modernes comme **Docker** ou **DDEV**.

A - Découvrir Magento et son écosystème

Magento est l'une des plateformes e-commerce les plus utilisées, en raison de sa flexibilité et de sa robustesse. Lorsqu'on aborde Magento, il est essentiel de comprendre que cette solution repose sur une **architecture très modulaire** basée sur les principes suivants :

- **Répartition en modules découplés** pour une meilleure extensibilité.
- **Respect du pattern MVC** (Model-View-Controller), combiné avec une architecture orientée service.
- Une structure qui adopte le plus possible des **standards modernes de PHP** (PSR-1/PSR-2, DI, Composer, etc.).

Voici quelques concepts essentiels pour poser les bases avant l'installation et la manipulation de Magento :

---

1. La structure de Magento :

Magento utilise une structure de dossiers bien définie, où chaque composant a un rôle précis.

1. **Organisation des fichiers principaux :**
    - `app/` : Cœur de votre projet Magento.
        - `code/` : Contient les modules Magento personnalisés.
        - `design/` : Contient les thèmes pour le Frontend et Backend.
        - `etc/` : Configuration des modules.
    - `pub/` : Dossier public (accessible via le navigateur). Fichiers frontaux.
    - `vendor/` : Système de packages géré avec Composer (Magento Core, outils tiers).
    - `generated/` : Code généré automatiquement (proxies, fichiers de cache PHP).

2. **Les modules :** Magento fonctionne grâce à une **architecture modulaire**, où chaque fonction (gestion de catalogues, clients, commandes) est encapsulée dans un module :
    - Les modules déclarent leurs fonctionnalités et leurs relations via un fichier `module.xml`.
    - Les **fichiers XML** jouent un rôle clé dans la configuration :
        - `layout.xml` : Gère les positions des blocs dans le thème.
        - `routes.xml` : Définit les chemins des URLs.

- `di.xml` : Définit l'*injection de dépendances* (dependency injection).
- `db_schema.xml` : Déclare les tables (API déclarative pour les bases de données).

3. **Service Contracts (API) :**

   Qu’est-ce qu’un Service Contract dans Magento ?

Les Service Contracts dans Magento 2 forment une **API interne** qui permet aux différents composants de communiquer entre eux de manière standardisée, stable et décorrélée.
Il s’agit d’une **approche contractuelle**, où les développeurs décrivent les fonctionnalités disponibles via des **interfaces (contrats)**, implémentées directement par les modules ou services de Magento.

Cette notion repose principalement sur trois concepts fondamentaux :

1. **Les Interfaces PHP** : Définissent les "contrats" ou comportements prévus d’un module/service.
2. **Les Data-Objects** : Objets simples pour échanger des données entre les couches (ex : `ProductInterface` pour représenter un produit).
3. **La transparence** : Une séparation totale entre l’interface (contrat) et l'implémentation réelle.

Pourquoi les Service Contracts sont-ils importants ?

1. Découplage & extensibilité :

- L’utilisation d’un contrat garantit que le reste du système ne dépend pas de l’implémentation d’un module, mais de son interface.
  Exemple : Si un module pour la gestion des produits est remplacé ou étendu, le reste du projet restant interagit toujours avec la même interface (`ProductRepositoryInterface`), sans se soucier de son implémentation en backend.

2. Stabilité des fonctionnalités API :

- Les interfaces des Services Contracts sont déclarées *publiques* et considérées stables entre les versions Magento. Cela garantit qu’elles resteront utilisables quel que soit l'évolution sous-jacente.

3. Compatibilité avec les APIs publiques :

- Les Services Contracts servent également à exposer des fonctionnalités aux **API REST et GraphQL** en utilisant les interfaces standard comme base.

4. Avantages pour les tests unitaires :

- Grâce au découplage entre l'interface et l'implémentation, les développeurs peuvent facilement simuler certains comportements avec des **mocks** (fakes ou objets de test) pour tester leurs modules sans dépendre des implémentations réelles.

