UNIVERSITÉ
CAEN
NORMANDIE

Rapport de Projet annuel

Alerte Prix

Encadré par Mr SPANIOL Marc

PIGNARD Alexandre - 21701890
BOCAGE Arthur - 21806332

Master 1 Informatique - Promotion 2021-2022

21 mars 2022

Table des matières

1 Introduction - Vision et Stack

Pour ce projet annuel nous avons voulu créer une application dans des conditions les plus réelles possibles avec comme objectif, non seulement de répondre aux attentes demandées par le sujet, mais aussi de nous former sur des outils, techniques et conditions semblables à une vraie mission professionnelle free-lance.

Pour cela, nous avons avant toute écriture de code, longtemps rechercher sur quels stack, frameworks et languages nous reposer pour mener à bien notre projet ; nous avons donc décidé d'être sous une stack type LAMP enrichie qui est :

1. Hébergement : Ubuntu 20.04
2. Serveur Web : Apache 5
3. système de gestion de bases de données : MySQL
4. Framework web : Symphony 5
5. Langage back-end : Python 3.9

1.1 Ubuntu 20.04

Pour ce choix nous nous sommes avant tout basés sur un constat, actuellement Ubuntu est la première distribution linux utilisée dans le millieu de l'hébergement web, de plus ayant tous les deux maintenant plusieurs années d'expériences avec cet OS, le choix fut vite validé . https ://w3techs.com/technologies/details/os-linux

1.2 Apache2

De part la nature du projet, un serveur web plus simple d'accès tel que Nginx, n'aurai pas fait l'affaire, en effet du à la génération de pages utilisateurs, nous avons du opter pour un serveur dynamique et nous nous sommes arrêtés sur Apache2, référence dans le milieu et répondant parfaitement à nos besoins.

1.3 Symphony 5

Ayant abordé les grands principes du design de frameworks php pour la première fois en L3, et poursuivi cet apprentissage lors du premier semestre de M1, nous avons voulu capitaliser sur cette lancée et utiliser un outil et une somme de connaissances demandées dans le milieu professionnel afin de perfectionner nos compétences dans cette technologie.

1.4 Python 3.9

Étant un langage avec lequel nous étions tous les deux assez familiers, nous avons décidé d'opter pour ce choix après une analyse comparative des besoins et problèmes que nous avons commencé à isoler, en particulier toute la partie de surveillance des prix et envoi d'alerte.

1.5 MySQL

Pour le design de la base de données du projet, 2 choix se sont posés, continuer sur une technologie vue en cours avec MongoDB, étant certes adaptée au stock de document json en grande quantités, cependant, au fil de la réalisation du projet nous avons du changer entièrement notre architecture de données, étant, encore malheureusement, assez mal supportée du côté de Symfony, nous nous sommes donc rabattus sur MySQL, un classique remplissant parfaitement son rôle et compatible avec une grande majorité de l'écosystème web actuel.

2 Interface utilisateur

2.1 Le site web

Comme dit précédemment le site web a été réalisé sous Symfony, en PHP donc. Symfony est basé sur l'architecture MVC (Model, Vue, Controleur) dont il permet de grandmeent simoplifier la mise en place. Le fonctionnement du site est assez sommaire, on se connecte et on ajoute à une liste associée à l'utilisateur des objets avec des liens ainsi qu'un prix auquel l'utilisateur souhaite recevoir une notification lorsque l'objet qu'il souhaite acheter se trouve en dessous de ce prix.
Le fonctionnement de Symfony repose sur des contrôleurs qui permettent de définir des routes vers des pages HTML.TWIG (permettant l'accès à des variables que le contrôleur lui fourni). Les contrôleurs se basent sur des entités afin d'écrire et de lire dans la base de données grâce au module Doctrine. Une entité représente un objet.

Nous avons logiquement deux entités dans ce site, l'utilisateur (un email et un mot de passe) et l'alerte (un article et son lien). La représentation dans notre base de données est elle plus complexe, afin de la rendre plus versatile (détaillé dans la partie dediée à la base de données du rapport).

