Projet final Tkinter

Anri Kennel* (L2-A)
Module Programmation d'interfaces · Paris 8

Année universitaire 2021-2022

Table des matières

Les explications sont en commentaire du code.

*Numéro d'étudiant : 20010664

1 Consigne

Ici ce trouve le cahier des charges du programme. Toutes les améliorations, apportés au programme sont rangés à côtés du champs correspondant, en gras.

Pour les éléments ajoutés au programme qui ne rentre dans aucune cases, il y a une catégorie "À savoir" à la fin du cahier des charges qui les précise. Il y a aussi des informations complémentaire par rapport au projet.

1.1 Dépendances

Les modules externes utilisés sont :

- `tkinter` pour la GUI
  - `.ttk` pour la liste déroulante et les lignes qui séparent les cases du tableau
  - `.messagebox` pour les messages pop-up
  - `.filedialog` pour la boîte de dialogue du fichier
- `sqlite3` pour la base de donnée SQLite
- `datetime` pour la date
- `re` pour le regex
- `csv` pour la gestion du fichier `CSV`
- `random` pour la génération du stock (prix et quantité)

1.2 Cahier des charges

- ☑ Page de login `/1.5`
  - ☑ Nom d'utilisateur ne contient que des lettres et des chiffres
  - ☑ Mot de passe de minimum 8 caractères dont 1 caractère spécial, une majuscule et une minuscule ⇒ **possibilité d'afficher ou non le mot de passe en clair**
  - ☑ Un bouton de connexion ⇒ **possibilité aussi d'utiliser la touche** `Entrer` ***(pour aller plus vite)*** **qui permet de se rendre sur l'interface Caissier ou Manager**
  - ☑ Un bouton pour quitter l'application
- ☑ Page de manager (définit par un nom d'utilisateur et un mot de passe) `/7.5`
  - ☒ Peut ajouter et supprimer un caissier ⇒ **lisiblité accru pour les champs mal renseignés, l'ID n'est pas à renseigné car assigné automatiquent par la base de donnée**
  - ☒ Peut voir la liste des caissiers ⇒ **possibilité d'ouvrir des informations étendues sur un utilisateur, ainsi que de filtrer les utilisateurs (manager et caissiers) mais impossible de tout déselectionner (caissier par défaut)**
  - ☑ Un histogramme présentant l'évolution des sommes totales des ventes journalières de la semaine passée d'un utilisateur ⇒ **accessible au double-clique dans la fenêtre des informations étendues d'un utilisateur**
  - ☑ Un bouton pour vider tous les champs de saisie
  - ☒ Un bouton pour quitter l'application ⇒ **j'ai préféré mettre un bouton pour se déconnecter**
  - ☑ Un bouton pour se mettre en "mode caissier"
- ☑ Page de caissier (définit par un identifiant, un nom d'utilisateur, un mot de passe, un nom, un prenom, une date de naissance, une adresse et un code postal) `/6`
  - ☑ Afficher le stock disponible
    - ☒ 4 rayons de chacun au moins 10 articles de votre choix (fruits/légumes, boulangerie, boucherie/poissonnerie ou produits d'entretien) ⇒ **toutes les images sont aux dimensions 50x50 et ont étés converties avec le logiciel Gimp**
    - ☒ Au clic sur le produit, l'identifiant, le nom, la quantité en stock et le prix s'affichent ⇒ **tout est affiché directement, pas besoin de cliquer sur le produit, il y a aussi un système de pages pour une meilleur lisibilité (10 éléments par page au maximum)**

- ☑ Possibilité de rajouter des produits en stock
- ☑ Affichage d'un ticket de caisse
  - ☑ Date de vente
  - ☑ ID, nom, quantité, prix des produits achetés
  - ☑ Prix total
  - ☑ Un bouton pour valider
- ☑ Interface d'export des statistiques (stock le montant total de vente par jour) ⇒ **export au format** `CSV`

Avec à savoir :

☑ Ergonomie /2
- ☒ Utilisation de `Frame` et peu de `TopLevel`, ainsi qu'une seule fenêtre `Tk` pour éviter de multiples ouverture/fermeture de fenêtre durant l'utilisation de l'application

☑ Utilisateurs stockés dans la base de donnée /2
- ☒ Possibilité de recréer la base de donnée automatiquement si elle n'existe plus
- ☒ Utilisation, en plus de `SQLite`, d'un fichier `CSV` pour exporter les statistiques des caissiers, et ainsi pouvoir traiter ces informations dans un tableur (outil externe) à l'avenir

☒ Ajout d'autres fonctionnalités /1
- ☒ J'ai pas vraiment ajouter une toute nouvelle fonctionnalité, mais améliorer ce qui était demandé pour une plus grande souplesse à l'utilisation (cf. les cases cochés avec des ✘)

☒ Lisibilité du code
- ☒ Toutes les fonctions sont commentés et typés (quand possible car j'utilises Python `3.9.7`)
- ☒ Tous le code est dans une classe et non directement dans le code (donc aucune variable globale)
- ☒ Plusieurs fichiers pour une meilleur lisibilité

☒ Affichage sous forme de tableau
- ☒ J'ai évité d'utiliser le widget `Treeview` du module `ttk` de tkinter car je le trouve que peu pratique/flexible (exemple : impossibilité de mettre des images dans les colonnes du tableau) alors j'ai développé une alternative (cf. l'affiche du stock avec un système de page)

2 Code

2.1 `main.py`, fichier principal

```python
# Tkinter
from tkinter import Canvas, IntVar, Checkbutton, LabelFrame, PhotoImage, Scrollbar, Listbox, Entry, Button, Label, Frame, Tk, Toplevel
from tkinter.ttk import Combobox, Separator
from tkinter.messagebox import showerror, showinfo, showwarning, askyesno
from tkinter.filedialog import askopenfile, asksaveasfile
# Regex
from re import sub
# Date
from datetime import date

# Import des fichiers pour gérer la base de donnée et l'export en CSV
from users import Utilisateurs
from stock import Stock
from stats import Stats

class GesMag:
    """Programme de Gestion d'une caise de magasin."""
    def __init__(self, presentation: bool = False) -> None:
        """Instancie quelques variables pour plus de clareté."""
        Utilisateurs().creationTable(presentation) # on créer la table utilisateurs si elle n'existe pas déjà
        Stock().creationTable(presentation) # on créer la table du stock si elle n'existe pas déjà
        Stats().creationCSV() # on créer le fichier CSV qui stockera les statistiques des utilisateurs

        self.nomApp = "GesMag" # nom de l'application
        self.parent = Tk() # fenêtre affiché à l'utilisateur
        self.parent.resizable(False, False) # empêche la fenêtre d'être redimensionnée
        self.f = Frame(self.parent) # `Frame` "principale" affiché à l'écran
        self.tableau = Frame() # `Frame` qui va afficher le tableau des éléments présents dans le stock
        self.imagesStock = [] # liste qui va contenir nos images pour l'affichage du stock
        self.dossierImage = PhotoImage(file = "img/dossier.gif") # image pour l'icone de selection
        self.panierAffichage = Frame() # `Frame` qui va afficher le panier
        self.panier = [] # liste des éléments "dans le panier"

    def demarrer(self) -> None:
        """Lance le programme GesMag."""
        self.font = ("Comfortaa", 14) # police par défaut

        self._interfaceConnexion() # on créer la variable `self.f` qui est la frame a affiché
        self.f.grid() # on affiche la frame

        self.parent.mainloop() # on affiche la fenêtre

    def motDePasseCorrect(self, motDPasse: str) -> tuple:
        """Détermine si un mot de passe suit la politique du programme ou non."""
        if len(motDPasse) == 0: # si le champs est vide
            return (False, "Mot de passe incorrect.")
        if len(motDPasse) < 8: # si le mot de passe est plus petit que 8 caractères
            return (False, "Un mot de passe doit faire 8 caractères minimum.")
        """
        - Pour le regex, la fonction `sub` élimine tout ce qui est donné en fonction
          du pattern renseigné, alors si la fonction `sub` renvoie pas exactement
          la même chaîne de charactère alors c'est qu'il y avait un charactère interdit.
        - J'utilises pas `match` parce que je suis plus à l'aise avec `sub`.
        """
        if not sub(r"[A-Z]", '', motDPasse) != motDPasse:
            return (False, "Un mot de passe doit au moins contenir une lettre majuscule.")
        if not sub(r"[a-z]", '', motDPasse) != motDPasse:
            return (False, "Un mot de passe doit au moins contenir une lettre minuscule.")
        if not sub(r" *?[^\w\s]+", '', motDPasse) != motDPasse:
            return (False, "Un mot de passe doit au moins contenir un caractère spécial.")

        return (True,) # si aucun des tests précédents n'est valide, alors le mot de passe est valide

    def utilisateurCorrect(self, utilisateur: str) -> tuple:
        """Détermine si un nom d'utilisateur suit la politique du programme ou non."""
        """
        Pour le nom d'utilisateur on vérifie si le champs n'est pas vide
        et si il y a bien que des lettres et des chiffres.
        """
        if len(utilisateur) == 0:
            return (False, "Utilisateur incorrect.")
        if sub(r" *?[^\w\s]+", '', utilisateur) != utilisateur:
            return (False, "Un nom d'utilisateur ne doit pas contenir de caractère spécial.")
        return (True,)

