Description/rappel du projet :

Introduction : Le but de cette application est d'analyser plusieurs sites web d'enchères pour créer des statistiques d'achat. Elle permettra d'extraire des données sur les produits mis en vente, d'analyser ces données à l'aide d'algorithmes d'intelligence artificielle, et de mettre à jour un tableau de bord Flask avec les données les plus pertinentes pour la prise de décision d'achat. L'application sera développée en utilisant le langage de programmation Python et plusieurs modules tels que Flask, Tkinter, Sci-kit learn, et d'autres modules à définir selon les besoins évolutifs du projet.

Fonctionnalités :

Bouton de mise à jour des données : cette fonctionnalité permettra aux utilisateurs de mettre à jour manuellement les données collectées sur les sites web d'enchères.

Mise à jour automatique : cette fonctionnalité permettra la mise à jour automatique et quotidienne des données collectées sur les sites web d'enchères.

Installation automatique des paquets manquants ou obsolètes : cette fonctionnalité permettra l'installation automatique des paquets nécessaires à l'exécution de l'application qui manquent ou qui sont obsolètes.

Dashboard : cette fonctionnalité permettra la visualisation et l'analyse des données collectées sous forme de graphiques et de tableaux de bord. Elle sera conçue de manière à ce que les utilisateurs puissent facilement comprendre les tendances et prendre des décisions éclairées.

Fonctionnalités de filtrage et de tri : cette fonctionnalité permettra aux utilisateurs de filtrer et de trier les données collectées selon leurs besoins.

Spécifications techniques :

Plateforme cible : l'application sera développée pour une utilisation sur ordinateur de bureau.

Configuration matérielle minimale requise : le système d'exploitation requis sera Windows 10 ou supérieur et linux. Il est recommandé d'avoir un processeur Intel Core i5 ou supérieur et 8 Go de RAM.

Exigences logicielles minimales requises : le système d'exploitation requis sera Windows 10 ou supérieur linux. Les logiciels tiers nécessaires incluront Python 3.7 ou supérieur, Flask, et d'autres modules à définir selon les besoins du projet.

Interface utilisateur :

Description de l'interface utilisateur : l'interface utilisateur sera conçue de manière à ce qu'elle soit facile à utiliser et à comprendre pour les utilisateurs.

Description de l'expérience utilisateur : l'expérience utilisateur sera conçue de manière à ce qu'elle soit intuitive et agréable pour les utilisateurs.

Tu me proposeras une description exacte de l’interface. L’application ne doit utiliser que les techno python flask

Architecture technique :

a. Description de l'architecture logicielle : L'architecture logicielle de l'application sera basée sur une architecture client-serveur. Le client sera l'interface utilisateur qui sera développée à l'aide de Tkinter pour les graphiques, tandis que le serveur sera une application Flask pour la gestion des requêtes HTTP et le stockage des données.

b. Description des technologies utilisées (langage de programmation, framework, bibliothèques tierces, etc.) : L'application sera développée en Python. Pour le web scraping, nous utiliserons le module BeautifulSoup. Pour le traitement des données, nous utiliserons le module Pandas. Pour la visualisation des données, nous utiliserons Matplotlib et Seaborn pour les graphiques. Enfin, pour l'analyse de données, nous utiliserons scikit-learn aucune base de données ne doit être utilisées

c. Description de l'architecture réseau (serveurs, base de données, etc.) : Le serveur sera une application Flask exécutée sur un serveur web. La base de données sera une base de données SQLite qui sera stockée sur le serveur. Les données seront stockées dans des tables relationnelles pour faciliter l'analyse de données.

Sécurité :

a. Exigences de sécurité (par exemple, cryptage, authentification, etc.) : Les données stockées dans la base de données seront cryptées à l'aide d'un algorithme de cryptage avancé pour garantir leur confidentialité. L'application sera protégée par une authentification basée sur un nom d'utilisateur et un mot de passe.

b. Politique de gestion de la sécurité des données : Nous mettrons en place une politique de sauvegarde et de récupération de données pour garantir que les données sont toujours disponibles en cas de panne du serveur ou de tout autre problème technique. Nous surveillerons également régulièrement les journaux de connexion pour détecter toute activité suspecte.

c. Politique de gestion des droits d'accès : Nous mettrons en place une politique de gestion des droits d'accès pour garantir que seuls les utilisateurs autorisés ont accès aux données stockées dans l'application. Les droits d'accès seront accordés en fonction du rôle de chaque utilisateur.

Performances :

a. Exigences de performances (par exemple, temps de réponse, charge maximale, etc.) : L'application devra avoir une réponse rapide, avec un temps de réponse maximal de 5 secondes. Elle devra également être capable de gérer un nombre important d'utilisateurs et de requêtes en même temps.

b. Politique de sauvegarde et de récupération de données : Nous mettrons en place une politique de sauvegarde régulière des données pour garantir que les données sont toujours disponibles en cas de panne du serveur ou de tout autre problème technique.

c. Politique de gestion de la mémoire et des ressources système : Nous surveillerons régulièrement l'utilisation de la mémoire et des ressources système pour garantir que l'application fonctionne de manière efficace et optimale.

Intégrations :

a. Description des systèmes tiers à intégrer : Aucun système tiers ne sera intégré pour l'instant, mais cela pourra être envisagé pour les évolutions futures de l'application.

b. Exigences d'interopérabilité : L'application sera développée pour être compatible avec toutes les versions de Python.

Intégrations : a. Description des systèmes tiers à intégrer : aucun système tiers n'est prévu à intégrer dans cette version de l'application. b. Exigences d'interopérabilité : la seule exigence d'interopérabilité pour l'application est que les données en entrées soient au format CSV ou Excel. c. Protocoles de communication à utiliser : aucun protocole de communication n'est requis pour cette version de l'application.

Exigences légales : les sites scraper m’appartienne je fais cette application pour la tester

Support et formation : a. Politique de support pour les utilisateurs : les utilisateurs pourront contacter l'entreprise par e-mail pour signaler les bugs ou poser des questions sur l'utilisation de l'application. Une réponse sera fournie dans un délai de 48 heures. b. Politique de formation pour les utilisateurs : une documentation détaillée sur l'utilisation de l'application sera fournie aux utilisateurs. c. Politique de maintenance de l'application : des mises à jour seront fournies périodiquement pour corriger les bugs et améliorer les fonctionnalités de l'application.