Les principaux éléments des Service Contracts

Les Services Contracts dans Magento reposent sur deux concepts clés : les **Repositories** et les **Data-Objects** (DTO, Data Transfer Objects).

---

1. Les Repositories (Repositories Patterns)

Les repositories permettent d'accéder aux **entités métier Magento** (produits, clients, commandes, etc.) en suivant un pattern d'accès standardisé. Ils agissent comme un **moyen unique d'interfacer une entité**, plutôt que d'accéder directement à un modèle.

Exemple classique :
Considérons la gestion d'un produit dans Magento. Sans Service Contracts, on pourrait interagir directement avec les modèles :

```
$product = $objectManager->create(\Magento\Catalog\Model\Product::class);
$product->load(1);
```

Avec les Service Contracts, on passe par un repository dédié, défini par une interface (`ProductRepositoryInterface`), qui agit comme couche d'abstraction :

```
use Magento\Catalog\Api\ProductRepositoryInterface;

// Exemple avec DI
/** @var ProductRepositoryInterface $productRepository */
$product = $productRepository->getById(1);
```

Interface d'un Repository (exemple pour les produits)

Dans le module `Magento\Catalog\Api`, on trouve une interface définissant le comportement du repository produit :

```
namespace Magento\Catalog\Api;

use Magento\Catalog\Api\Data\ProductInterface;

interface ProductRepositoryInterface
{
    /**
     * Retourne un produit par son identifiant.
     *
     * @param int $id
     * @return \Magento\Catalog\Api\Data\ProductInterface
     */
    public function getById($id);
```

```
    /**
     * Enregistre un produit.
     *
     * @param \Magento\Catalog\Api\Data\ProductInterface $product
     * @return \Magento\Catalog\Api\Data\ProductInterface
     */
    public function save(ProductInterface $product);

    /**
     * Supprime un produit.
     *
     * @param \Magento\Catalog\Api\Data\ProductInterface $product
     * @return bool
     */
    public function delete(ProductInterface $product);
}
```

Comment cela fonctionne ?

Magento repose sur son système d'**Injection de dépendances (DI)** pour associer cette interface (`ProductRepositoryInterface`) à une implémentation réelle :

```
<type name="Magento\Catalog\Api\ProductRepositoryInterface">
    <arguments>
        <argument name="class" xsi:type="object">Magento\Catalog\Model\
ProductRepository</argument>
    </arguments>
</type>
```

Ainsi, lorsqu'un repository est injecté dans une classe, Magento sait automatiquement quelle implémentation utiliser.

---

2. Les Data-Objects ou DTOs (Data Transfer Objects)

Dans Magento, les **Data-Objects** sont utilisés pour transporter les données entre les couches (souvent entre le repository et le reste du code). Contrairement aux modèles Magento classiques, les Data-Objects sont des **objets POPO (Plain Old PHP Objects)**, sans logique métier ni dépendance à la base de données.

Exemple : Interface pour représenter un produit

Un produit est souvent représenté par une interface `ProductInterface` (fournie dans `Magento\Catalog\Api\Data\ProductInterface`) :

```
namespace Magento\Catalog\Api\Data;

interface ProductInterface
{
    const ID = 'entity_id';
    const NAME = 'name';
    const SKU = 'sku';

    /**
     * Identifiant du produit.
```

```
     *
     * @return int|null
     */
    public function getId();
    public function setId($id);

    /**
     * Nom du produit.
     *
     * @return string
     */
    public function getName();
    public function setName($name);

    /**
     * SKU du produit.
     *
     * @return string
     */
    public function getSku();
    public function setSku($sku);
}
```

Le repository retourne toujours ce type d'objet via son API. Exemple pratique :

```
$product = $productRepository->getById(1);
echo $product->getName(); // Récupère le nom
$product->setName("Nouveau nom"); // Modifie le nom
$productRepository->save($product); // Sauvegarde les changements
```

Résumé : Points clés des Service Contracts

1. Les **Interfaces** garantissent un accès standard aux entités et modules Magento.
2. Les **Data-Objects** (DTO) isolent la logique métier des données brutes.
3. Les Repositories fournissent des méthodes claires pour accéder aux entités *sans dépendance directe* à la base de données.
4. Ils permettent un **découplage total**, utile pour la maintenabilité, les tests, et les extensions futures.