2.2 Les contrôleurs

Le site comprends 8 contrôleurs, chacun associé à un template html, et donc à une page. Voici les contrôleurs :

- Homepage : Permet le rendu de la page d'accueil.
- About : Permet le rendu de la page d'informations du site.
- MyAccount : Permet d'avoir une page avec l'affichage des informations de l'utilisateur et la possiblité de supprimer son compte.
- List : Permet d'afficher les alertes d'un utilisateur.
- Object : Permet l'affichage des détails d'une alerte, d'ajouter un lien à cette alerte et de la supprimer.
- AddObject : Permet d'avoir une page et un formulaire d'ajout d'alerte.

- Registration et Security : Permettent l'authentification d'un utilisateur ou la création de compte.

2.3 Les entités

Dans Symfony, les entités sont des objets que l'on va pouvoir insérer dans la base de données. Notre site comporte cinq entités que nous pouvons retrouver dans la diagramme précédent.

- User : Un objet représentant un utilisateur et stocké dans la table du même nom.
- Article : Le nom d'un objet à traquer avec son ID que l'on va associer à un Lien
- Links : Un objet représentant un lien et associé à un objet Article via son ID.
- ArtcleUser : Un objet permettant de faire le lien entre un Article et un User et comprenant le prix que l'utilisateur aura précisé.
- ObjectRequest : Un objet qui représente la demande d'un utilisateur à ajouter une alerte, sert à avoir un historique des demande d'alertes et à ne créer qu'un seul formulaire pour écrire dans toutes les tables en même temps.

Le dossier Repository comprends les classes associées aux Entités qui permettent de gérer les opérations vers la base de données (comme écrire, mettre à jour ou encore supprimer). Ces classes sont générées automatiquement par le framework Symfony.

2.4 CRUD

CRUD (create, read, update, delete), en français (créer, lire, mettre à jour et supprimer) est un acronyme pour les façons dont on peut fonctionner sur des données stockées. C'est un moyen mnémotechnique pour les quatre fonctions de base du stockage persistant. Dans notre application ces fonctions sont toutes disponibles : il est possible de créer un utilisateur, de le supprimer, de créer des alertes, de mettre à jour les liens ou bien supprimer cette alerte.

Pour ce qui est de la lecture le site dispose d'une page permettant d'accéder aux informations de l'utilisateur, d'une page listant les différentes alertes d'un utilisateur qui peuvent être elles mêmes être ouvertes dans une page afin d'en accéder aux détails.

2.5 Les formulaires

Le dossier Form contient les fichiers nécessaires à la création de formulaires. Ils sont au nombre de trois.
Le premier : UserType permet de générer un formulaire lié à l'entité User et permet d'avoir les formulaires d'inscription et de connexion du site.

Le deuxième fichier est ObjectRequestType, il permet de ne créer qu'un seul formulaire lié à une entité ObjectRequest qui permettra d'écrire dans toutes les tables en même temps dans la base de données. ObjectRequest est en quelque sorte une entité tampon qui fait le lien avec les entités dans la base de données.

Le dernier fichier est LinksFormType. Il permet d'avoir un formulaire d'ajout de lien qui sera utilisé dans la page de détails d'un objet afin de lui ajouter un lien.

Les fichiers HTML sont stockés dans le dossier Templates et utlisent TWIG, technologie qui nous permet d'effectuer des opérations (notamment des boucles) et d'accéder à des variables sans ouvrir de balise php, ce qui est très pratique.
Les templates ont un fonctionnement simple, on dispose d'une base qui sera implémentée par chaque autre template. La base comprends le menu de notre application web et les autres templates vont venir écrire dans le body du site.

Les fichiers css sont stockés quant à eux dans le dossier public et son sous-dossier css. Nous avons deux fichiers css, le premier est bootstrap.css, nous avons fait le choix d'utiliser bootstrap (plus particulièrement bootswatch) afin d'avoir un design propre et de pouvoir se concentrer sur les fonctionnalités du site. Le deuxième fichier css est responsive.css, son rôle est de garantir que le site soit responsive en modifiant la taille du body en fonction de la taille de l'écran de l'utilisateur.

Au sein du dossier public se trouvent aussi les images du site (dans un dossier images) mais aussi les fichiers JavaScript, dans le dossier js. Ils sont au nombre de deux et ont un fonctionnement très similaire : avertir un utilisateur lorsqu'il va vouloir supprimer une alerte ou encore son compte.

Enfin, on va aussi retrouver notre fichier index.php.

2.6 Les autres dossier et fichiers du framework

Les autres dossiers sont des dossiers auxquels nous n'avons que peu touché, il s'agit des dossiers générés par Symfony et sont, pour certains, rarement amenés à être modifié par le développeur (sauf dans certains cas).