```

```
76      def nomCorrect(self, nom: str) -> bool:
77          """Détermine si un nom suit la politique du programme ou non."""
78          if len(nom) == 0:
79              return False
80          if sub(r" *?[^\w\s]+", '', nom) != nom: # pas de caractères spéciaux dans un nom
81              return False
82          return True
83
84      def prenomCorrect(self, prenom: str) -> bool:
85          """Détermine si un prénom suit la politique du programme ou non."""
86          if len(prenom) == 0:
87              return False
88          if sub(r" *?[^\w\s]+", '', prenom) != prenom: # pas de caractères spéciaux dans un prénom
89              return False
90          return True
91
92      def naissanceCorrect(self, naissance: str) -> bool:
93          """Détermine si une date de naissance suit la politique du programme ou non."""
94          if len(naissance) == 0:
95              return False
96          # lien pour mieux comprendre ce qui se passe : https://www.debuggex.com/r/hSD-6BfSqDD1It5Z
97          if sub(r"[0-9]{4}\/(0[1-9]|1[0-2])\/(0[1-9]|[1-2][0-9]|3[0-1])", '', naissance) != '':
98              return False
99          return True
100
101     def adresseCorrect(self, adresse: str) -> bool:
102         """Détermine si une adresse suit la politique du programme ou non."""
103         if len(adresse) == 0:
104             return False
105         return True
106
107     def postalCorrect(self, code: str) -> bool:
108         """Détermine si un code postal suit la politique du programme ou non."""
109         if len(code) == 0:
110             return False
111         if sub(r"\d{5}", '', code) != '':
112             return False
113         return True
114
115     def connexion(self, utilisateur: str, motDePasse: str):
116         """Gère la connexion aux différentes interfaces de l'application."""
117         """
118         Vérification nom d'utilisateur / mot de passe correctement entré
119         avec leurs fonctions respectives.
120         """
121         pseudoOk = self.utilisateurCorrect(utilisateur)
122         if not pseudoOk[0]:
123             showerror("Erreur",  pseudoOk[1])
124             return
125         mdpOk = self.motDePasseCorrect(motDePasse)
126         if not mdpOk[0]:
127             showerror("Erreur", mdpOk[1])
128             return
129
130         # Redirection vers la bonne interface
131         utilisateurBaseDeDonnee = Utilisateurs().verificationIdentifiants(utilisateur, motDePasse)
132         if utilisateurBaseDeDonnee[0] > 0:
133             if utilisateurBaseDeDonnee[1] == 0: # si le métier est "Manager"
134                 self._interfaceManager(utilisateurBaseDeDonnee[0])
135             elif utilisateurBaseDeDonnee[1] == 1: # si le métier est "Caissier"
136                 self._interfaceCaissier(utilisateurBaseDeDonnee[0])
137             else:
138                 showerror("Erreur", "Une erreur est survenue : métier inconnue.")
139         else:
140             showerror("Erreur", "Utilisateur ou mot de passe incorrect.")
141
142     def dimensionsFenetre(self, fenetre, nouveauX: int, nouveauY: int):
143         """Permet de changer les dimensions de la fenêtre parent et la place au centre de l'écran."""
144         largeur = fenetre.winfo_screenwidth()
145         hauteur = fenetre.winfo_screenheight()
146
147         x = (largeur // 2) - (nouveauX // 2)
148         y = (hauteur // 2) - (nouveauY // 2)
149
150         fenetre.geometry(f"{nouveauX}x{nouveauY}+{x}+{y}")
151
152     def _interfaceConnexion(self):
153         """Affiche l'interface de connexion."""
154         # Paramètres de la fenêtre
155         self.dimensionsFenetre(self.parent, 400, 600)
156         self.parent.title(f"Fenêtre de connexion  {self.nomApp}")
157
158         # Suppresssion de la dernière Frame
159         self.f.destroy()
160         # Instanciation d'une nouvelle Frame, on va donc ajouter tout nos widgets à cet Frame
```

```
161             self.f = Frame(self.parent)
162             self.f.grid()
163 
164             # Affichage des labels et boutons
165             tentativeDeConnexion = lambda _ = None: self.connexion(utilisateur.get(), motDpasse.get()) #
        lambda pour envoyer les informations entrés dans le formulaire
166             ecart = 80 # écart pour avoir un affichage centré
167             Label(self.f).grid(row=0, pady=50) # utilisé pour du padding (meilleur affichage)
168 
169             Label(self.f, text="Utilisateur", font=self.font).grid(column=0, row=1, columnspan=2, padx=
        ecart - 20, pady=20, sticky='w')
170             utilisateur = Entry(self.f, font=self.font, width=18)
171             utilisateur.grid(column=1, row=2, columnspan=2, padx=ecart)
172 
173             Label(self.f, text="Mot de passe", font=self.font).grid(column=0, row=3, columnspan=2, padx=
        ecart - 20, pady=20, sticky='w')
174             motDpasse = Entry(self.f, font=self.font, show='', width=18)
175             motDpasse.grid(column=1, row=4, columnspan=2, padx=ecart)
176             motDpasse.bind("<Return>", tentativeDeConnexion)
177 
178             def __afficherMDP(self):
179                 """Permet de gérer l'affichage du mot de passe dans le champs sur la page de connexion.
        """
180                 if self.mdpVisible == False: # si mot de passe caché, alors on l'affiche
181                     self.mdpVisible = True
182                     motDpasse.config(show='')
183                     bouttonAffichageMDP.config(font=("Arial", 10, "overstrike"))
184 
185                 else: # inversement
186                     self.mdpVisible = False
187                     motDpasse.config(show='')
188                     bouttonAffichageMDP.config(font=("Arial", 10))
189 
190             bouttonAffichageMDP = Button(self.f, text='', command=lambda: __afficherMDP(self))
191             bouttonAffichageMDP.grid(column=2, row=4, columnspan=2)
192             self.mdpVisible = False
193 
194             bouton = Button(self.f, text="Se connecter", font=self.font, command=tentativeDeConnexion)
195             bouton.grid(column=0, row=5, columnspan=3, padx=ecart, pady=20)
196             bouton.bind("<Return>", tentativeDeConnexion)
197 
198             Button(self.f, text="Quitter", font=self.font, command=quit).grid(column=0, row=6, columnspan
        =4, pady=20)
199 
200         def _interfaceCaissier(self, id: int):
201             """Affiche l'interface du caissier."""
202             caissier = Utilisateurs().recuperationUtilisateur(id=id)
203             self.parent.title(f"Caissier {caissier['nom']} {caissier['prenom']}  {self.nomApp}")
204             self.dimensionsFenetre(self.parent, 1160, 710)
205 
206             self.panier = [] # remet le panier à 0
207 
208             # Suppresssion de la dernière Frame
209             self.f.destroy()
210             # Instanciation d'une nouvelle Frame, on va donc ajouter tout nos widgets à cet Frame
211             self.f = Frame(self.parent)
212             self.f.grid()
213 
214             Label(self.f, text="Interface Caissier", font=(self.font[0], 20)).grid(column=0, row=0) #
        titre de l'interface
215 
216             def __formatPrix(prix: str) -> str:
217                 """
218                 Renvoie un string pour un meilleur affichage du prix :
219                     - ',' au lieu de '.'
220                     - Symbole ''
221                     - 2 chiffres après la virgule
222                 """
223                 return f"{float(prix):.2f} ".replace('.', ',')
224 
225             # -> Partie affichage du Stock
226             stock = LabelFrame(self.f, text="Stock")
227             stock.grid(column=0, row=1, sticky='n', padx=5)
228 
229             # Variables pour les filtres du tableau
230             stockDisponibleVerif = IntVar(stock) # controle si on affiche que les éléments en stocks ou
        non
231             # Cache un certain type de produit
232             fruitsLegumesVerif = IntVar(stock)
233             boulangerieVerif = IntVar(stock)
234             boucheriePoissonnerieVerif = IntVar(stock)
235             entretienVerif = IntVar(stock)
236 
237             def __affichageTableau(page: int = 1):
```

```
238             """Fonction qui va actualiser le tableau avec une page donnée (par défaut affiche la
        première page)."""
239             # On supprime et refais la frame qui va stocker notre tableau
240             self.tableau.destroy()
241             self.tableau = Frame(stock)
242             self.tableau.grid(column=0, row=1, columnspan=7)
243 
244             # Filtre pour le tableau
245             filtres = Frame(stock) # Morceau qui va contenir nos checkbutton
246             ecartFiltre = 10 # écart entre les champs des filtres
247             Label(filtres, text="Filtre", font=self.font).grid(column=0, row=0) # titre
248             Checkbutton(filtres, text="Stock disponible\nuniquement", variable=stockDisponibleVerif,
        command=__affichageTableau).grid(sticky='w', pady=ecartFiltre)
249             Checkbutton(filtres, text="Cacher les\nfruits & légumes", variable=fruitsLegumesVerif,
        command=__affichageTableau).grid(sticky='w', pady=ecartFiltre)
250             Checkbutton(filtres, text="Cacher les produits de\nla boulangerie", variable=
        boulangerieVerif, command=__affichageTableau).grid(sticky='w', pady=ecartFiltre)
251             Checkbutton(filtres, text="Cacher les produits de\nla boucherie\net poissonnerie",
        variable=boucheriePoissonnerieVerif, command=__affichageTableau).grid(sticky='w')
252             Checkbutton(filtres, text="Cacher les produits\nd'entretien", variable=entretienVerif,
        command=__affichageTableau).grid(sticky='w', pady=ecartFiltre)
253             filtres.grid(column=7, row=1, sticky='w')
254 
255             stockListe = Stock().listeStocks() # stock récupéré de la base de données
256 
257             def ___miseAJourPanier(element: dict, action: bool):
258                 """
259                 Permet d'ajouter ou de retirer des éléments au panier
260                 -> Action
261                     -> Vrai : Ajout
262                     -> Faux : Retire