2. Fonctionnalités natives de Magento :

Magento est particulièrement puissant grâce à certaines fonctionnalités clés disponibles nativement :

- **Multiboutique** pour gérer plusieurs instances (différentes langues, devises...).
- Stockage des données via **EAV (Entity-Attribute-Value)** pour une grande flexibilité (exemple : ajout d'attributs personnalisés pour les produits ou clients).
- Gestion avancée des commandes et paiements.
- Système de cache intégré (Full Page Cache, Redis).
- APIs REST et GraphQL pour exploiter Magento en tant que backend (*headless*).

B - Installation de Magento via Composer

Magento repose sur l'utilisation de **Composer**, qui simplifie la gestion des dépendances et des librairies tierces. Cette méthode est à privilégier, car elle permet d'automatiser les mises à jour et de maintenir une bonne organisation du projet.

Étapes de l'installation :

1. **Accéder aux clés API de Magento Marketplace :**
    - Les développeurs devront créer un compte gratuit sur Magento Marketplace.
    - Aller dans "My Profile" -> "Access Keys", puis générer une clé *public* et une clé *private*.
    - Ajouter ces clés globalement pour Composer :

```
composer config --global http-basic.repo.magento.com <public-key> <private-key>
```

Installer Magento :

- Créez un nouveau projet :

```
composer create-project --repository-url=https://repo.magento.com/ magento/project-community-edition magento2
```

Effectuez les configurations nécessaires lors de l'installation (base de données MySQL, URL de base, etc.).

Installation complète :

- Finalisez la configuration grâce à la CLI Magento :

```
php bin/magento setup:install \
--base-url=http://magento.test/ \
--db-host=127.0.0.1 \
--db-name=magento \
--db-user=root \
--db-password=yourpassword \
--admin-firstname=Admin \
--admin-lastname=User \
--admin-email=admin@example.com \
--admin-user=admin \
--admin-password=admin123 \
--language=en_US \
--currency=USD \
--timezone=America/New_York \
--use-rewrites=1
```

Accéder au backend pour vérifier le bon fonctionnement : `[base_url]/admin`**.**

C - Mise en place de l'environnement local avec Docker ou DDEV

Pour un environnement local simple, stable et reproductible, **Docker** (ou **DDEV**) est essentiel. L'intérêt principal est de pouvoir travailler de manière standardisée, quelle que soit la machine de développement. Voici un guide rapide :

Mise en place avec Docker Compose :

1. **Créez un fichier** `docker-compose.yml` **:** Exemple minimal pour héberger Magento :

```
version: "3.7"

services:
```

```yaml
web:
  image: nginx:latest
  volumes:
    - ./magento2:/var/www/html
    - ./nginx.conf:/etc/nginx/nginx.conf
  ports:
    - "80:80"
  depends_on:
    - php

php:
  image: php:8.1-fpm
  volumes:
    - ./magento2:/var/www/html
  depends_on:
    - db
  environment:
    - COMPOSER_ALLOW_SUPERUSER=1
    - PHP_MEMORY_LIMIT=2G

db:
  image: mysql:8.0
  volumes:
    - db_data:/var/lib/mysql
  environment:
    MYSQL_ROOT_PASSWORD: root
    MYSQL_DATABASE: magento
    MYSQL_USER: magento
    MYSQL_PASSWORD: magento

volumes:
  db_data:
```

Configurer Magento dans Docker :

- Copiez le code Magento dans un dossier local (`./magento2` dans cet exemple).
- Lancez les services :

```
docker-compose up -d
```

Mise en place avec DDEV :

DDEV est plus simple d'utilisation pour Magento. Après installation de DDEV, procédez ainsi :

1. Initialiser le projet Magento :

```
ddev config --project-type=magento --docroot=pub --create-docroot
```

Installer Magento :

```
ddev composer create magento/project-community-edition
```

Conclusion :

À ce stade, les développeurs auront :

1. Une bonne compréhension des bases de Magento (structure, fichiers clé, API).
2. Une instance Magento fonctionnelle installée via Composer.
3. Un environnement de développement isolé et optimal (Docker ou DDEV).