- bin : dossier contenant les fichiers executables (comme bin/console)
- config : dossier contenant les fichiers de configuration de notre application
- migrations : dossier contenant les paramètres des requêtes exécutées par doctrine qui permettent d'interagir avec la base de données.
- tests : dossier des tests automatiques (que nous n'avons pas utilisé)
- tools : dossier contenant des outils que nous avons pu utiliser (comme php fixer qui nous a permis de vérifier le bon respect de la syntaxe php dans nos documents).
- translations : dossier permettant de stocker les fichiers de traduction du site (inutilisé ici).
- var : dossier contenant des fichiers générés automatiquement par le framework.
- vendor : les dépendances de notre framework.

Enfin, deux fichiers importants méritent notre attention, le premier est le fichier .env qui contient des paramètres importants au bon fonctionnement du site, notamment l'adresse de notre base de données ou encore le mode de fonctionnement du site (dev ou prod). Enfin, ce fichier et l'un des fichiers les plus sensibles du projet car il contient l'addresse de notre Base de données ainsi que le mot de passe nécessaire pour y accéder. Une erreur à éviter est d'oublier de configurer un fichier .gitignore afin de ne pas envoyer ce fichier sur

Git. Le deuxième fichier est composer.json permettant le bon fonctionnement de composer (permettant d'installer des packages utiles à Symfony) ou encore Symfony lui même.

3 Back-End

3.1 Hébergement

Pour nous assurer une disponibilité, un total contrôle sur le contenu du site et de rester dans la thématique de développement d'une "Mission free-lance" nous avons préféré opter pour une machine virtuelle distante plutôt qu'un hébergeur web classique ; nous permettant de remplir à la fois la contrainte d'hébergement de site web tout en mettant à disposition une instance pour pouvoir faire tourner notre application Python se chargeant de récupérer les prix sur différents site webs .

3.2 Serveur et Service web

- Apache2

Notre site web est servi via apache2, après avoir installer le serveur web sur notre machine distante et ouvert les ports 80 et 443 la route par défaut pour cibler notre dossier de déploiement, nous y avons installé notre site web et synchronisé Symfony avec Apache2 rendant ainsi accessible notre site à l'ip : 139.162.131.82 .

- Dns

Toujours dans l'optique de réaliser un rendu livrable et fonctionnel, nous nous devions de créer un acces par dns, pour rappel,un serveur DNS (Domain Name System, ou Système de noms de domaine en français) est un service dont la principale fonction est de traduire un nom de domaine en adresse IP. Pour simplifier, le serveur DNS agit comme un annuaire que consulte un ordinateur au moment d'accéder à un autre ordinateur via un réseau. Autrement dit, le serveur DNS est ce service qui permet d'associer à site web (ou un ordinateur connecté ou un serveur) une adresse IP, comme un annuaire téléphonique permet d'associer un numéro de téléphone à un nom

d'abonné.
Dans notre cas nous avons optés pour un dns gratuit, fourni par le service No-ip et y avons associé notre ip statique avec le nom "pricetrackerm1.hopto.org" donnant ainsi à l'internaute une adresse plus lisible pour la connection, dans notre cas l'hébergeur fournissant une ip statique pour se connecter au serveur, il n'y eu que la mise en relation de l'enregistrement DNS, cependant si le besoin de migrer vers une instance hébergée de manière privée par un particulier avec une ip dynamique se présente, une migration reste tout à fais possible.

- Certification TLS (SSL)

Un certificat TLS (Transport Layer Security, aciennement SSL) est un certificat électronique qui authentifie l'identité d'un site Web et permet une connexion chiffrée. SSL signifie « Secure Sockets Layer », c'est un protocole de sécurité qui crée un lien chiffré entre un serveur Web et un navigateur Web. Dans notre situation nous avons opté pour une solution open-source **Let's Encrypt** mise en place par le programe python **certbot**, faisant la demande auprès de l'autorité de certification ainsi que mettant en place une redirection automatique vers une version cryptée du site.

3.3 Base de données - MySQL

3.3.1 Design de la base de données

Le site web utlise les tables User, Article, Links, ArticleUser ainsi que ObjectRequest. Voici ce à quoi elles servent :

- User est la table contenant les utilisateurs
- Article contient le nom d'un objet à placer en alerte
- Links contient des liens associés aux objets de la table Article
- ArticleUser contient des Articles associés aux User et le prix qu'ils auront entrés
- ObjectRequest contient les requête d'un utilisateur voulant créer une alerte (nom de l'objet, lien, prix et id de l'user).