263                 """
264                 # On compte combien de fois l'élément est présent dans le panier
265                 nombreDeFoisPresentDansLePanier = 0
266                 index = None
267                 for idx, elementDansLePanier in enumerate(self.panier):
268                     if elementDansLePanier[0] == element:
269                         index = idx # On met à jour l'index
270                         nombreDeFoisPresentDansLePanier = elementDansLePanier[1]
271                         break # on peut quitter la boucle car on a trouvé notre élément
272 
273                 # On vérifie que on peut encore l'ajouter/retirer
274                 if nombreDeFoisPresentDansLePanier == 0 and not action: # pop-up seulement si on veut
         retirer un élément pas présent
275                     showerror("Erreur", "Impossible de retirer cet élément au panier.\nNon présent
        dans le panier.")
276                     return
277                 if nombreDeFoisPresentDansLePanier >= element["quantite"] and action: # pop-up
        seulement si on veut en rajouter
278                     showerror("Erreur", "Impossible de rajouter cet élément au panier.\nLimite
        excédée.")
279                     return
280 
281                 if index != None: # on retire l'ancienne valeur du panier si déjà présente dans le
        panier
282                     self.panier.pop(index)
283                 else: # sinon on définie un index pour pouvoir ajouté la nouvelle valeur à la fin de
        la liste
284                     index = len(self.panier)
285 
286                 # On change la valeur dans le panier
287                 if action: # si on ajoute
288                     nombreDeFoisPresentDansLePanier += 1
289                 else: # si on retire
290                     nombreDeFoisPresentDansLePanier -= 1
291 
292                 # On rajoute l'élément avec sa nouvelle quantité seulement s'il y en a
293                 if nombreDeFoisPresentDansLePanier > 0:
294                     self.panier.insert(index, (element, nombreDeFoisPresentDansLePanier))
295 
296                 __affichagePanier() # met-à-jour le panier
297 
298             for i in range(0, len(stockListe)): # on retire les éléments plus présent dans la liste
299                 if stockDisponibleVerif.get() == 1 and stockListe[i]["quantite"] < 1:
300                     stockListe[i] = None
301                 elif fruitsLegumesVerif.get() == 1 and stockListe[i]["type"] == "fruits legumes":
302                     stockListe[i] = None
303                 elif boulangerieVerif.get() == 1 and stockListe[i]["type"] == "boulangerie":
304                     stockListe[i] = None
305                 elif boucheriePoissonnerieVerif.get() == 1 and stockListe[i]["type"] == "boucherie
        poissonnerie":
306                     stockListe[i] = None
307                 elif entretienVerif.get() == 1 and stockListe[i]["type"] == "entretien":
308                     stockListe[i] = None
309 
```

```
            # Supprime toutes les valeurs ‘None‘ de la liste
            stockListe = list(filter(None, stockListe))

            ecart = 10 # écart entre les champs

            elementsParPage = 10 # on définit combien d’élément une page peut afficher au maximum
            pageMax = -(-len(stockListe) // elementsParPage) # on définit combien de page il y au maximum

            if pageMax <= 1:
                page = 1 # on force la page à être à 1 si il n’y a qu’une page, peut importe l’argument donnée à la fonction

            limiteIndex = elementsParPage * page # on définit une limite pour ne pas afficher plus d’éléments qu’il n’en faut par page
            if len(stockListe) > 0: # si stock non vide
                # Définition des colonnes
                Label(self.tableau, text="ID").grid(column=0, row=0, padx=ecart)
                Label(self.tableau, text="Image").grid(column=1, row=0, padx=ecart)
                Label(self.tableau, text="Type").grid(column=2, row=0, padx=ecart)
                Label(self.tableau, text="Nom").grid(column=3, row=0, padx=ecart)
                Label(self.tableau, text="Quantité").grid(column=4, row=0, padx=ecart)
                Label(self.tableau, text="Prix unité").grid(column=5, row=0, padx=ecart)
                Label(self.tableau, text="Action").grid(column=6, row=0, padx=ecart)
                Separator(self.tableau).grid(column=0, row=0, columnspan=7, sticky="sew")
                Separator(self.tableau).grid(column=0, row=0, columnspan=7, sticky="new")
                for j in range(0, 8):
                    Separator(self.tableau, orient=’vertical’).grid(column=j, row=0, columnspan=2, sticky="nsw")

                curseur = limiteIndex - elementsParPage # on commence à partir du curseur
                i = 1 # on commence à 1 car il y a déjà le nom des colonnes en position 0
                self.imagesStock = [] # on vide la liste si elle contient déjà des images
                for element in stockListe[curseur:limiteIndex]: # on ignore les éléments avant le curseur et après la limite
                    Label(self.tableau, text=element["id"]).grid(column=0, row=i, padx=ecart)

                    """
                    L’idée est que on a une liste ‘images‘ qui permet de stocker toutes nos images
                    (c’est une limitation de tkinter que de garder nos images en mémoire)
                    Une fois ajouté à la liste, on l’affiche dans notre Label
                    """
                    if Stock().fichierExiste(element["image_url"]): # si l’image existe, utilisation de la fonction de ‘db.py‘
                        self.imagesStock.append(PhotoImage(file = element["image_url"]))
                    else: # si l’image n’existe pas
                        self.imagesStock.append(PhotoImage(file = "img/defaut.gif")) # image par défaut
                    Label(self.tableau, image=self.imagesStock[i - 1]).grid(column=1, row=i, padx=ecart)

                    Label(self.tableau, text=element["type"].capitalize()).grid(column=2, row=i, padx=ecart)
                    Label(self.tableau, text=element["nom"].capitalize()).grid(column=3, row=i, padx=ecart)
                    Label(self.tableau, text=element["quantite"]).grid(column=4, row=i, padx=ecart)
                    Label(self.tableau, text=__formatPrix(element["prix"])).grid(column=5, row=i, padx=ecart)
                    # boutons d’actions pour le panier
                    Button(self.tableau, text=’+’, font=("Arial", 7, "bold"), command=lambda e = element: ___miseAJourPanier(e, True)).grid(column=6, row=i, sticky=’n’, padx=ecart)
                    Button(self.tableau, text=’’, font=("Arial", 7, "bold"), command=lambda e = element: ___miseAJourPanier(e, False)).grid(column=6, row=i, sticky=’s’, pady=2)
                    for j in range(0, 8):
                        Separator(self.tableau, orient=’vertical’).grid(column=j, row=i, columnspan=2, sticky="nsw")
                        Separator(self.tableau).grid(column=j, row=i, columnspan=2, sticky="sew")
                    curseur += 1
                    i += 1

                # Information sur la page actuelle
                Label(self.tableau, text=f"Page {page}/{pageMax}").grid(column=2, row=i, columnspan=3)

                # Boutons
                precedent = Button(self.tableau, text="Page précédente", command=lambda: __affichageTableau(page - 1))
                precedent.grid(column=0, row=i, columnspan=2, sticky=’w’, padx=ecart, pady=ecart)
                suivant = Button(self.tableau, text="Page suivante", command=lambda: __affichageTableau(page + 1))
                suivant.grid(column=5, row=i, columnspan=2, sticky=’e’, padx=ecart)
                if page == 1: # si on est a la première page on désactive le boutton précédent
                    precedent.config(state="disabled")
                if page == pageMax: # si on est a la dernière page on désactive le boutton suivant
                    suivant.config(state="disabled")
```

```
            else:
                Label(self.tableau, text="Il n'y a rien en stock\nEssayez de réduire les critères dans le filtre.").grid(column=0, row=0, columnspan=7)

        __affichageTableau() # affichage du tableau

        # -> Partie affichage du ticket de caisse
        ecart = 10
        ticket = LabelFrame(self.f, text="Ticket de caisse")
        ticket.grid(column=1, row=1, sticky='n', padx=5)

        Label(ticket, text=f"Date de vente : {date.today().strftime('%Y/%m/%d')}").grid(column=0, row=0, pady=ecart)

        def __affichagePanier():
            """Affiche le panier actuel dans le ticket de caisse."""
            self.panierAffichage.destroy()
            self.panierAffichage = Frame(ticket)
            self.panierAffichage.grid(column=0, row=1, pady=ecart)
            elementsAchetes = Label(self.panierAffichage)
            elementsAchetes.grid(column=0, columnspan=2)
            prixTotal = 0
            compteurElements = 0
            for idx, element in enumerate(self.panier):
                Label(self.panierAffichage, text=f"[{element[0]['id']}] -").grid(column=0, row=idx + 1, sticky='e')
                if element[1] > 1:
                    message = f"{element[1]}x {element[0]['nom'].capitalize()} ({__formatPrix(element[0]['prix'])} | total: {__formatPrix(element[0]['prix'] * element[1])})"
                else:
                    message = f"{element[1]}x {element[0]['nom'].capitalize()} ({__formatPrix(element[0]['prix'])})"
                Label(self.panierAffichage, text=message).grid(column=1, row=idx + 1, sticky='w')
                prixTotal += (element[0]["prix"] * element[1]) # ajout du prix
                compteurElements += element[1]

            elementsAchetes.config(text=f"Élément{'s' if compteurElements > 1 else ''} acheté{'s' if compteurElements > 1 else ''} ({compteurElements}) :")

            try: # désactive le bouton si rien n'est dans le panier
                if len(self.panier) <= 0:
                    validationTicketDeCaisseBouton.config(state="disabled")
                else:
                    validationTicketDeCaisseBouton.config(state="active")
            except NameError: # si pas renseigné, alors = panier vide, déjà désactiver
                pass

            Label(self.panierAffichage, text=f"Prix total : {__formatPrix(prixTotal)}").grid(column=0, pady=ecart, columnspan=2)

        __affichagePanier()

        def __validationTicketDeCaisse():
            """Lance plusieurs méthodes pour valider le ticket de caisse."""
            # Met à jour la valeur dans le fichier 'CSV' (statistiques)
            Stats().miseAJourStatsUtilisateur(id, sum([element[0]["prix"] * element[1] for element in self.panier]))

            # Informe l'utilisateur que tout est validé
            showinfo("Validation", "Ticket de caisse validé !")

            # Retire les éléments renseigné dans le panier du stock
            for element in self.panier:
                Stock().reduitQuantiteStock(element[0]["id"], element[1])

            # Remet le panier à 0
            self.panier = []

            # Met-à-jour le panier et le tableau du stock
            __affichagePanier()
            __affichageTableau()

        validationTicketDeCaisseBouton = Button(ticket, text="Valider le\nticket de caisse", font=self.font, command=__validationTicketDeCaisse, state="disabled")
        validationTicketDeCaisseBouton.grid(column=0, pady=ecart)

        # -> Partie ajout élément au stock
        def __ajouterElementStock():
            """Ouvre une fenêtre qui permet d'ajouter un nouvel élément à la base de donnée."""
            """
            L'enfant ('TopLevel') dépend de la 'Frame' et non du parent ('Tk')
            pour éviter de resté ouverte meme lorsque le caissier se déconnecte.
            """
            enfant = Toplevel(self.f)
            enfant.title(f"Ajouter un élément au stock  {self.nomApp}")
```

```
454
455             def ___verification():
456                 """Vérifie si les champs renseignées sont valides."""
457                 """
458                 La variable ‘ok‘ sert à savoir si la vérification est passée
459                 si elle vaut ‘True‘ alors tout est bon,
460                 Par contre si elle vaut ‘False‘ alors il y a eu une erreur.
461                 Les valeurs ‘Entry‘ qui ne sont pas passés seront dans
462                 la liste ‘mauvaisChamps‘.
463                 """
464                 ok = True
465                 mauvaisChamps = []
466                 # vérification pour l'image, on utilise la fonction du fichier ‘db.py‘
467                 if Stock().fichierExiste(image.get()) == False:
468                     ok = False
469                     mauvaisChamps.append(image)
470                 # vérification pour le type
471                 if typeElement.get() not in Stock().listeTypes():
472                     ok = False
473                     # Pas de coloration orange si le type est mauvais parce que on ne peut pas changé
        la couleur de fond d'une ComboBox
474                 # vérification pour le nom
475                 def ___nomValide(nom: str) -> bool:
476                     if len(nom) <= 0:
477                         return False
478                     if Stock().stockExistant(nom) == True:
479                         return False
480                     return True
481                 if ___nomValide(nom.get()) == False:
482                     ok = False
483                     mauvaisChamps.append(nom)
484                 # vérification pour la quantité
485                 try:
486                     int(quantite.get()) # conversion en int
487                 except ValueError: # si la conversion a échoué
488                     ok = False
489                     mauvaisChamps.append(quantite)
490                 # vérification pour le prix
491                 try:
492                     float(prix.get()) # conversion en float
493                 except ValueError: # si la conversion a échoué
494                     ok = False
495                     mauvaisChamps.append(prix)
496
497                 if ok == False:
498                     """
499                     Tous les champs qui n'ont pas réunies les conditions nécéssaires
500                     sont mis en orange pendant 3 secondes pour bien comprendre quelles champs
501                     sont à modifié.
502
503                     La fonction lambda ‘remettreCouleur‘ permet de remettre la couleur initial
504                     après les 3 secondes.
505                     """
506                     remettreCouleur = lambda widget, ancienneCouleur: widget.configure(bg=
        ancienneCouleur)
507                     for champs in mauvaisChamps:
508                         couleur = champs["background"] # couleur d'avant changement
509                         champs.configure(bg="orange") # on change la couleur du champs en orange
510                         # dans 3 secondes on fait : ‘remettreCouleur(champs, couleur)‘
511                         champs.after(3000, remettreCouleur, champs, couleur)
512                 else:
513                     """
514                     Tous les tests sont passés, on peut ajouter l'utilisateur à la base de donnée
515                     Pas besoin de gérer les erreurs lors des casts car on a déjà vérifié que c'était
        bien les bons types avant
516                     """
517                     Stock().ajoutStock(
518                         typeElement.get(),
519                         nom.get(),
520                         int(quantite.get()),
521                         float(prix.get()),
522                         image.get()
523                     )
524                     __affichageTableau() # met à jour le tableau
525
526             # Champs de saisie
527             # Image
528             Label(enfant, text="Image :").grid(column=0, row=0, sticky='e')
529             image = Entry(enfant)
530             image.grid(column=1, row=0, sticky='w')
531             def ___selectionImage():
532                 """Fonction qui permet de choisir une image dans l'arborescence de fichiers de l'
        utilisateur."""
533                 try:
```

```
534                     chemin = askopenfile(title="Choisir une image", filetypes=[("Image GIF", ".gif")
    ])
535                     image.delete(0, "end")
536                     image.insert(0, chemin.name)
537                 except AttributeError: # si l'utilisateur n'a pas choisit d'image
538                     pass
539 
540             Button(enfant, image=self.dossierImage, command=___selectionImage).grid(column=1, row=0,
    sticky='e')
541             # Type (ComboBox)
542             Label(enfant, text="Type :").grid(column=0, row=1, sticky='e')
543             typeElement = Combobox(enfant, values=Stock().listeTypes())
544             # typeElement.current(0) # valeur 0 par défaut
545             typeElement.grid(column=1, row=1, sticky='w')
546             # Nom
547             Label(enfant, text="Nom :").grid(column=0, row=2, sticky='e')
548             nom = Entry(enfant)
549             nom.grid(column=1, row=2, sticky='w')
550             # Quantité
551             Label(enfant, text="Quantité :").grid(column=0, row=3, sticky='e')
552             quantite = Entry(enfant)
553             quantite.grid(column=1, row=3, sticky='w')
554             # Prix à l'unité
555             Label(enfant, text="Prix à l'unité :").grid(column=0, row=4, sticky='e')
556             prix = Entry(enfant)
557             prix.grid(column=1, row=4, sticky='w')
558 
559             def ___viderChamps():
560                 """Vide tout les champs de leur contenu"""
561                 # On récupère toutes les 'Entry' de la fenêtre et on change leur contenu
562                 for champ in [widget for typeElement, widget in enfant.children.items() if "entry" in
     typeElement]:
563                     champ.delete(0, "end")
564                     champ.update()
565 
566             # Boutons
567             Button(enfant, text="Valider", command=___verification).grid(column=0, row=8, columnspan
    =3, sticky='w')
568             Button(enfant, text="Vider les champs", command=___viderChamps).grid(column=0, row=8,
    columnspan=3)
569             Button(enfant, text="Quitter", command=enfant.destroy).grid(column=0, row=8, columnspan
    =3, sticky='e')
570 
571         Button(self.f, text="Ajouter un élément\nau stock", font=self.font, command=
    __ajouterElementStock).grid(column=1, row=2)
572 
573         # -> Partie export des statistiques
574         def __exportation():
575             """Exporte dans un fichier choisie par l'utilisateur ses statistiques de la journée."""
576             chemin = asksaveasfile(title=f"Exportation des statistiques de {caissier['nom']} {
    caissier['prenom']}", filetypes=[("Fichier CSV", ".csv")])
577             if chemin == None: # si rien n'a été spécifie on arrête l'exportation
578                 return
579             Stats().exporteCSV(chemin.name, id)
580         Button(self.f, text="Exporter les statistiques", font=self.font, command=__exportation).grid(
    column=0, row=2, sticky='e', padx=ecart)
581 
582         # -> Boutton pour passer en mode manager si la personne est un manager
583         if caissier["metier"] == 0:
584             Button(self.f, text="Passer en mode Manager", font=self.font, command=lambda: self.
    _interfaceManager(id)).grid(column=0, row=2, sticky='w', padx=ecart)
585 
586         Button(self.f, text="Se déconnecter", font=self.font, command=self._interfaceConnexion).grid(
    column=0, row=2)
587 
588     def _interfaceManager(self, id: int):
589         """Affiche l'interface du manager."""
590         manager = Utilisateurs().recuperationUtilisateur(id=id)
591         # Dans le cas où un utilisateur réussi à trouvé cette interface alors qu'il n'a pas le droit,
     il sera bloqué
592         if manager["metier"] != 0:
593             showerror("Erreur", "Vous ne pouvez pas accéder à cette interface.")
594             return
595         self.parent.title(f"Manager {manager['nom']} {manager['prenom']}  {self.nomApp}")
596         self.dimensionsFenetre(self.parent, 580, 310)
597 