Ils seront prêts à entrer dans le vif du sujet avec **le développement de modules personnalisés** dès le jour suivant.

- **1.2 Préparer un nouveau projet e-commerce**

Objectif :

Cette étape consiste à poser les premières fondations pour notre projet e-commerce. Nous allons :

1. Configurer Magento pour prendre en charge un **catalogue multimarques**, une fonctionnalité fréquemment utilisée dans des projets multi-boutiques, où plusieurs marques peuvent avoir chacune leurs propres catégories ou produits spécifiques.
2. Créer un **thème Magento personnalisé** (Frontend), qui servira de base pour toutes les personnalisations graphiques futures.

A - Configurer Magento Backend pour un catalogue multimarques

Magento offre une puissante fonctionnalité native compatible avec des projets **multi-boutiques** ou **multimarques**. Cela permet de gérer plusieurs instances de boutiques ou marques via une seule installation Magento. Chaque "boutique" peut, par exemple, disposer de son propre **catalogue de produits**, ses propres prix, devises ou langues.

1. Concepts clés : niveaux de structure dans Magento :

Magento utilise une hiérarchie à **3 niveaux** pour organiser les structures multiboutiques :

1. **Global (Scope Global)** : Affecte toute l'installation Magento.
    - Exemples : Monnaies, options de configuration globales.
2. **Site Web (Website)** : Division principale. Chaque **site web** peut avoir sa propre base de clients, de devises et de catalogues de produits.
    - Exemples : "Site Européen", "Site Américain".
3. **Boutique (Store)** : Chaque site web peut comporter plusieurs boutiques. Cela permet d’afficher des catalogues différents sous un même site web.
    - Exemple : Une boutique "Vêtements" et une boutique "Chaussures" au sein du même site web.
4. **Vue de la boutique (Store View)** : Gestion des traductions ou affichages spécifiques.
    - Exemples : Langues multiples (anglais, français, etc.).

Cette architecture est idéale pour des catalogues multimarques.

---

2. Création d'une configuration multimarques :

Étape 1 : Créer un nouveau "site web" :

1. Accédez à l’administration Magento :
   `Stores > Settings > All Stores`.
2. Cliquez sur **"Create Website"**, puis fournissez les informations nécessaires :
   - **Name** : Nom explicite (ex. : "Marque A").
   - **Code** : Code unique (ex. : `marque_a`).
   - **Sort Order** : Ordre d'affichage si plusieurs sites coexistent.

Étape 2 : Ajouter une boutique à ce site web :

1. Accédez à nouveau à :
   `Stores > Settings > All Stores`.
2. Cliquez sur **"Create Store"** :
   - Associez cette boutique au **site web** créé (ex. : "Marque A").
   - **Name** : Nom clair (ex. : "Boutique de Marque A").
   - **Root Category** : Choisissez une catégorie racine (vous devrez probablement en créer une nouvelle pour chaque marque).

Étape 3 : Configurer une vue de boutique :

1. Accédez à nouveau à :
   `Stores > Settings > All Stores`.
2. Cliquez sur **"Create Store View"** :
   - Sélectionnez la boutique associée à votre site web (ex. : "Boutique de Marque A").
   - **Code** : `marque_a_fr` (par exemple pour la langue française).
   - **Status** : `Enabled`.
   - **Sort Order** : Définissez un ordre d'affichage.

Étape 4 : Configurer des URLs propres pour chaque marque :

Une fois vos boutiques créées :

1. Modifiez leurs configurations d’URL individuelles :
   - **Stores** > **Configuration** > **General** > **Web**.
   - Changez les valeurs de `Base URL` pour chaque site web (ex. : `https://marque-a.local/`).
2. Modifiez votre fichier `.htaccess` ou votre configuration serveur pour pointer chaque domaine ou sous-domaine vers les bons répertoires.