On a donc le schéma suivant :

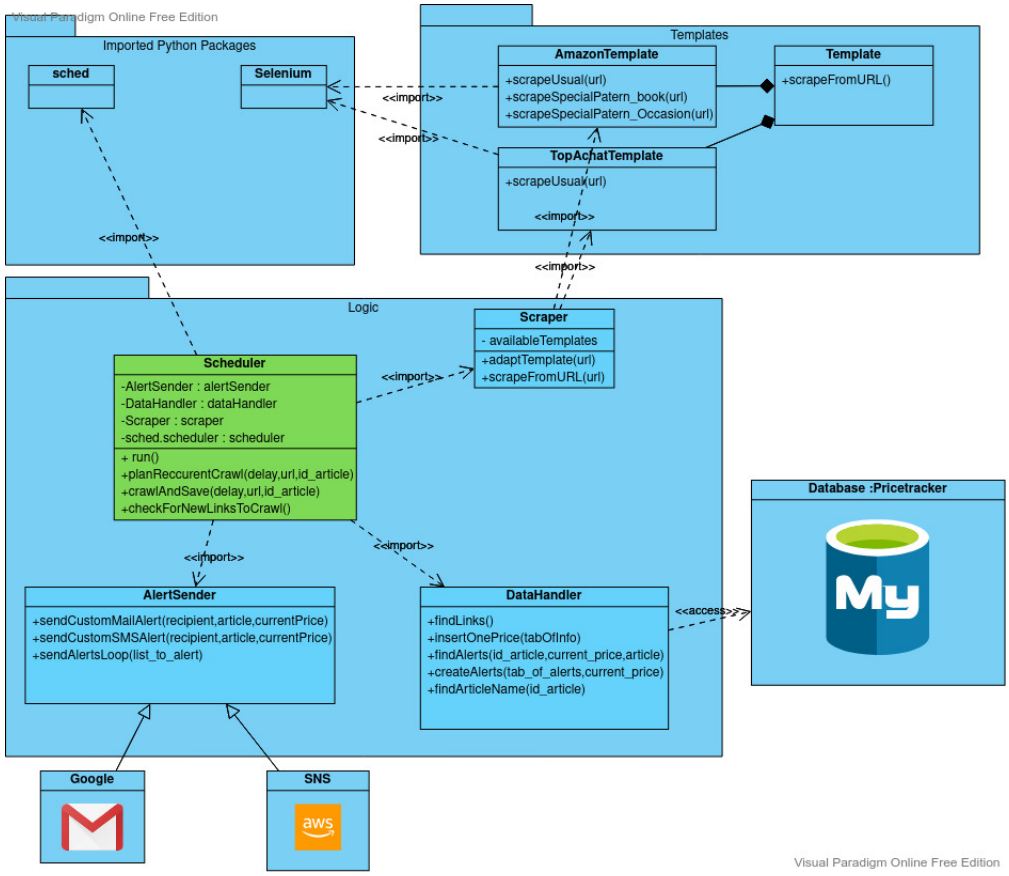
La table User représente l'utilisateur, la table Article représente un article associé à un lien grâce à la table Links. La table ArticleUser nous permet de faire l'association entre un Utilisateur et un article auquel il souhaite être notifié avec un prix dont il pourra choisir le montant.

3.3.2 Création de la base de données via Doctrine

La base de données est gérée par Doctrine, module de Symfony permettant l'automatisation des requêtes vers la base de données. Pour créer la base de données il faut d'abord configurer ses entités notamment les contraintes que l'on peut définir avant même de créer nos tables, par exemple un article peut avoir plusieurs liens, en revanche un lien ne peut avoir qu'un seul article d'associé (ManyToOne). Une fois les entités crées, on exécute la commande permettant à Doctrine de créer les tables et leurs contraintes. Nous utilisons donc des procédures stockées.