598         # Suppresssion de la dernière Frame
599         self.f.destroy()
600         # Instanciation d'une nouvelle Frame, on va donc ajouter tout nos widgets à cet Frame
601         self.f = Frame(self.parent)
602         self.f.grid()
603 
604         Label(self.f, text="Interface Manager", font=(self.font[0], 20)).grid(column=0, row=0)
605 
```

```
606         Button(self.f, text="Se déconnecter", font=self.font, command=self._interfaceConnexion).grid(
    column=1, row=0, padx=50)
607 
608         Label(self.f).grid(row = 1, pady=10) # séparateur
609 
610         def __ajouterUtilisateur(metier: int):
611             """Permet de créer un nouvel utilisateur, manager (‘metier = 0‘) et caissier (‘metier =
    1‘)."""
612             """
613             L’enfant (‘TopLevel‘) dépend de la ‘Frame‘ et non du parent (‘Tk‘)
614             pour éviter de resté ouverte meme lorsque le manager se déconnecte.
615             """
616             enfant = Toplevel(self.f)
617             enfant.title(f"Ajouter un {’manager’ if metier == 0 else ’caissier’}  {self.nomApp}")
618 
619             def ___verification():
620                 """Vérifie si les champs renseignées sont valides."""
621                 """
622                 Les valeurs ‘Entry‘ qui ne sont pas passés seront dans
623                 la liste ‘mauvaisChamps‘.
624                 Si la liste ‘mauvaisChamps‘ contient un élément alors un test n’est pas ok.
625                 """
626                 mauvaisChamps = []
627                 # vérification pour le nom d’utilisateur
628                 if self.utilisateurCorrect(pseudo.get())[0] == False or Utilisateurs().
    utilisateurExistant(pseudo.get()) == True:
629                     mauvaisChamps.append(pseudo)
630                 # vérification pour le mot de passe
631                 if self.motDePasseCorrect(passe.get())[0] == False:
632                     mauvaisChamps.append(passe)
633                 # vérification pour le nom
634                 if self.nomCorrect(nom.get()) == False:
635                     mauvaisChamps.append(nom)
636                 # vérification pour le prénom
637                 if self.prenomCorrect(prenom.get()) == False:
638                     mauvaisChamps.append(prenom)
639                 # vérification pour la date de naissance
640                 if self.naissanceCorrect(naissance.get()) == False:
641                     mauvaisChamps.append(naissance)
642                 # vérification pour l’adresse
643                 if self.adresseCorrect(adresse.get()) == False:
644                     mauvaisChamps.append(adresse)
645                 # vérification pour le code postal
646                 if self.postalCorrect(postal.get()) == False:
647                     mauvaisChamps.append(postal)
648 
649                 if len(mauvaisChamps) != 0:
650                     """
651                     Tous les champs qui n’ont pas réunies les conditions nécéssaires
652                     sont mis en orange pendant 3 secondes pour bien comprendre quelles champs
653                     sont à modifié.
654 
655                     La fonction lambda ‘remettreCouleur‘ permet de remettre la couleur initial
656                     après les 3 secondes.
657                     """
658                     remettreCouleur = lambda widget, ancienneCouleur: widget.configure(bg=
    ancienneCouleur)
659                     for champs in mauvaisChamps:
660                         couleur = champs["background"] # couleur d’avant changement
661                         champs.configure(bg="orange") # on change la couleur du champs en orange
662                         # dans 3 secondes on fait : ‘remettreCouleur(champs, couleur)‘
663                         champs.after(3000, remettreCouleur, champs, couleur)
664                 else:
665                     # Tous les tests sont passés, on peut ajouter l’utilisateur à la base de donnée
666                     Utilisateurs().ajoutUtilisateur(
667                         pseudo.get(),
668                         passe.get().strip(),
669                         metier,
670                         nom.get(),
671                         prenom.get(),
672                         naissance.get(),
673                         adresse.get(),
674                         int(postal.get()), # pas besoin de gérer d’erreur lors du cast car on a
    vérifié avant que c’était bien une suite de chiffre
675                     )
676                     __ajouterUtilisateursListe(listeUtilisateurs) # met à jour la liste
677 
678             # Champs de saisie
679             # Nom d’utilisateurs
680             Label(enfant, text="Nom d’utilisateur :").grid(column=0, row=0, sticky=’e’)
681             Label(enfant, text="Pas de caractères spéciaux", font=("Arial", 10, "italic")).grid(
    column=2, row=0, sticky=’w’)
682             pseudo = Entry(enfant)
683             pseudo.grid(column=1, row=0, sticky=’w’)
684             # Mot de passe
```

```
685             Label(enfant, text="Mot de passe :").grid(column=0, row=1, sticky='e')
686             Label(enfant, text="1 majuscule, miniscule et caractère spécial minimum", font=("Arial",
        10, "italic")).grid(column=2, row=1, sticky='w')
687             passe = Entry(enfant)
688             passe.grid(column=1, row=1, sticky='w')
689             # Nom
690             Label(enfant, text="Nom :").grid(column=0, row=2, sticky='e')
691             Label(enfant, text="Pas de caractères spéciaux", font=("Arial", 10, "italic")).grid(
        column=2, row=2, sticky='w')
692             nom = Entry(enfant)
693             nom.grid(column=1, row=2, sticky='w')
694             # Prénom
695             Label(enfant, text="Prénom :").grid(column=0, row=3, sticky='e')
696             Label(enfant, text="Pas de caractères spéciaux", font=("Arial", 10, "italic")).grid(
        column=2, row=3, sticky='w')
697             prenom = Entry(enfant)
698             prenom.grid(column=1, row=3, sticky='w')
699             # Date de naissance
700             Label(enfant, text="Date de naissance :").grid(column=0, row=4, sticky='e')
701             Label(enfant, text="Format : AAAA/MM/JJ", font=("Arial", 10, "italic")).grid(column=2,
        row=4, sticky='w')
702             naissance = Entry(enfant)
703             naissance.grid(column=1, row=4, sticky='w')
704             # Adresse
705             Label(enfant, text="Adresse").grid(column=0, row=5, sticky='e')
706             adresse = Entry(enfant)
707             adresse.grid(column=1, row=5, sticky='w')
708             # Code postal
709             Label(enfant, text="Code postal :").grid(column=0, row=6, sticky='e')
710             Label(enfant, text="5 chiffres", font=("Arial", 10, "italic")).grid(column=2, row=6,
        sticky='w')
711             postal = Entry(enfant)
712             postal.grid(column=1, row=6, sticky='w')
713 
714             def ___viderChamps():
715                 """Vide tout les champs de leur contenu"""
716                 # On récupère toutes les `Entry` de la fenêtre et on change leur contenu
717                 for champ in [widget for typeElement, widget in enfant.children.items() if "entry" in
         typeElement]:
718                     champ.delete(0, "end")
719                     champ.update()
720 
721             # Boutons
722             Button(enfant, text="Valider", command=___verification).grid(column=0, row=8, columnspan
        =3, sticky='w')
723             Button(enfant, text="Vider les champs", command=___viderChamps).grid(column=0, row=8,
        columnspan=3)
724             Button(enfant, text="Quitter", command=enfant.destroy).grid(column=0, row=8, columnspan
        =3, sticky='e')
725 
726         def __retirerUtilisateur(metier: int):
727             """Permet de supprimer un utilisateur existant, manager (`metier = 0`) et caissier (`
        metier = 1`)."""
728             enfant = Toplevel(self.f) # cf. l'explication dans `__ajouterUtilisateur`
729             enfant.title(f"Retirer un {'manager' if metier == 0 else 'caissier'}  {self.nomApp}")
730 
731             # Liste des utilisateurs
732             Label(enfant, text=f"Liste des {'manager' if metier == 0 else 'caissier'}", font=self.
        font).grid(column=0, row=0) # titre
733             # On définit une barre pour pouvoir scroller dans la liste
734             scroll_retirer = Scrollbar(enfant)
735             scroll_retirer.grid(column=1, row=1, sticky="nse")
736             # On définit notre liste et on la lie à notre `Scrollbar`
737             listeUtilisateurs_retirer = Listbox(enfant, width=25, height=4, yscrollcommand=
        scroll_retirer.set)
738             scroll_retirer.config(command=listeUtilisateurs_retirer.yview) # scroll à la verticale
        dans notre liste
739             # On ajoute nos utilisateurs à notre liste
740             __ajouterUtilisateursListe(listeUtilisateurs_retirer, metier)
741             listeUtilisateurs_retirer.grid(column=0, row=1)
742             # On affiche l'utilisateur quand on double-clique dessus
743 
744             def ___suppressionUtilisateur():
745                 """Supprime l'utilisateur actuellement sélectionné dans la liste"""
746                 element = listeUtilisateurs_retirer.curselection()
747                 if len(element) == 0: # si aucun élément n'est selectionné
748                     showwarning("Attention", "Aucun utilisateur n'a été selectionné.")
749                 else:
750                     utilisateur = listeUtilisateurs_retirer.get(listeUtilisateurs_retirer.
        curselection()[0]).split('(')[0][:-1]
751                     reponse = askyesno("Confirmation", f"Voulez vous supprimer {utilisateur} ?")
752                     if reponse == True:
753                         Utilisateurs().suppressionUtilisateurs(utilisateur)
754                         __ajouterUtilisateursListe(listeUtilisateurs_retirer) # met à jour la liste
        dans la fenêtre de suppression
```

```
                        __ajouterUtilisateursListe(listeUtilisateurs) # met à jour la liste dans l'
    interface principale
                        showinfo("Information", f"Utilisateur {utilisateur} supprimé.")