B - Créer un thème personnalisé de base

Un thème Magento est essentiel car il contrôle tous les aspects visuels de votre site. Ici, nous allons créer un **thème personnalisé basique**, qui servira de point de départ pour des ajustements esthétiques spécifiques à votre catalogue.

---

1. Structure d’un thème Magento

Tous les thèmes Magento sont organisés sous le dossier suivant : `app/design/frontend`.

La structure d’un thème personnalisé ressemble à ceci :

```
app/
└── design/
    └── frontend/
        └── VendorName/ThemeName/
            ├── etc/
            │   └── view.xml
            ├── web/
            │   ├── css/
            │   ├── js/
            │   └── images/
            ├── Magento_Theme/
            │   ├── templates/
            │   ├── layout/
            │   └── web/
            ├── theme.xml
            └── registration.php
```

2. Étape 1 : Déclarer le thème dans `theme.xml`

Créez votre dossier de thème : `app/design/frontend/MonEntreprise/MonTheme/`.

Dans ce dossier, ajoutez un fichier nommé `theme.xml`, qui contient des métadonnées pour déclarer le thème :

```xml
<theme xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:framework:Config/etc/theme.xsd">
    <title>Mon Thème Personnalisé</title> <!-- Nom du thème -->
    <parent>Magento/blank</parent> <!-- Héritage du thème Blank de Magento -->
</theme>
```

Important :
Magento recommande d’utiliser Magento/blank comme parent pour partir sur un design minimaliste, sauf si vous avez besoin d’un thème plus riche graphiquement, auquel cas vous pourrez utiliser Magento/luma comme parent.

3, Étape 2 : Enregistrer le thème dans Magento

Créez un fichier nommé `registration.php` dans votre dossier de thème : `app/design/frontend/MonEntreprise/MonTheme/registration.php`.

Voici le contenu :

```php
<?php
use Magento\Framework\Component\ComponentRegistrar;

ComponentRegistrar::register(
    ComponentRegistrar::THEME,
```

```
    'frontend/MonEntreprise/MonTheme',
    __DIR__
);
```

4. Étape 3 : Configuration des médias avec `view.xml`

Ajoutez un fichier `view.xml` dans :
`app/design/frontend/MonEntreprise/MonTheme/etc/view.xml`.

Ce fichier permet de configurer les tailles des images utilisées dans le thème. Exemple de contenu minimal :

```
<view xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:framework:Config/etc/view.xsd">
    <media>
        <images module="Magento_Catalog">
            <image id="category_page_grid" type="small_image">
                <width>240</width>
                <height>300</height>
            </image>
        </images>
    </media>
</view>
```

5. Étape 4 : Appliquer le thème au site

Une fois votre thème créé :

1. Accédez au backend Magento :
   `Content > Design > Configuration`.
2. Cliquez sur "Edit" pour la boutique ou la vue à laquelle vous souhaitez assigner le thème.
3. Sélectionnez votre thème personnalisé dans le menu déroulant.
4. Videz le cache Magento pour appliquer les modifications :

```
php bin/magento cache:flush
```

Conclusion

À la fin de cette étape, vous aurez :

1. Une configuration Magento capable de supporter un **catalogue multimarques** grâce à la gestion multi-boutiques.
2. Un **thème personnalisé de base**, prêt pour des personnalisations avancées.

Vous êtes maintenant prêt à passer aux customisations spécifiques pour améliorer l’apparence et gérer vos produits via une intégration comme l’API Amazon lors des sessions suivantes !

1.3. Introduction à l’API Amazon

Objectif :

L’objectif de cette section est de comprendre et de maîtriser les bases de l’Amazon **Product Advertising API** (PA API). Cette API permet d'accéder aux données produits d'Amazon pour les intégrer dans un autre site, notamment pour afficher des fiches produits, récupérer des catégories, ou même réaliser des associations publicitaires.