3.4 Python

Diagramme des classes



Concept du programme

Ce programme python est le coeur logique de la traque des prix, fonctionnant en parallèle du site web sur le serveur, permettant de récupérer les prix de manière périodique et contrôlée tout en envoyant des notifications aux utilisateurs si leurs prix voulus ont été atteints, Il se décompose en 4 grandes classes logiques qui sont :

1. Le Scraper web (Scraper)
2. L'alerteur (AlertSender)
3. Le gestionnaire de base de données (DataHandler)

4. Le plannificateur de tâches (Scheduler)

Ainsi que plusieurs classes d'aide et de sous traitements :

— DefaultTemplate
— UnitTester

Un cycle de récupération fonctionne de la manière suivante, le programme regarde quelles URL récupérées par l'objet **DataHandler** dans la table Links de la base de données il à traiter, et prévoit d'exécuter la récupération des pages web après un interval défini prévu par le **Scheduler**, étant dans un context de faible variation de prix, nous avons fixé un interval de 12 heures entre chaque cycle , à noté que nous pouvons modifier cette variable au besoin.
Une fois l'interval passé, Le **Scraper** web trouve le bon **Template** pour le site enregistré et lui demande de récupérer les informations nécessaires sur la page, ces informations vont ensuite être passées à l'objet **AlerteSender** qui recherchera tout les utilisateurs à prévenir pour un produit ainsi que leur prix voulu, si le prix trouvé est inférieur ou égal au prix voulu, il envois une notification par mail et sms (à noté que nous sommes toujours en environnement de test pour cette fonctionnalité). Une fois ce cycle fini, une nouvelle fonction de récupération est planifiée pour un nouveau cycle .

Scraper

Nous voulions à la base utiliser l'api officielle de chaque site, pour accéder aux catalogues de produits, mais dans la plupart des cas elles n'étaient pas disponibles, pour Amazon par exemple, une api existe, mais est réservée uniquement aux partenaires commerciaux, notre utilisation n'entrant pas dans ce cadre, nous avons dû improviser. Deux solutions se sont alors présentées à nous, trouver les requêtes vers la base de données formulées par notre navigateur et leurs formats pour les reproduire à l'intérieur de notre scraper, méthode qui présente de nets avantages, format simple, bon temps de réponse mais malheureusement trop repérable par Amazon, ces derniers n'aimant pas que l'on s'intéresse de trop près à la récupération de prix automatisée sans leur consentement.
Reste donc une manière dérivée de récupérer l'information, certes plus lente et plus lourde à gérer, mais plus difficile à tracer, le requêtage des pages web

associées aux produits pour en extraire le contenu.
Cette méthode n'exclut pas pour autant la possibilité qu'a Amazon et d'autres sites puissent toujours "flag" notre trafic et nous blacklister, ainsi des librairies comme **requests** ne sont pas de bons candidats pour effectuer cette tâche, ayant, dans un premier temps, une signature reconnaissable et facilement exclue, mais aussi, que la plupart des pages actuelles sont générées entièrement via Javascript, il nous fallait donc une solution capable d'émuler un vrai trafic, tout en étant peu reconnaissable.
Cette solution fut apportée par le module **sélénium**, ce dernier permet, après installation de drivers, d'émuler une navigation d'un internaute lambda tout en ayant la possibilité de customiser la signature de l'agent web, nous permettant ainsi de récupérer toutes les informations voulues.

Une fois le problème de la récupération des données résolut, nous sommes alors partis sur une approche modulaire du problème de scraping des sites, avec des **templates** pour chaque site ayant les instructions précises sur comment récupérer le prix, le nom de produit recherché, rendant l'objet scraper responsable de la logique de mise en relation des liens stocké dans la base de données à leurs templates respectifs après analyse du nom de domaine présent dans l'URL, cette méthode à 2 avantages, comme mentionnée plus haut, la modularité permet le rajout de nouveaux sites au scraper de manière simple, le deuxième et lié au risque inhérent à laisser n'importe quel utilisateur insérer de potentiels liens malveillants donnant l'exécution de codes malicieux, qui est, grâce au prétraitement, rendu plus difficile, le scraper ne visitant que des sites White listés.

Fonction principales du Scraper :

— scrapeFromUrl(searchUrl)

  Elle est la fonction de contacte pour les autres modules voulant récupérer un produit, une fois appellée, le scraper trouve le bon Template pour l'url proposé via la fonction adaptTemplateFromUrl() et se renvoit le résulat de la Fonction scrapeFromUrl(url) du Template

— adaptTemplateFromUrl(url)

  Cette fonction à usage interne cherche dans le dictionnaire de templates associant un nom de domaine à un template à exploiter, si le domaine de l'url fournie est présent ou non, et si oui renvoit la fonction du

template à exploiter pour lancer la récupération du prix de l'article

— scrapeTest(searchUrl,html)
Fonction à utiliser un tandem avec les test unitaires, permettant de d'appeler la méthode de scrapeTest de chaque template, ignorant la partie récupération de la page html et analysant directement le html founit avec ses propres fonction de scraping .