Tests et validation : a. Plan de tests : des tests unitaires et d'intégration seront réalisés pour assurer le bon fonctionnement de chaque fonctionnalité de l'application. b. Plan de validation : une validation des exigences et de la qualité de l'application sera effectuée avant le lancement. c. Critères de validation : l'application sera considérée comme valide si elle répond à toutes les exigences énoncées dans les spécifications et si elle est fonctionnelle sans erreur.

Budget et échéances : a. Budget alloué pour le développement de l'application : le budget alloué pour le développement de l'application est de X dollars. b. Échéances pour chaque étape du projet : le développement de l'application devrait être terminé dans X mois, suivi d'une période de test et de validation de X semaines. c. Risques identifiés et stratégie de gestion des risques : les risques identifiés pour le projet comprennent des retards dans le développement, des bugs imprévus et des problèmes de sécurité. Une stratégie de gestion des risques sera mise en place pour minimiser l'impact de ces risques sur le projet.

Contexte du cdc

Introduction : a. Contexte et objectif du projet b. Portée et périmètre du projet c. Description du public cible de l'application

Fonctionnalités : a. Liste des fonctionnalités principales de l'application b. Description détaillée de chaque fonctionnalité c. Diagrammes de cas d'utilisation pour chaque fonctionnalité

Spécifications techniques : a. Plateforme cible (ordinateur de bureau, mobile, tablette, etc.) b. Configuration matérielle minimale requise c. Exigences logicielles minimales requises (système d'exploitation, versions de logiciels tiers, etc.)

Interface utilisateur : a. Description de l'interface utilisateur (mise en page, navigation, icônes, etc.) b. Description de l'expérience utilisateur c. Wireframes ou maquettes de l'interface utilisateur

Architecture technique : a. Description de l'architecture logicielle b. Description des technologies utilisées (langage de programmation, framework, bibliothèques tierces, etc.) c. Description de l'architecture réseau (serveurs, base de données, etc.)

Sécurité : a. Exigences de sécurité (par exemple, cryptage, authentification, etc.) b. Politique de gestion de la sécurité des données c. Politique de gestion des droits d'accès

Performances : a. Exigences de performances (par exemple, temps de réponse, charge maximale, etc.) b. Politique de sauvegarde et de récupération de données c. Politique de gestion de la mémoire et des ressources système

Intégrations : a. Description des systèmes tiers à intégrer b. Exigences d'interopérabilité c. Protocoles de communication à utiliser

Développement CDc :

Chapitre 1 : definition objectif :

Contexte et objectif du projet application graphique python compatible toutes versions de python qui va nalayser un site web pour créer des statistiques d’achat avec des remonter quotidienne dans un dashboard sous application Flask et a l’aide de tkinkerpour les graphiques,sci-kit learn les autres modules sont a définir selon les besoin évolutif du projet

Questions réponses :

Introduction : a. Contexte et objectif du projet :

Quel est le contexte de votre projet d'application Python ?

b. Portée et périmètre du projet : application personnelle développer sous forme de plusieurs modules qui seront ensuite transformer en application multi os

c. Description du public cible de l'application : seulement moi

Quel est l'objectif principal de votre application ? permettre de remonter des données statistiques qui seront ensuite comparée aux données de différents sites web sur le type d’objet que l’on peut choisir, d’analyser ces données a l’aide d’un module ia (non defini encore) une fois les données remontées et analysée par l’application, la mise a jour du dashboard sous flask avec les données les plus pertinentes pour prévoir l’achat derevente de produit mis en vente aux encheres

Quelles sont les principales fonctionnalités que vous souhaitez inclure dans votre dashboard ?

Un bouton de d’actualisation des données , dans le dashboard , une mise a jour automatique et quotidien, une fonction de installation automatique des paquets manquant ou obsolètes,un dashboard une page dédiée a la verification des modules instalées et des différentes informations, design et travaillé optimisé pour la prise de décision d’achat

Dans quelle mesure votre application sera-t-elle limitée en termes de fonctionnalités et d'utilisation ? c. Description du public cible de l'application : application portable sans installatio

Description dashboard :

Une palette de couleurs cohérente et agréable à regarder, en utilisant des couleurs complémentaires et des contrastes pour mettre en évidence les informations les plus importantes.

Des graphiques clairs et bien structurés, avec des légendes facilement lisibles et des couleurs distinctes pour chaque catégorie de données.

Une disposition claire et organisée, avec des sections distinctes pour chaque type d'informations et un menu de navigation facile à utiliser pour accéder à différentes fonctionnalités.

Des icônes et des illustrations pour faciliter la compréhension des données, en ajoutant des images significatives pour illustrer les graphiques et les tableaux.

Des polices de caractères faciles à lire, avec des tailles différentes pour hiérarchiser l'information et mettre en évidence les points clés.

Des fonctionnalités interactives telles que des filtres pour faciliter la personnalisation des données affichées en fonction des besoins de l'utilisateur.

Une interface responsive qui s'adapte aux différentes tailles d'écrans et permet une utilisation confortable sur tous les types d'appareils.

Des animations subtiles pour améliorer l'expérience utilisateur, en ajoutant des transitions fluides entre les différents éléments de l'interface.

En résumé, un design de dashboard réussi doit être à la fois beau, facile

Architecture technique de l’application

a. Description de l'architecture logicielle : L'architecture logicielle de l'application sera basée sur une architecture client-serveur. Le client sera une application développée à l'aide de Tkinter pour les graphiques, tandis que le serveur sera une application Flask pour la gestion des requêtes HTTP et le stockage des données. b. Description des technologies utilisées : L'application sera développée en Python. Pour le web scraping, nous utiliserons le module BeautifulSoup. Pour le traitement des données, nous utiliserons le module Pandas. Pour la visualisation des données, nous utiliserons Matplotlib et Seaborn pour les graphiques. Enfin, pour l'analyse de données, nous utiliserons scikit-learn. Aucune base de données ne sera utilisée. c. Description de l'architecture réseau : Le serveur sera une application Flask exécutée sur un serveur web. Les données seront stockées dans des fichiers CSV stockés sur le serveur. Les données seront stockées dans des tables relationnelles pour faciliter l'analyse de données.