Nous allons explorer :

1. Les concepts clés de l'API : Authentification, gestion des clés API et points d'accès (endpoints).
2. L’analyse de la documentation officielle pour extraire des informations pertinentes.
3. Les outils pour tester l'API, comme **Postman** et **cURL**.

A. Concepts clés de l’Amazon Product Advertising API

L'Amazon Product Advertising API permet de :

- Rechercher des produits sur la base de critères spécifiques (nom, catégorie, ID ASIN...).
- Accéder aux détails produits : images, description, prix, stock, liens affiliés, etc.
- Explorer des catégories ou types de produits.
- Récupérer des recommandations basées sur des produits.

1. Authentification à l'API

Pour accéder à l'API, Amazon exige une authentification stricte. Vous devez fournir une signature numérique pour chaque requête, basée sur :

- Une **clé d'accès publique**.
- Une **clé secrète**.
- L'**horodatage** de la requête.
- Les paramètres d'URL (organisés par ordre alphabétique).

La signature est un hash HMAC-SHA256 généré à partir de ces éléments.

Étape 1 : Récupérer vos clés d’accès

1. Créez un compte avec le **programme Amazon Associates** :
   Lien Amazon Associates.
2. Rendez-vous dans la section **API Tools**.
3. Générez vos clés d’accès :
   - **Access Key ID** : clé publique utilisée pour identifier vos requêtes.
   - **Secret Access Key** : clé privée utilisée pour signer vos requêtes.

Lorsque vous travaillez avec des clés, assurez-vous de ne jamais exposer votre **Secret Access Key** (dans un projet open-source par exemple).

Étape 2 : Les Headers obligatoires pour l'authentification

Pour chaque requête, les headers suivants doivent être inclus :

1. `Host` : L'URL de l'endpoint ciblé (ex. : `webservices.amazon.com`).
2. `X-Amz-Date` : Timestamp UTC sous format ISO8601 (ex. : `20231103T120000Z`).
3. `Authorization` : Contient la signature d'authentification générée.

Vous pouvez utiliser des librairies tierces comme **AWS SDK for PHP** qui simplifient la gestion de la signature.

Exemple de signature (en PHP) :

Voici un exemple de génération manuelle d'une signature :

```php
$method = 'GET';
$host = 'webservices.amazon.com';
$uri = '/paapi5/searchitems';
$accessKey = '<VotreAccessKey>';
$secretKey = '<VotreSecretKey>';
$region = 'us-east-1';
$service = 'ProductAdvertisingAPI';
$timestamp = gmdate('Ymd\THis\Z');
$date = gmdate('Ymd');

$params = [
    'Keywords' => 'laptop',
    'Marketplace' => 'www.amazon.com',
    'PartnerTag' => '<VotrePartnerTag>',
    'PartnerType' => 'Associates',
    'ItemCount' => 10,
    'Resources' => [
        'Images.Primary.Medium',
        'ItemInfo.Title',
        'ItemInfo.Features',
        'Offers.Listings.Price'
    ],
];

$queryString = http_build_query($params, '', '&', PHP_QUERY_RFC3986);

// Construction de la chaîne pour la signature
$canonicalRequest = implode("\n", [
    $method,
    $uri,
    $queryString,
    "host:$host",
    "x-amz-date:$timestamp",
    '',
    "host;x-amz-date",
    hash('sha256', '')
```

```
]);

$stringToSign = implode("\n", [
   "AWS4-HMAC-SHA256",
   $timestamp,
   "$date/$region/$service/aws4_request",
   hash('sha256', $canonicalRequest)
]);

$signingKey = hash_hmac('sha256', $date, "AWS4$secretKey", true);
$signingKey = hash_hmac('sha256', $region, $signingKey, true);
$signingKey = hash_hmac('sha256', $service, $signingKey, true);
$signingKey = hash_hmac('sha256', 'aws4_request', $signingKey, true);

$signature = hash_hmac('sha256', $stringToSign, $signingKey);
```

Vous inclurez cette signature dans vos headers `Authorization`.