Templates

Les templates sont un ensemble de classes représentant chacune un site web à explorer, fonctionnant de pair avec l'objet Scraper ce dernier se basant sur la méthode scrapefromurl() commune à chaque objet héritant de la classe Template pour retrouver les informations de chaque page, elle-même contenant les différentes méthodes de scraping à utiliser pour réussir l'extraction d'informations.

— scrapeFromUrl(searchUrl)

Fonction permettant de récupérer dans un premier temps le html de l'url passé en argument via un webdriver importé, dans notre cas selenium et ensuite le soumettre à plusieurs fonctions de scraping propres à chaque Template avec la librairie **BeautifulSoup 4** pour récupérer le tableau d'information voulues .

— scrapeTest(searchUrl,html)

Fonction appellée par l'objet UnitTester, fonction sur la même idée que scrapeFromUrl mais ignorant la partie de récupération du html, ce dernier étant stocké dans un fichier et passé en argument pour tester les fonctions des scraping.

Tests Unitaires

Pour nous assurer que nos modifications ne modifie pas les résultats précédant et même temps accélérer notre cycle de développement, nous avons créé une suite de tests unitaires via l'objet Unit Tester(), ce dernier se charge de sauvegarder dans un fichier, pour chaque URL testée, le HTML de la page ainsi que le prix attendu en réponse.

— addToTests(url,expectedPrice)

Fonction permettant de récupérer dans un premier temps le html de l'url passé en argument via la fonction scrapeRawHtml du scraper et de le stocker dans un fichier en lui associant un prix à trouver .

— testTemplate(url,html)

Fonction recherchant dans le fichier de tests unitaires le html correspondant une url, et compare le prix implémenté par le testeur et celui trouvé par la fonction scrapeTest() du template de site, si bon, affiche la réussite de test sinon l'échec.

— runAllTests()

Fonction executant tous les tests étants stockés dans le fichier unitTests.pkl.

AlertSender

Cet objet regroupe toutes les fonctions relatives aux notifications à envoyer à l'utilisateur quand le prix défini par celui-ci est inférieur ou égal au prix trouvé par le scraper. Fonctions principales de l'Alert Sender ;

— sendAlertsLoop(self,listToAlert)

Une boucle recevant en paramètre une liste des utilisateurs à prévenir avec toutes les informations pour produire un message personnaliser, lançant les différents formats d'alertes présent.

— sendCustomMailAlert(self,recipient,article,currentPrice)

Fonction d'alerte utilisateur s'occupant d'envoyer des mails, pour la réaliser nous avons eu besoin du module "email" de python, ainsi d'un compte gmail ayant l'autorisation d'envoyer des mails provenant d'application tierces, créant ainsi un mail personnalisé avec les informations extraitent dans la boucle de sendAlertsLoop().

— sendCustomSMSAlert(self,phoneNumber,article,currentPrice)

Fonction d'alerte utilisateur s'occupant d'envoyer des notifications sms, pour la réaliser nous nous sommes reposé sur le service Simple Notification Service que propose la plateforme Amazon webservice ou "SNS", à l'aide de leur module "boto3" on instancie un client se connectant au service "sns" et envoie un sms personnalisé avec les

informations extraitent dans la boucle de sendAlertsLoop().

DataHandler

Objet gérant toute la partie récupération et stockage d'informations dans la base de données, tous les objets voulant interagir avec cette dernière doivent passer par une de ces méthodes, le but principal de cet objet est d'avoir un code organisé et lisible avec toutes les requêtes centralisées dans un seul objet, ce dernier effectue des requêtes avec sur la bdd via librairie de connexion **mysql.connector**, pour chaque action une connexion et un curseur sont crées, et exécutent une demande précise.

— findLinks(self)
Fonction permettant au scheduler de trouver tous les liens à scraper enregistré dans la table textttlinks.

— insertOneLink(self,valuesToInsert)
Appelée dans l'objet Scheduler après un scraping réussi d'une url par l'objet Scraper, enregistre le nom du produit, le liens pour y accéder et le prix trouvé dans la table `price_tracker` de la base de données.