Sécurité : a. Exigences de sécurité : Les données stockées sur le serveur seront cryptées à l'aide d'un algorithme de cryptage avancé pour garantir leur confidentialité. L'application sera protégée par une authentification basée sur un nom d'utilisateur et un mot de passe. b. Politique de gestion de la sécurité des données : Nous mettrons en place une politique de sauvegarde et de récupération de données pour garantir que les données sont toujours disponibles en cas de panne du serveur ou de tout autre problème technique. Nous surveillerons également régulièrement les journaux de connexion pour détecter toute activité suspecte. c. Politique de gestion des droits d'accès : Nous mettrons en place une politique de gestion des droits d'accès pour garantir que seuls les utilisateurs autorisés ont accès aux données stockées dans l'application. Les droits d'accès seront accordés en fonction du rôle de chaque utilisateur.

Performances : a. Exigences de performances : L'application devra avoir une réponse rapide, avec un temps de réponse maximal de 5 secondes. Elle devra également être capable de gérer un nombre important d'utilisateurs et de requêtes en même temps. b. Politique de sauvegarde et de récupération de données : Nous mettrons en place une politique de sauvegarde régulière des données pour garantir que les données sont toujours disponibles en cas de panne du serveur ou de tout autre problème technique. c. Politique de gestion de la mémoire et des ressources système : Nous surveillerons régulièrement l'utilisation de la mémoire et des ressources système pour garantir que l'application fonctionne de manière efficace et optimale.

Intégrations : a. Description des systèmes tiers à intégrer : Aucun système tiers ne sera intégré pour l'instant, mais cela pourra être envisagé pour les évolutions futures de l'application. b. Exigences d'interopérabilité : L'application sera développée pour être compatible avec toutes les versions de Python. c. Protocoles de communication à utiliser : Aucun protocole de communication n'est requis pour cette version de l'application.

per.py:

scrape\_data(url)

get\_html(url) [sous-fonction]

parse\_html(html) [sous-fonction]

stats.py:

analyze\_data(data)

calculate\_statistics(data) [sous-fonction]

views.py:

home()

data()

get\_data\_from\_file(filename) [sous-fonction]

get\_data\_from\_web(sites) [sous-fonction]

stats()

analyze\_data(data) [sous-fonction]

calculate\_statistics(data) [sous-sous-fonction]

update()

get\_data\_from\_web(sites) [sous-fonction]

save\_data\_to\_file(data, filename) [sous-fonction]

utils.py:

check\_packages(packages)

save\_data(data)

load\_data()

check\_package(package) [sous-fonction]

main.py:

main()

data\_scraper.py:

scrape\_data(url)

get\_html(url) [sous-fonction]

parse\_html(html) [sous-fonction]

data\_parser.py:

parse\_data(raw\_data)

data\_analyzer.py:

analyze\_data(data)

calculate\_statistics(data) [sous-fonction]

data\_visualizer.py:

visualize\_data(data)

plot\_data(data) [sous-fonction]

data\_updater.py:

update\_data(data)

dashboard.py:

create\_dashboard(data)

create\_plot(data) [sous-fonction]

Méthodes:

scraper.py:

scrape\_data(url): récupère le contenu HTML à partir de l'URL spécifiée, le parse en utilisant la fonction parse\_html et retourne un objet Pandas DataFrame contenant les données extraites.

get\_html(url): récupère le contenu HTML à partir de l'URL spécifiée et le retourne sous forme de chaîne de caractères.

parse\_html(html): parse le contenu HTML spécifié en utilisant un parser HTML et retourne un objet Pandas DataFrame contenant les données extraites.

stats.py:

analyze\_data(data): analyse les données passées en paramètre en utilisant la fonction calculate\_statistics et retourne un objet Pandas DataFrame contenant les statistiques calculées.

calculate\_statistics(data): calcule les statistiques à partir des données passées en paramètre et retourne un objet Pandas DataFrame contenant les statistiques calculées.

views.py:

home(): renvoie la page d'accueil de l'application.

data(): récupère les données à partir des sites web enregistrés dans le fichier sites.txt en utilisant la fonction get\_data\_from\_web, ou à partir du fichier CSV de sauvegarde en utilisant la fonction get\_data\_from\_file, et les retourne sous forme d'un objet Pandas DataFrame.

get\_data\_from\_file(filename): charge les données à partir du fichier CSV spécifié et les retourne sous forme d'un objet Pandas DataFrame.

get\_data\_from\_web(sites): récupère les données à partir des sites web spécifiés en utilisant la fonction scrape\_data et les retourne sous forme d'un objet Pandas DataFrame.

stats(): récupère les données en utilisant la fonction data, les analyse en utilisant la fonction analyze\_data et retourne un objet Pandas DataFrame contenant les statistiques calculées. analyze\_data(data): analyse les données passées en paramètre en utilisant la fonction calculate\_statistics et retourne un objet Pandas DataFrame contenant les statistiques calculées. calculate\_statistics(data): calcule les statistiques à partir des données passées en paramètre et retourne un objet Pandas DataFrame contenant les statistiques calculées. update(): récupère les données à partir des sites web enregistrés dans le fichier sites.txt en utilisant la fonction get\_data\_from\_web, met à jour les données en utilisant la fonction save\_data\_to\_file, et retourne un message de confirmation indiquant que les données ont été mises à jour. get\_data\_from\_web(sites): récupère les données à partir des sites web spécifiés en utilisant la fonction scrape\_data et les retourne sous forme d'un objet Pandas DataFrame. save\_data\_to\_file(data, filename): sauvegarde les données passées en paramètre dans un fichier CSV avec le nom spécifié. utils.py:

check\_packages(packages): vérifie si les packages spécifiés sont installés sur le système. save\_data(data): sauvegarde les données passées en paramètre dans un fichier CSV. load\_data(): charge les données à partir d'un fichier CSV et les retourne sous forme d'un objet Pandas DataFrame. check\_package(package): vérifie si le package spécifié est installé sur le système. main.py:

main(): fonction principale qui lance l'application Flask. data\_scraper.py:

scrape\_data(url): récupère le contenu HTML à partir de l'URL spécifiée, le parse en utilisant la fonction parse\_html et retourne une liste de dictionnaires contenant les données extraites. get\_html(url): récupère le contenu HTML à partir de l'URL spécifiée et le retourne sous forme de chaîne de caractères. parse\_html(html): parse le contenu HTML spécifié en utilisant un parser HTML et retourne une liste de dictionnaires contenant les données extraites. data\_parser.py:

parse\_data(raw\_data): convertit la liste de dictionnaires de données brutes passée en paramètre en un objet Pandas DataFrame. data\_analyzer.py:

analyze\_data(data): analyse les données passées en paramètre en utilisant la fonction calculate\_statistics et retourne un objet Pandas DataFrame contenant les statistiques calculées. calculate\_statistics(data): calcule les statistiques à partir des données passées en paramètre et retourne un objet Pandas DataFrame contenant les statistiques calculées. data\_visualizer.py:

visualize\_data(data): affiche les données passées en paramètre sous forme de graphiques en utilisant la fonction plot\_data. plot\_data(data): crée les graphiques à partir des données passées en paramètre et les affiche. data\_updater.py:

update\_data(data): met à jour les données passées en paramètre en utilisant la fonction scrape\_data, la fonction parse\_data et la fonction save\_data.

Details des modules unitaires :

scraper.py:

* scrape\_data(url) : Cette fonction récupère le contenu HTML à partir de l'URL spécifiée, le parse en utilisant la fonction parse\_html et retourne un objet Pandas DataFrame contenant les données extraites.
* get\_html(url) [sous-fonction] : Cette sous-fonction récupère le contenu HTML à partir de l'URL spécifiée et le retourne sous forme de chaîne de caractères.
* parse\_html(html) [sous-fonction] : Cette sous-fonction parse le contenu HTML spécifié en utilisant un parser HTML et retourne un objet Pandas DataFrame contenant les données extraites.

stats.py:

* analyze\_data(data) : Cette fonction analyse les données passées en paramètre en utilisant la fonction calculate\_statistics et retourne un objet Pandas DataFrame contenant les statistiques calculées.
* calculate\_statistics(data) [sous-fonction] : Cette sous-f

n'oublies de conserver selenium et anticaptcha

D'accord, voici la liste mise à jour avec les modules Selenium et AntiCaptcha:

Fonctions:

scraper.py:

* scrape\_data(url): récupère le contenu HTML à partir de l'URL spécifiée en utilisant Selenium et AntiCaptcha, le parse en utilisant la fonction parse\_html et retourne un objet Pandas DataFrame contenant les données extraites.
* get\_html(url): récupère le contenu HTML à partir de l'URL spécifiée en utilisant Selenium et AntiCaptcha et le retourne sous forme de chaîne de caractères.
* parse\_html(html): parse le contenu HTML spécifié en utilisant un parser HTML et retourne un objet Pandas DataFrame contenant les données extraites.

stats.py:

* analyze\_data(data): analyse les données passées en paramètre en utilisant la fonction calculate\_statistics et retourne un objet Pandas DataFrame contenant les statistiques calculées.
* calculate\_statistics(data): calcule les statistiques à partir des données passées en paramètre et retourne un objet Pandas DataFrame contenant les statistiques calculées.

views.py:

* home(): renvoie la page d'accueil de l'application.
* data(): récupère les données à partir des sites web enregistrés dans le fichier sites.txt en utilisant la fonction get\_data\_from\_web, ou à partir du fichier CSV de sauvegarde en utilisant la fonction get\_data\_from\_file, et les retourne sous forme d'un objet Pandas DataFrame.
* get\_data\_from\_file(filename): charge les données à partir du fichier CSV spécifié et les retourne sous forme d'un objet Pandas DataFrame.
* get\_data\_from\_web(sites): récupère les données à partir des sites web spécifiés en utilisant la fonction scrape\_data et les retourne sous forme d'un objet Pandas DataFrame.
* stats(): récupère les données en utilisant la fonction data, les analyse en utilisant la fonction analyze\_data et retourne un objet Pandas DataFrame contenant les statistiques calculées.
* analyze\_data(data): analyse les données passées en paramètre en utilisant la fonction calculate\_statistics et retourne un objet Pandas DataFrame contenant les statistiques calculées.
* calculate\_statistics(data): calcule les statistiques à partir des données passées en paramètre et retourne un objet Pandas DataFrame contenant les statistiques calculées.
* update(): récupère les données à partir des sites web enregistrés dans le fichier sites.txt en utilisant la fonction get\_data\_from\_web, met à jour les données en utilisant la fonction save\_data\_to\_file, et retourne un message de confirmation indiquant que les données ont été mises à jour.
* get\_data\_from\_web(sites): récupère les données à partir des sites web spécifiés en utilisant la fonction scrape\_data et les retourne sous forme d'un objet Pandas DataFrame.
* save\_data\_to\_file(data, filename): sauvegarde les données passées en paramètre dans un fichier CSV avec le nom spécifié.

utils.py:

* check\_packages(packages): vérifie si les packages spécifiés sont installés sur le système.
* save\_data(data): sauvegarde les données passées en paramètre dans un fichier CSV.
* load\_data(): charge les données à partir d'un fichier CSV et les retourne sous forme d'un objet Pandas DataFrame.
* check\_package(package): vérifie si le package spécifié est installé sur le système.

main.py:

* main(): fonction principale qui lance l'application Flask.

data\_scraper.py:

* scrape\_data(url): récupère le contenu HTML à partir de l'URL spécifiée en utilisant Selenium et AntiCaptcha, le parse

modules a déclarer dans les fichiers

modules à ajouter pour chaque fichier :

scraper.py:

* pandas
* selenium
* requests
* json
* anticaptchaofficial

stats.py:

* pandas

views.py:

* pandas
* flask

utils.py:

* pandas

main.py:

* flask
* views
* utils

data\_scraper.py:

* pandas
* selenium
* requests
* json
* anticaptchaofficial

data\_parser.py:

* pandas

data\_analyzer.py:

* pandas

data\_visualizer.py:

* pandas
* matplotlib

data\_updater.py:

* pandas
* selenium
* requests
* json
* anticaptchaofficial

dashboard.py:

* pandas
* dash
* dash\_core\_components

Le système actuel est une application web Python qui permet de récupérer des données à partir de sites web, de les analyser et de les afficher sous forme de graphiques. L'application utilise plusieurs modules Python, tels que Pandas, Flask, Selenium, requests, json, anticaptchaofficial et matplotlib, pour effectuer ces tâches.