---

2. Endpoints et gestion des régions

L'API propose différents points d'accès (endpoints) en fonction des zones géographiques ou marchés cibles. Voici quelques exemples :

- US : `webservices.amazon.com`
- FR : `webservices.amazon.fr`
- UK : `webservices.amazon.co.uk`

Votre configuration de requête (`Marketplace`) doit correspondre à la région choisie.

---

3. Analyse de la documentation officielle

La documentation officielle reste la source la plus fiable pour maîtriser la **Product Advertising API** :

1. Documentation officielle de l'API Amazon PA5.
2. Les concepts clés expliqués :
    - Requêtes disponibles : `SearchItems`, `GetItems`, `GetVariations`, etc.
    - Paramètres à inclure dans les requêtes.
    - Ressources retournées : informations détaillées sur les produits (Images, Prix, Évaluations, etc.).

Chaque méthode contient des exemples prêts à tester.

B. Outils pour tester l'API

Un des points essentiels dans la prise en main d'une API comme celle d'Amazon est de la tester efficacement. Magento étant une plateforme back-end, la phase de test est cruciale avant d'implémenter l'API dans vos modules.

---

1. Postman

Postman est un outil graphique permettant de :

- Construire des requêtes API.
- Ajouter des clés d'authentification.
- Voir les réponses et les détails des erreurs (statuts HTTP, corps JSON).

Étapes pour utiliser Postman :

1. Créez une nouvelle requête dans Postman (GET par exemple).
2. Appliquez les headers nécessaires :
    - **Authorization** : Clé générée (voir étape d'authentification).
    - **x-amz-date** : Timestamp.
3. Ajoutez les paramètres dans l'URL :

```
https://webservices.amazon.fr/paapi5/searchitems?
Keywords=laptop&Marketplace=www.amazon.fr
```

Envoyez la requête et vérifiez :

- Code HTTP `200` : Succès.
- Si `403`, vérifiez votre signature ou permissions.

---

2. cURL

Pour les développeurs préférant la ligne de commande, **cURL** est une excellente alternative.

Exemple de requête de recherche d'articles :

```
curl -X GET \
    'https://webservices.amazon.fr/paapi5/searchitems?
Keywords=laptop&Marketplace=www.amazon.fr' \
    -H 'x-amz-date: 20231103T120000Z' \
    -H 'Authorization: AWS4-HMAC-SHA256 Credential=<VotreAccessKey>/...../[Signature]'
```

Conclusion

À la fin de cette section, les participants auront :

1. Une compréhension approfondie des **concepts de l'Amazon PA API**, notamment l'authentification et la gestion des clés.
2. La possibilité de tester leurs implémentations en utilisant soit **Postman**, soit **cURL**.
3. Les bases nécessaires pour intégrer cette API dans un projet Magento (via des modules ou services personnalisés).

Demandes possibles à une IA pour assister l'apprenant :

1. Configuration initiale du projet Magento

- *"Comment installer Magento 2 avec Composer depuis mon IDE ?"*
- *"Génère les étapes pour configurer un environnement de développement Magento local sur ma machine (avec XAMPP/Docker/Vagrant)."*
- *"Quels fichiers dois-je configurer pour connecter Magento à une base de données MySQL existante ?"*
- *"Comment activer le mode développeur dans Magento, et pourquoi est-ce nécessaire pour ma formation ?"*

---

2. Développement d'un module Magento

- *"Peux-tu me guider pour créer un module Magento de base nommé* `AmazonIntegration` *?"*
- *"Quels fichiers composer.json doivent être inclus pour un nouveau module Magento ?"*
- *"Génère un squelette de script pour enregistrer un module dans Magento (*`registration.php`*)."*
- *"Comment déclarer un module dans le fichier* `module.xml` *?"*
- *"Peux-tu m'expliquer à quoi servent les fichiers* `etc/config.xml` *et* `etc/di.xml` *?"*