— findAlerts(self,idArticle,currentPrice,article)
Appelée dans l'objet Scheduler après un scraping réussi d'une url par l'objet Scraper, cherche dans la table relation `article_user` les utilisateurs qui sont abonné à cet article et récupère les informations le concernant, retournant le résultats de la fonction createAlerts.

— createAlerts(self,tabOfAlerts,currentPrice)
Reprends les données trouvées par findAlerts() et recherche les informations de contact de l'utilisateur si et les stockent dans un tableau sous forme d'un tuple d'informations si le prix défini par l'utilisateur a était atteint.

— findArticleName(self,idArticle)
Fonction utilisée lors de createAlerts, permet de trouver le nom qu'un utilisateur a donné à un article.

Scheduler

Objet central du programme, il est en charge de toute l'organisation et de la prévision de toutes les tâches à effectuer, prenant en arguments un objet Scraper, un DataHandler et un AlertSender qu'il utilise pour accéder aux ressources dont il a besoin , le tout de manière délayé grâce à la libraire **sched** permettant de planifier des tâches à l'avance.

- planReccurentCrawl(self,delay,url,idArticle)
  Prévoit un appel de crawlAndSave() après un certain délai spécifié par l'utilisateur d'une url via l'objet scheduler du module sched.

- crawlAndSave(self,delay,url,idArticle)
  Engage le scraping d'une page web via la fonction scrapeFromUrl() du scraper, et si le résultat correspond à une récupération valide :
  - Prévoit un autre scraping au même interval via la fonction planReccurentCrawl()
  - Enregistre dans la base de données via le DataHandler
  - Cherche les utilisateurs à alerter avec l'AlertSender .

- checkForNewLinksToCrawl()
  Permet de rajouter à la liste des tâches les nouveaux liens à parcourir rajouter dans la base de données durant l'entre temps du dernier checkForNewLinksToCrawl et relance un même checkForNewLinksToCrawl avec le même délai .

- run()
  Cherche tous les liens dans la table textttlinks et lance la fonction planReccurentCrawl sur chaque lien trouvé, ainsi qu'une tâche checkForNewLinksToCrawl().

4 Possibles améliorations

Système d'intelligence collective

Ayant une base de données regroupant les domaines, prix,et nom de chaque article, si un utilisateur cherche un produit sur un site particulier mais qu'un doublon existe déjà pour cet article mais sur un autre site alors

nous pourrions notifier l'utilisateur du possible choix à suivre en plus,

Détection automatique des prix et nom d'article

Pendant la création de "Template" de scraper, nous avons remarqué une certaine tendance, les éléments contenant les prix et nom d'article contiennent souvent des id ou noms de classes significatifs("Price","ProductName"). De ce constat nous est venu l'idée de remplacer la création de template manuel, par un possible module IA analysant les balises HTML du nouveau site et en retiendra les balises et classes à sélectionner pour trouver le nom et le prix du l'article, avec un filtre dans le style d'un Bayes naïf .

Procédures SQL

Actuellement nos requêtes SQL sont créées à l'intérieur du DataHandler du programme python, nous pourrions déléguer cette partie au gestionnaire My Sql, pouvant créer des interactions semblables à l'aide de procédures, qui en plus de rendre le code python plus lisible seraient plus optimisées, SQL pouvant ainsi précharger certains comportements.

4.1 Du côté du site Web

Le site pourrait profiter de quelques améliorations, en voici une liste (sans ordre d'importance particulier) :

- Ajout de la prise en charge de plusieurs langues (notamment une version anglaise du site).
- Possibilité de modifier le prix d'une alerte
- Possibilité de modifier ou encore de supprimer le numéro de téléphone d'un compte.
- Ajout d'une page tutoriel détaillée.
- Ajout d'un bouton de changement de thème (clair ou sombre).
- Ajout de la possibilité de supprimer toutes ses alertes d'un seul coup dans la page de listes.
- Ajout d'un utilisateur admin ayant accès à toutes les alertes et à tous les comptes.
- Améliorer le système d'authentification afin de pouvoir se connecter avec son compte Google ou encore Facebook.
- Ajout d'une page contenant un contrat de confidentialité et d'une case à cocher pour que l'utilisateur confirme l'avoir lu lors de la création de son compte.