La première étape de l'implémentation consiste à mettre à jour le module scraper.py. Cette mise à jour inclut l'ajout de la fonctionnalité d'utilisation de Selenium et AntiCaptcha pour récupérer les données à partir de sites web. En outre, les modules Pandas, selenium, requests, json et anticaptchaofficial seront importés dans le fichier. La fonction scrape\_data récupérera le contenu HTML à partir de l'URL spécifiée en utilisant Selenium et AntiCaptcha, le parse en utilisant la fonction parse\_html et retourne un objet Pandas DataFrame contenant les données extraites. La sous-fonction get\_html récupérera le contenu HTML à partir de l'URL spécifiée en utilisant Selenium et AntiCaptcha et le retourne sous forme de chaîne de caractères. La sous-fonction parse\_html parse le contenu HTML spécifié en utilisant un parser HTML et retourne un objet Pandas DataFrame contenant les données extraites.

La deuxième étape de l'implémentation consiste à mettre à jour le module stats.py. Cette mise à jour inclut l'ajout des modules Pandas et numpy dans le fichier. La fonction analyze\_data analyse les données passées en paramètre en utilisant la fonction calculate\_statistics et retourne un objet Pandas DataFrame contenant les statistiques calculées. La sous-fonction calculate\_statistics calcule les statistiques à partir des données passées en paramètre et retourne un objet Pandas DataFrame contenant les statistiques calculées.

La troisième étape de l'implémentation consiste à mettre à jour le module views.py. Cette mise à jour inclut l'ajout des modules Pandas et Flask dans le fichier. La fonction home renvoie la page d'accueil de l'application. La fonction data récupère les données à partir des sites web enregistrés dans le fichier sites.txt en utilisant la fonction get\_data\_from\_web, ou à partir du fichier CSV de sauvegarde en utilisant la fonction get\_data\_from\_file, et les retourne sous forme d'un objet Pandas DataFrame. La fonction get\_data\_from\_file charge les données à partir du fichier CSV spécifié et les retourne sous forme d'un objet Pandas DataFrame. La fonction get\_data\_from\_web récupère les données à partir des sites web spécifiés en utilisant la fonction scrape\_data et les retourne sous forme d'un objet Pandas DataFrame. La fonction stats récupère les données en utilisant la fonction data, les analyse en utilisant la fonction analyze\_data et retourne un objet Pandas DataFrame contenant les statistiques calculées. La fonction analyze\_data analyse les données passées en paramètre en utilisant la fonction calculate\_statistics et retourne un objet Pandas DataFrame contenant les statistiques calculées. La fonction calculate\_statistics calcule les statistiques à partir des données passées en paramètre et retourne un objet Pandas DataFrame contenant les statistiques calculées. La fonction update récupère les données à partir des sites web enregistrés dans le fichier sites.txt en utilisant la fonction get\_data\_from\_web, met à jour les données en utilisant la fonction save\_data\_to\_file, et

aider à vérifier la cohérence entre les fichiers Python et à confirmer leur bonne structure. Pour cela, je peux parcourir chaque fichier et m'assurer que les noms de variables, de fonctions et de classes sont cohérents avec le cahier des charges. Je peux également vérifier que les imports sont correctement effectués et que les fichiers sont organisés de manière appropriée.

Pour comparer entre chaque module, je peux vérifier que les fonctions, variables et classes sont correctement utilisées et référencées entre les fichiers. Si certains fichiers sont censés interagir ensemble, je peux vérifier que les entrées et les sorties sont correctes.

Cependant, veuillez noter que pour une vérification approfondie, il peut être nécessaire d'examiner en détail chaque fichier et son contenu. Il est également important d'avoir un cahier des charges clair et détaillé pour pouvoir effectuer une vérification complète et précis

mkdir my\_app && cd my\_app && touch scraper.py stats.py views.py utils.py main.py data\_scraper.py data\_parser.py data\_analyzer.py data\_visualizer.py data\_updater.py dashboard.py sites.txt && echo -e "import pandas as pd\nimport requests\nimport json\nimport time\nimport selenium.webdriver as webdriver\nfrom bs4 import BeautifulSoup\nfrom utils import check\_package, save\_data\n\nif check\_package('anticaptchaofficial'):\n import anticaptchaofficial\n\n\ndef scrape\_data(url):\n driver = webdriver.Chrome()\n if 'google.com/recaptcha/' in url:\n captcha\_text = solve\_captcha(driver)\n html = get\_html(url, captcha\_text)\n else:\n html = get\_html(url)\n data = parse\_html(html)\n driver.quit()\n return data\n\n\ndef get\_html(url, captcha\_text=None):\n headers = {\n 'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/58.0.3029.110 Safari/537.36',\n 'Referer': url\n }\n if captcha\_text:\n cookies = requests.get('https://2captcha.com/in.php?key=' + api\_key + '&method=userrecaptcha&googlekey=' + site\_key + '&pageurl=' + url).text\n captcha\_id = cookies.split('|')[1]\n captcha\_answer = requests.get('https://2captcha.com/res.php?key=' + api\_key + '&action=get&id=' + captcha\_id).text\n while 'CAPCHA\_NOT\_READY' in captcha\_answer:\n time.sleep(5)\n captcha\_answer = requests.get('https://2captcha.com/res.php?key=' + api\_key + '&action=get&id=' + captcha\_id).text\n captcha\_token = captcha\_answer.split('|')[1]\n r = requests.get(url + '&g-recaptcha-response=' + captcha\_token, headers=headers, cookies={'2Captcha': captcha\_id})\n return r.text\n else:\n r = requests.get(url, headers=headers)\n return r.text\n\n\ndef parse\_html(html):\n soup = BeautifulSoup(html, 'html.parser')\n table = soup.find('table')\n rows = table.find\_all('tr')\n data = []\n for row in rows:\n cols = row.find\_all('td')\n cols = [ele.text.strip() for ele in cols]\n data.append([ele for ele in cols if ele])\n df = pd.DataFrame(data[1:], columns=data[0])\n return df\n\n\ndef solve\_captcha(driver):\n task = anticaptchaofficial.ImageToTextTask(image\_url=get\_image\_path(driver), \*\*{'clientKey': api\_key})\n captcha\_text = task.solve()\n return captcha\_text\n\n\ndef get\_image\_path(driver):\n driver.get('https://www.google.com/recaptcha/api2/demo')\n frame = driver.find\_element\_by\_xpath('//iframe[contains(@src, "recaptcha")]')\n driver.switch\_to.frame(frame)\n image\_element = driver.find\_element\_by\_xpath('//img[contains(@src, "google.com/recaptcha/api2/p")]')\n image\_url = image\_element.get\_attribute('src')\n return image\_url\n\n" > scraper.py && echo -e "import pandas as pd\n\ndef analyze\_data(data):\n return calculate