---

3. Intégration avec l'API Amazon

- *"Quels sont les prérequis pour utiliser l'API Amazon dans un projet PHP ?"*
- *"Comment puis-je ajouter une bibliothèque comme* `guzzlehttp/guzzle` *à mon projet via Composer ?"*
- *"Génère un exemple de code pour se connecter à l'API Amazon et récupérer une liste de produits."*
- *"Peux-tu m'expliquer comment conserver mes clés API AWS en toute sécurité ?"*
- *"Comment structurer une méthode pour synchroniser des produits depuis Amazon dans une classe helper de Magento ?"*

---

4. Création des produits et catégories dans Magento

- *"Comment utiliser le modèle Magento pour insérer un nouveau produit dans la base de données ?"*
- *"Montre-moi un exemple d'utilisation de la classe* `\Magento\Catalog\Model\Product` *pour créer un produit simple."*
- *"Comment créer automatiquement une catégorie dans Magento et y associer des produits ?"*

- *"Génère un exemple de code pour mapper les données JSON des produits Amazon aux champs des produits Magento."*

---

5. Gestion des schémas de base de données

- *"Montre-moi comment créer une table personnalisée dans Magento à l'aide de* `InstallSchema` *pour stocker des données Amazon importées."*
- *"Comment puis-je utiliser les* `UpgradeSchema` *pour ajouter de nouvelles colonnes ou modifier une table existante ?"*
- *"Quels outils de Magento permettent de gérer le cache après des changements dans le schéma de la base de données ?"*

---

6. Gestion des logs et du débogage

- *"Montre-moi comment ajouter des logs personnalisés dans un module Magento pour suivre les problèmes."*
- *"Comment activer les logs Magento natifs (report, debugger) pour surveiller les erreurs de mon module AmazonIntegration ?"*
- *"Génère un modèle de gestionnaire d'exception pour capturer les erreurs API Amazon et enregistrer les détails dans un fichier de log."*

---

7. Test de l'environnement

- *"Quels sont les différences entre les modes de déploiement (Default, Developer, Production) dans Magento ?"*
- *"Comment puis-je valider que mon module Shopify est correctement activé dans Magento ?"*
- *"Guide-moi pour tester une connexion API depuis un script PHP embarqué dans Magento."*
- *"Comment surveiller la performance des requêtes API Amazon et optimiser leur intégration dans Magento ?"*

Exemple de cas pratiques pendant la première journée :

Pour les concepts d'importation :

- *"Génère un exemple de tâche cron pour exécuter automatiquement la synchronisation des produits amazoniens."*
- *"Comment puis-je paginer automatiquement les résultats des produits Amazon pour des catalogues volumineux ?"*

Pour le débogage dynamique :

- *"Aide-moi à créer et inscrire une commande de console Magento pour tester les appels aux API Amazon directement depuis l’IDE."*

- *"Quels outils de Magento peuvent être utilisés pour profiler et déboguer les lenteurs causées par des intégrations tierces ?"*

Pour les concepts de sécurité :

- *"Comment utiliser la classe Magento* `\Magento\Framework\Encryption\EncryptorInterface` *pour sécuriser mes clés API dans le fichier env.php ?"*
- *"Peux-tu générer un exemple d'injection de dépendance (Dependency Injection) pour protéger mes clés API et éviter que les secrets soient codés en dur ?"*

Pourquoi ces demandes sont bénéfiques pour un apprenant :

1. **Gain de temps** : Les apprentis peuvent déléguer des tâches complexes ou récurrentes à l'IA, tout en se concentrant sur l'analyse et la compréhension.
2. **Meilleures pratiques** : L'IA peut introduire progressivement des techniques avancées ou optimisées, comme l'intégration des standards Magento.
3. **Renforcement des concepts** : En posant des questions spécifiques, les apprenants peuvent mieux comprendre le fonctionnement des divers composants de Magento.
4. **Débogage et dépannage** : Un développeur en herbe peut s'assurer que les erreurs courantes ou les configurations complexes sont détectées et corrigées efficacement.