            # Boutons
            Button(enfant, text="Supprimer", command=___suppressionUtilisateur).grid(column=0, row=8,
     columnspan=3, sticky='w')
            Button(enfant, text="Quitter", command=enfant.destroy).grid(column=0, row=8, columnspan
    =3, sticky='e')

        def __afficherInformationsUtilisateur(_):
            """Permet d'afficher les informations d'un utilisateur"""
            element = listeUtilisateurs.curselection()
            if len(element) == 0: # si aucun élément n'est selectionné
                return
            """
            On split le champs car dans la liste on affiche le métier entre
            parenthèses et on doit donner que le nom d'utilisateur à
            la fonction `recuperationUtilisateur`, aussi on retire le dernièr
            charactère avec [:-1] car c'est un espace.
            """
            utilisateur = Utilisateurs().recuperationUtilisateur(pseudo=listeUtilisateurs.get(element
    [0]).split('(')[0][:-1])
            enfant = Toplevel(self.f) # cf. l'explication dans `__ajouterUtilisateur`
            enfant.title(f"{utilisateur['nom']} {utilisateur['prenom']}  {self.nomApp}")

            # Informations sur l'utilisateur
            frameInfos = LabelFrame(enfant, text="Informations utilisateur", font=self.font)
            frameInfos.grid(column=0, row=0, sticky='n', padx=5)
            utilisateur["metier"] = "Manager" if utilisateur["metier"] == 0 else "Caissier"
            del utilisateur["passe"] # le manager ne doit pas connaître le mot de passe de l'
    utilisateur
            for idx, cle in enumerate(utilisateur):
                if cle == "id": # on ignore l'ID
                    continue
                cleAffichage = cle.capitalize()
                Label(frameInfos, text=f"{cleAffichage} :").grid(column=0, row=idx + 1, sticky='e')
                Label(frameInfos, text=utilisateur[cle]).grid(column=1, row=idx + 1, sticky='w')

            frameSuivi = LabelFrame(enfant, text="Histogramme des ventes", font=self.font)
            frameSuivi.grid(column=1, row=0, sticky="ns", padx=5)

            def ___actualisationCanvas():
                """Affiche l'histogramme des vente d'un utilisateur dans un canvas."""
                donnees = Stats().recuperationDonneesCSV(utilisateur['id'])
                if len(donnees) <= 0:
                    histogramme.create_text(10, 10, anchor='w', text="Aucun résultat récemment
    enregistré")
                else:
                    # Les dates dans le fichier CSV ne sont pas dans l'ordre
                    # On retire l'ID et le pseudo du dictionnaire
                    donnees.pop("id")
                    donnees.pop("pseudo")
                    ecart = 10
                    couleurs = [
                        "CadetBlue3",
                        "HotPink2",
                        "IndianRed1",
                        "MediumPurple2",
                        "burlywood2",
                        "brown3",
                        "chocolate1",
                        "goldenrod2"
                    ]
                    maxVente = 0 # par défaut la meilleur vente est de 0
                    for prix in donnees.values():
                        prix = float(prix)
                        if prix > maxVente: # si on trouve une valeure plus grande
                            maxVente = prix

                    for date in sorted(donnees.keys()): # on regarde les dates dans l'ordre
                        # Affichage de la date
                        histogramme.create_text(ecart + 10, 60, anchor='w', text=date, font=("Arial",
     8), angle=90)
                        # Affichage de la barre
                        hauteur = 190 - (float(donnees[date]) * 100) / maxVente # calcul de la
    hauteur en fonction de la plus grosse vente
                        # On fait `- 20` au résultat pour allonger la barre, aussi on met une barre
    de `2` pixel quand valeur petite
                        histogramme.create_rectangle(ecart, 180, ecart + 15, hauteur - 20 if hauteur
    < 180 else 178, fill=couleurs.pop())
                        # Affichage du montant
                        histogramme.create_text(ecart, 190, anchor='w', text=donnees[date], font=("
    Arial", 8))
                        ecart += 30
```

```
829
830             histogramme = Canvas(frameSuivi, width=270, height=200)
831             histogramme.grid()
832
833             ___actualisationCanvas()
834
835             Button(enfant, text="Quitter", command=enfant.destroy).grid(column=0, row=1, columnspan
        =2)
836
837         Button(self.f, text="Ajouter un caissier", font=self.font, command=lambda:
        __ajouterUtilisateur(1)).grid(column=0, row=2)
838         Button(self.f, text="Retirer un caissier", font=self.font, command=lambda:
        __retirerUtilisateur(1)).grid(column=1, row=2)
839
840         Label(self.f).grid(row = 3, pady=10) # séparateur
841
842         # Liste des utilisateurs
843         managerVerif = IntVar(self.f) # filtre pour afficher ou non les managers dans la liste
844         caissierVerif = IntVar(self.f) # filtre pour afficher ou non les caissiers ou non dans la
        liste
845
846         caissierVerif.set(1) # par défaut on affiche que les caissiers
847
848         def __ajouterUtilisateursListe(liste: Listbox, force: int = None):
849             """
850             Ajoute des utilisateurs à la liste du Manager.
851                 -> metier =  0 : manager uniquement
852                 -> metier =  1 : caissier uniquement
853                 -> metier =  2 : manager et caissier
854             """
855             liste.delete(0, "end") # vidé la liste des utilisateurs
856             if force: # si `force` n'est pas `None`, alors on force l'utilisation d'un métier
857                 metier = force
858             else: # sinon on fait une vérification normale en fonction des filtres de l'interface
        manager
859                 if managerVerif.get() == 1:
860                     if caissierVerif.get() == 1:
861                         metier = None # on affiche les 2
862                     else:
863                         metier = 0 # on affiche seulement les managers
864                 else:
865                     metier = 1 # on affiche les caissiers
866                     if caissierVerif.get() == 0: # rien est coché, on revient à la configuration par
        défaut (caissiers uniquement)
867                         metier = 1
868                         caissierVerif.set(1)
869
870             if metier == None: # on ajoute tous les utilisateurs
871                 for idx, utilisateur in enumerate(Utilisateurs().listUtilisateurs()):
872                     liste.insert(idx, f"{utilisateur[0]} ({'manager' if utilisateur[1] == 0 else '
        caissier'})")
873             elif metier == 0: # on ajoute que les managers
874                 for idx, utilisateur in enumerate(Utilisateurs().listUtilisateurs()):
875                     if utilisateur[1] == metier:
876                         liste.insert(idx, f"{utilisateur[0]} ({'manager' if utilisateur[1] == 0 else
        'caissier'})")
877             elif metier == 1: # on ajoute que les caissiers
878                 for idx, utilisateur in enumerate(Utilisateurs().listUtilisateurs()):
879                     if utilisateur[1] == metier:
880                         liste.insert(idx, f"{utilisateur[0]} ({'manager' if utilisateur[1] == 0 else
        'caissier'})")
881             else: # ce cas est là au cas où mais n'est pas sensé être appellé
882                 raise NameError("Métier inconnu.")
883
884         # Label d'information
885         Label(self.f, text="""
886                             Double-cliquez sur un
887                             utilisateur de la liste
888                             pour obtenir des informations
889                             supplémentaire à son sujet.
890                             """, justify="right").grid(column=1, row=4, rowspan=2, sticky="e")
891
892         Label(self.f, text="Liste des utilisateurs", font=self.font).grid(column=0, row=4) # titre
893         # On définit une barre pour pouvoir scroller dans la liste
894         scroll = Scrollbar(self.f)
895         scroll.grid(column=0, row=5, sticky="nse")
896         # On définit notre liste et on la lie à notre `Scrollbar`
897         listeUtilisateurs = Listbox(self.f, width=25, height=4, yscrollcommand=scroll.set)
898         scroll.config(command=listeUtilisateurs.yview) # scroll à la verticale dans notre liste
899         # On ajoute nos utilisateurs à notre liste
900         __ajouterUtilisateursListe(listeUtilisateurs)
901         listeUtilisateurs.grid(column=0, row=5)
902         listeUtilisateurs.bind('<Double-Button>', __afficherInformationsUtilisateur) # on affiche l'
        utilisateur quand on double-clique dessus
903
```

```
904             # Filtre pour la liste
905             Label(self.f, text="Filtre", font=self.font).grid(column=1, row=4, sticky='w', padx=10) #
        titre
906             filtres = Frame(self.f) # Morceau qui va contenir nos checkbutton
907             filtres.grid(column=1, row=4, rowspan=2, sticky='w')
908             Checkbutton(filtres, text="Manager", variable=managerVerif, command=lambda:
        __ajouterUtilisateursListe(listeUtilisateurs)).grid(sticky='w')
909             Checkbutton(filtres, text="Caissier", variable=caissierVerif, command=lambda:
        __ajouterUtilisateursListe(listeUtilisateurs)).grid(sticky='w')
910 
911             Button(self.f, text="Passer en mode caissier", font=self.font, command=lambda: self.
        _interfaceCaissier(id)).grid(column=0, row=6, columnspan=3, pady=10)
912 
913 if __name__ == "__main__":
914     """Application "GesMag" pour le module de Programmation d'interfaces (2021-2022)"""
915     """
916     Si presentation = True alors une base de donnée par défaut sera généré.
917     Si presentation = False ou n'est même pas mentionné, alors aucune base de donnée par défaut ne
        sera généré.
918     """
919     GesMag(presentation = True).demarrer()
```