&& echo -e "import pandas as pd\n\ndef analyze\_data(data):\n return calculate\_statistics(data)\n\n\ndef calculate\_statistics(data):\n # Perform analysis and return results\n pass\n" > stats.py && echo -e "import pandas as pd\n\ndef view\_data(data):\n # Visualize data\n pass\n" > views.py && echo -e "import pandas as pd\nimport os\n\ndef check\_package(package\_name):\n try:\n \_\_import\_\_(package\_name)\n except ImportError:\n return False\n return True\n\ndef save\_data(data, filename):\n path = os.path.join(os.getcwd(), filename)\n data.to\_csv(path, index=False)\n" > utils.py && echo -e "from scraper import scrape\_data\nfrom data\_parser import parse\_html\nfrom utils import save\_data\n\n# Scraping data\nurl = 'https://www.example.com'\ndata = scrape\_data(url)\n\n# Parsing data\nparsed\_data = parse\_html(data)\n\n# Saving data to CSV file\ndata\_filename = 'data.csv'\nsave\_data(parsed\_data, data\_filename)\n" > main.py && echo -e "from scraper import scrape\_data\nfrom data\_parser import parse\_html\nfrom data\_analyzer import analyze\_data\nfrom data\_visualizer import view\_data\nfrom data\_updater import update\_data\nfrom utils import save\_data\n\n# Scraping data\nurl = 'https://www.example.com'\ndata = scrape\_data(url)\n\n# Parsing data\nparsed\_data = parse\_html(data)\n\n# Analyzing data\ndata\_stats = analyze\_data(parsed\_data)\n\n# Visualizing data\nview\_data(data\_stats)\n" > dashboard.py && echo -e "https://www.example.com" > sites.txt

Donne moi le reste a partir de l’etape 5

Test unitaires

Étape 3 : Conception de l'interface utilisateur

La conception de l'interface utilisateur est une étape importante pour assurer l'expérience utilisateur optimale. Nous allons créer un design simple et intuitif pour faciliter la compréhension des utilisateurs et leur permettre de prendre des décisions éclairées. Nous allons créer des wireframes pour chacune des pages de l'interface utilisateur et les soumettre pour validation aux parties prenantes.

La page d'accueil de l'application sera la première page à s'afficher. Elle comportera un bouton de mise à jour manuelle des données et une option pour lancer la mise à jour automatique. La page d'accueil permettra également aux utilisateurs de filtrer les données selon leurs besoins. La page d'accueil ressemblera à ceci :



La page de visualisation des données sera la deuxième page à s'afficher. Elle permettra aux utilisateurs de visualiser les données collectées sous forme de graphiques et de tableaux de bord. Les graphiques aideront les utilisateurs à comprendre les tendances et à prendre des décisions éclairées. La page de visualisation des données ressemblera à ceci :



La page de filtrage et de tri sera la troisième page à s'afficher. Elle permettra aux utilisateurs de filtrer et de trier les données collectées selon leurs besoins. La page de filtrage et de tri ressemblera à ceci :



Étape 4 : Conception de l'architecture technique

L'architecture technique est une étape importante pour garantir la performance, la sécurité et la qualité de l'application. Nous allons concevoir une architecture client-serveur pour garantir une réponse rapide et une gestion efficace des requêtes.

Le client sera l'interface utilisateur qui sera développée à l'aide de Flask et Tkinter pour les graphiques, tandis que le serveur sera une application Flask pour la gestion des requêtes HTTP et le stockage des données. Nous allons excel pour stocker les données reorganiscollectées.

Le code pour la conception de l'architecture technique ressemblera à ceci :

pythonCopy code

from flask import Flask, render\_template, request import sqlite3 app = Flask(\_\_name\_\_) @app.route('/') def home(): return render\_template('home.html') @app.route('/visualisation') def visualisation(): return render\_template('visualisation.html') @app.route('/filtrage') def filtrage(): return render\_template('filtrage.html') if \_\_name\_\_ == '\_\_main\_\_': app.run(debug=True)

Étape 5 : Conception de la sécurité

La sécurité est une étape importante pour garantir la confidentialité des données collectées. Nous allons mettre en place une politique de sécurité stricte pour garantir que les données collectées sont stockées de manière sécurisée.

Nous allons crypter les données stockées dans la base de données à l'aide d'un algorithme de cryptage avancé pour garantir leur confidentialité. L'application sera également protégée par une authentification basée sur un nom d'utilisateur et un mot de passe.

Nous allons mettre en place une politique de gestion de la sécurité des donnée

Tests unitaires nécessaires :

* Test d'authentification pour s'assurer que l'authentification fonctionne correctement.
* Test de cryptage pour s'assurer que les données stockées dans la base de données sont correctement cryptées.
* Test de gestion des droits d'accès pour s'assurer que les utilisateurs ont les droits appropriés en fonction de leur rôle.

Extrait de code : Voici un exemple de code pour l'authentification :

pythonCopy code

from flask import Flask, request, jsonify, make\_response import jwt import datetime from functools import wraps app = Flask(\_\_name\_\_) app.config['SECRET\_KEY'] = 'thisissecret' def token\_required(f): @wraps(f) def decorated(\*args, \*\*kwargs): token = request.args.get('token') if not token: return jsonify({'message': 'Token is missing!'}), 401 try: data = jwt.decode(token, app.config['SECRET\_KEY']) except: return jsonify({'message': 'Token is invalid!'}), 401 return f(\*args, \*\*kwargs) return decorated @app.route('/unprotected') def unprotected(): return jsonify({'message': 'Anyone can view this!'}) @app.route('/protected') @token\_required def protected(): return jsonify({'message': 'This is only available for people with valid tokens.'})

Ce code utilise un décorateur pour vérifier si l'utilisateur a un jeton valide avant d'autoriser l'accès à la route protégée.