2.2 `db.py`, gère la communication avec la base de donnée en sa globalité

```
1  import sqlite3
2  
3  class BaseDeDonnees:
4      """Gère la base de donnée."""
5      def __init__(self, urlBaseDeDonnee: str):
6          self.connexion = self.creerConnexion(urlBaseDeDonnee)
7  
8      def creerConnexion(self, path: str):
9          """Connexion à une base de donnée SQLite."""
10         if not self.fichierExiste(path): # si l base de donnée n'existe pas
11             open(path, 'x') # on la créer
12         try:
13             connnexion = sqlite3.connect(path)
14         except sqlite3.Error as e:
15             print(e) # on affiche l'erreur
16             connnexion = None # et renvoie None
17         return connnexion
18 
19     def fichierExiste(self, path: str) -> bool:
20         """Vérifie qu'un fichier existe."""
21         try: # on essaie d'ouvrir le fichier
22             open(path, 'r')
23         except FileNotFoundError: # si le fichier n'existe pas
24             return False
25         else: # si le fichier existe
26             return True
27 
28     def requete(self, requete: str, valeurs = None):
29         """Envois une requête vers la base de données."""
30         try:
31             curseur = self.connexion.cursor()
32             if valeurs: # s'il y a des valeurs alors on lance la commande ‘execute‘ avec ses
        dernières
33                 if type(valeurs) not in [list, tuple]: # si la valeur c'est juste une chaîne de
        charactère (par exemple), alors la converti en liste
34                     valeurs = [valeurs]
35                 curseur.execute(requete, valeurs)
36             else: # sinon on lance juste la requête
37                 curseur.execute(requete)
38             self.connexion.commit() # applique les changements à la base de donnée
39             return (curseur, curseur.lastrowid) # renvoie le curseur et l'ID de l'élément modifié
40         except sqlite3.Error as e: # s'il y a eu une erreur SQLite
41             print(e)
42 
43     def affichageResultat(self, curseur: tuple) -> list:
44         """Affiche le résultat d'une requête."""
45         tableau = []
46         if curseur == None: # si le curseur est vide il n'y a rien a affiché (tableau vide)
47             return tableau
48         lignes = curseur[0].fetchall() # sinon on récupère les éléments
49         for ligne in lignes:
50             tableau.append(ligne) # on les ajoute au tableau
51         return tableau # on le renvoie
52 
53     def affichageResultatDictionnaire(self, cles, curseur: sqlite3.Cursor) -> dict:
54         """
55         Même but que ‘affichageResultat()‘ mais avec
56         les clés qui correspondent aux valeurs.
```

```
        """
        valeurs = self.affichageResultat(curseur)
        if len(valeurs) == 0:
            valeurs = []
        else:
            valeurs = valeurs[0]
        if type(cles) not in [list, tuple]:
            cles = [cles]
        if len(cles) != len(valeurs):
            raise IndexError # il y a pas autant de clés que de valeurs
        return dict(zip(cles, valeurs))
```