Étape 6 : Conception des performances

Dans cette étape, nous allons concevoir les exigences de performances de l'application.

a. Exigences de performances : L'application devra avoir une réponse rapide, avec un temps de réponse maximal de 5 secondes. Elle devra également être capable de gérer un nombre important d'utilisateurs et de requêtes en même temps.

b. Politique de sauvegarde et de récupération de données : Nous mettrons en place une politique de sauvegarde régulière des données pour garantir que les données sont toujours disponibles en cas de panne du serveur ou de tout autre problème technique.

c. Politique de gestion de la mémoire et des ressources système : Nous surveillerons régulièrement l'utilisation de la mémoire et des ressources système pour garantir que l'application n’est pas trop gourmande en ressources

ce qu’il me reste a faire :

Bien sûr, voici un récapitulatif de ce qu'il vous reste à faire pour chaque fichier afin de rendre votre application opérationnelle en respectant les spécifications techniques et fonctionnelles que vous avez définies :

1. main.py : Il vous reste à ajouter le code pour les fonctionnalités suivantes :

* Bouton de mise à jour des données : permet aux utilisateurs de mettre à jour manuellement les données collectées sur les sites web d'enchères.
* Mise à jour automatique : permet la mise à jour automatique et quotidienne des données collectées sur les sites web d'enchères.
* Installation automatique des paquets manquants ou obsolètes : permet l'installation automatique des paquets nécessaires à l'exécution de l'application qui manquent ou qui sont obsolètes.
* Dashboard : permet la visualisation et l'analyse des données collectées sous forme de graphiques et de tableaux de bord. Elle sera conçue de manière à ce que les utilisateurs puissent facilement comprendre les tendances et prendre des décisions éclairées.
* Fonctionnalités de filtrage et de tri : permet aux utilisateurs de filtrer et de trier les données collectées selon leurs besoins.

Il est important de respecter l'architecture logicielle de l'application et d'utiliser les technologies que vous avez spécifiées dans les spécifications techniques. Vous devrez également mettre en place les mesures de sécurité et de performances que vous avez définies.

1. scraper.py : Le fichier scraper.py doit contenir le code pour extraire les données des sites web d'enchères. Vous devez utiliser BeautifulSoup pour extraire les données des pages web et les stocker dans des fichiers Excel ou CSV. Vous devez utiliser le code existant et l’adapter pour qu’il colle au cahier des chargeVous pouvez utiliser des tests unitaires pour vous assurer que le scraper fonctionne correctement.
2. data\_processing.py : Le fichier data\_processing.py doit contenir le code pour traiter les données extraites par le scraper. Vous pouvez utiliser Pandas pour effectuer des opérations de traitement de données telles que le tri, le filtrage et l'analyse statistique. Vous pouvez également utiliser des tests unitaires pour vous assurer que le code fonctionne correctement.
3. dashboard.py : Le fichier dashboard.py doit contenir le code pour générer les graphiques et les tableaux de bord pour la visualisation des données. Vous pouvez utiliser Matplotlib et Seaborn pour générer des graphiques et des tableaux de bord interactifs. Vous pouvez également utiliser des tests unitaires pour vous assurer que le code fonctionne correctement.
4. update.py : Le fichier update.py doit contenir le code pour mettre à jour les données automatiquement tous les jours. Vous pouvez utiliser une tâche planifiée pour exécuter le script tous les jours et mettre à jour les données. Vous pouvez également utiliser des tests unitaires pour vous assurer que le code fonctionne correctement.
5. install\_packages.py : Le fichier install\_packages.py doit contenir le code pour installer automatiquement les paquets manquants ou obsolètes. Vous pouvez utiliser la commande pip pour installer les paquets nécessaires. Vous pouvez également utiliser des tests unitaires pour vous assurer que le code fonctionne correctement.
6. backup.py : Le fichier backup.py doit contenir le code pour sauvegarder les données collectées et les stocker dans un emplacement sécurisé. Vous pouvez utiliser une base de données SQLite pour stocker les données et les sauvegarder régulièrement. Vous pouvez également utiliser des tests unitaires pour vous assurer que le code fonction
7. Vous avez créé un fichier **scraper.py** contenant les fonctions pour récupérer les données HTML à partir des sites web, les parser, et les stocker dans un objet Pandas DataFrame.
8. Vous avez créé un fichier **stats.py** contenant les fonctions pour analyser les données stockées dans un objet Pandas DataFrame et calculer les statistiques.
9. Vous avez créé un fichier **views.py** contenant les fonctions pour gérer les vues web de votre application Flask et interagir avec les utilisateurs.
10. Vous avez créé un fichier **utils.py** contenant les fonctions utilitaires pour charger et sauvegarder des données dans un fichier CSV.

Il reste à faire :

1. Mettre à jour le fichier **scraper.py** pour ajouter la fonctionnalité d'utilisation de Selenium et AntiCaptcha pour récupérer les données à partir de sites web. En outre, les modules Pandas, selenium, requests, json et anticaptchaofficial seront importés dans le fichier.
2. Mettre à jour le fichier **stats.py** pour ajouter les modules Pandas et numpy dans le fichier.
3. Mettre à jour le fichier **views.py** pour ajouter les modules Pandas et Flask dans le fichier. En outre, ajouter des fonctions pour la gestion des données, telles que la fonction pour récupérer les données à partir des sites web, les stocker dans un fichier CSV et les charger à partir d'un fichier CSV.
4. Ajouter un fichier **dashboard.py** pour afficher les données collectées sous forme de graphiques en utilisant le module Dash.
5. Ajouter des tests unitaires pour chaque module afin de vous assurer que le code fonctionne correctement.
6. Ajouter un fichier **backup.py** pour sauvegarder régulièrement les données collectées et les stocker dans un emplacement sécurisé, comme un cloud de stockage ou un serveur distant.

A vérifier dans chaque fichier

1. main.py
   * Fonction pour le bouton de mise à jour manuelle des données
   * Fonction pour la mise à jour automatique quotidienne des données
   * Fonction pour l'installation automatique des paquets manquants ou obsolètes
   * Fonction pour la visualisation et l'analyse des données collectées sous forme de graphiques et de tableaux de bord
   * Fonction pour le filtrage et le tri des données collectées
   * Appel aux fonctions de scraper.py pour extraire les données des sites web d'enchères
   * Appel aux fonctions de data\_processing.py pour traiter les données extraites
   * Appel aux fonctions de dashboard.py pour générer les graphiques et les tableaux de bord pour la visualisation des données
   * Appel aux fonctions de update.py pour mettre à jour les données automatiquement tous les jours
   * Appel aux fonctions de install\_packages.py pour installer automatiquement les paquets manquants ou obsolètes
   * Appel aux fonctions de backup.py pour sauvegarder les données collectées et les stocker dans un emplacement sécurisé
2. scraper.py
   * Fonction pour extraire les données des sites web d'enchères à l'aide de BeautifulSoup
   * Fonction pour stocker les données extraites dans un objet Pandas DataFrame
   * Fonction pour utiliser Selenium et AntiCaptcha pour récupérer les données à partir de sites web
   * Fonction pour importer les modules Pandas, selenium, requests, json et anticaptchaofficial
3. data\_processing.py ////////////////////////////////////////////////
   * Fonction pour trier, filtrer et analyser les données stockées dans un objet Pandas DataFrame
   * Fonction pour importer les modules Pandas et numpy
4. dashboard.py
   * Fonction pour générer des graphiques et des tableaux de bord interactifs pour la visualisation des données en utilisant Matplotlib et Seaborn
   * Fonction pour importer les modules Pandas, Matplotlib, Seaborn et Dash
5. update.py
   * Fonction pour mettre à jour les données automatiquement tous les jours
   * Fonction pour importer les modules Pandas et apscheduler
6. install\_packages.py
   * Fonction pour installer automatiquement les paquets nécessaires à l'exécution de l'application qui manquent ou qui sont obsolètes en utilisant la commande pip
   * Fonction pour importer le module pip
7. backup.py
   * Fonction pour sauvegarder régulièrement les données collectées et les stocker dans un emplacement sécurisé, comme un cloud de stockage ou un serveur distant
   * Fonction pour importer les modules Pandas, le module excel pour python et datetime

Test unitaire :

Creation du modele d’apprentissage

1. Informations générales sur le produit :

* Nom du produit
* Marque / Constructeur
* Modèle
* Catégorie (ex. ordinateur portable, téléphone portable, tablette, etc.)
* Année de sortie / âge du produit
* Caractéristiques techniques (ex. processeur, RAM, stockage, résolution d'écran, etc.)
* État du produit (neuf, d'occasion, reconditionné, etc.)

1. Informations sur la vente :

* Date de la vente
* Prix de vente
* Coût d'acquisition (si possible)
* Vendeur (si possible)

1. Analyse des données :

* Comparaison des prix de vente avec le coût d'acquisition pour calculer la marge brute
* Analyse de la tendance des prix pour chaque produit
* Analyse de la relation entre le sentiment vis-à-vis du constructeur et le prix de vente
* Analyse de la relation entre le sentiment vis-à-vis du modèle et le prix de vente
* Identification des produits les plus rentables en termes de marge brute
* Identification des produits les plus populaires en termes de volume de ventes
* Identification des tendances du marché (ex. émergence de nouveaux produits, évolution des préférences des acheteurs, etc.)

1. Critères de sélection des produits :

* Quels sont les types de produits à sélectionner pour l'analyse ?
* Comment ces critères seront-ils utilisés pour filtrer les données collectées aux enchères ?
* Comment les données collectées seront-elles nettoyées pour éliminer les doublons et les données inutiles ?

1. Stockage et organisation des données :

* Comment les données collectées seront-elles stockées dans la base de données ?
* Quelle sera la structure de la base de données pour permettre une analyse efficace ?
* Comment les données seront-elles organisées pour faciliter leur récupération et leur analyse ultérieure ?

1. Construction et entraînement du modèle de classification des sentiments :

* Quel sera l'algorithme de machine learning utilisé pour construire le modèle de classification des sentiments des produits ?
* Comment le modèle sera-t-il entraîné pour garantir une précision optimale ?
* Quels seront les critères de performance pour le modèle de classification des sentiments ?

1. Construction et entraînement du modèle de prédiction des prix :

* Quel algorithme de machine learning sera utilisé pour construire le modèle de prédiction des prix ?
* Comment les données seront-elles préparées pour garantir une précision optimale ?
* Quels seront les critères de performance pour le modèle de prédiction des prix ?

1. Évaluation et amélioration des modèles d'apprentissage automatique :

* Comment les résultats des modèles seront-ils évalués pour déterminer leur précision ?
* Comment les modèles seront-ils améliorés pour garantir une précision optimale ?
* Quels seront les critères de performance pour les modèles d'apprentissage automatique ?

1. Accès aux résultats de l'analyse et des prévisions de prix :

* Comment les utilisateurs pourront-ils accéder aux résultats de l'analyse et des prévisions de prix ?
* Comment les résultats seront-ils présentés pour permettre une analyse efficace ?

1. Mises à jour des données collectées :

* Comment les mises à jour des données collectées seront-elles gérées ?
* À quelle fréquence les données seront-elles mises à jour ?
* Comment les données mises à jour seront-elles intégrées dans les modèles d'apprentissage automatique ?

1. Fonctionnalités de filtrage et de tri des données :

* Comment les fonctionnalités de filtrage et de tri des données seront-elles implémentées dans l'interface utilisateur ?
* Quels seront les critères de filtrage et de tri disponibles pour les utilisateurs ?
* Comment les résultats seront-ils présentés pour permettre une analyse efficace ?

1. Éléments de sécurité :

* Comment les données collectées et les modèles de machine learning seront-ils protégés ?
* Quelles seront les mesures de sécurité mises en place pour garantir la confidentialité des données collectées ?

1. Intégration des modèles de machine learning :

* Comment les modèles de machine learning seront-ils intégrés dans l'application Flask existante ?
* Quels seront les éléments à prendre en compte pour garantir une intégration efficace ?
* Comment les résultats des modèles seront-ils présentés dans l'interface utilisateur ?