2.3 `users.py`, implante la base de donnée pour les utilisateurs

```
from db import BaseDeDonnees

class Utilisateurs(BaseDeDonnees):
    """Gère une table "utilisateurs" pour une base de donnée donné."""
    def __init__(self):
        super().__init__(r"db.sqlite3") # connexion à la base de donnée

    def creationTable(self, presentation: bool = False) -> None:
        """Créer la table qui stocker les utilisateurs."""
        requete = """
                  CREATE TABLE IF NOT EXISTS utilisateurs (
                      id INTEGER PRIMARY KEY,
                      pseudo TEXT,
                      passe TEXT,
                      metier INTEGER,
                      nom TEXT,
                      prenom TEXT,
                      naissance TEXT,
                      adresse TEXT,
                      postal INTEGER
                  );
                  """
        self.requete(requete)
        # Ajout d'un utilisateur par défaut si aucun utilisateur n'existe dans la base de donnée
        if len(self.listUtilisateurs()) == 0 and presentation:
            self.ajoutUtilisateur(
                pseudo="admin",
                passe="P@ssword",
                metier=0,
                nom="Admin",
                prenom="Système",
                naissance="2000/10/09",
                adresse="12 Rue de Montmartre",
                postal=46800
            )
            print("--  Compte par défaut --\nNom d'utilisateur: admin\nMot de passe: P@ssword")

    def ajoutUtilisateur(self, pseudo: str, passe: str, metier: int, nom: str, prenom: str, naissance
    : str, adresse: str, postal: int) -> list:
        """Ajoute un utilisateur et retourne l'ID de ce dernier."""
        requete = """
                  INSERT INTO utilisateurs (
                      pseudo, passe, metier, nom, prenom, naissance, adresse, postal
                  ) VALUES (
                      ?, ?, ?, ?, ?, ?, ?, ?
                  );
                  """
        self.requete(requete, [pseudo.lower(), passe, metier, nom.upper(), prenom.capitalize(),
     naissance, adresse, postal])
        return self.affichageResultat(self.requete("SELECT last_insert_rowid();"))

    def suppressionUtilisateurs(self, pseudo: str) -> None:
        """Supprime un utilisateur."""
        requete = """
                  DELETE FROM utilisateurs
                  WHERE pseudo = ?
                  """
        self.requete(requete, pseudo)

    def verificationIdentifiants(self, pseudo: str, motDePasse: str):
        """
        Retourne l'ID de l'utilisateur si trouvé dans la base de donnée ainsi
        que son métier (‘tuple‘), sinon renvoie ‘(0,)‘.
        """
        requete = """
                  SELECT id, metier FROM utilisateurs
                  WHERE pseudo = ? AND passe = ?
                  """
```

```
        reponseBaseDeDonnee = self.affichageResultat(self.requete(requete, [pseudo.lower(), motDePasse]))
        if len(reponseBaseDeDonnee) == 0: # si les identifiants renseignés sont mauvais
            return (0,)
        return reponseBaseDeDonnee[0]

    def listUtilisateurs(self) -> list:
        """Retourne la liste des nom d'utilisateurs (avec leur métier)."""
        requete = """
                  SELECT pseudo, metier FROM utilisateurs
                  """
        return self.affichageResultat(self.requete(requete))

    def recuperationUtilisateur(self, id: int = None, pseudo: str = None) -> dict:
        """Retourne les informations d'un utilisateur grâce à son ID ou son pseudo (ID en priorité).
        """
        recuperation = [
            "id",
            "pseudo",
            "passe",
            "metier",
            "nom",
            "prenom",
            "naissance",
            "adresse",
            "postal",
        ]
        if not id: # si la variable 'id' n'est pas définie
            if not pseudo: # si seul la variable 'pseudo' n'est pas définie
                raise ValueError # Aucun utilisateur renseigné
            else: # si un pseudo est renseigné, c'est ce qu'on va utilisé
                requete = f"""
                           SELECT {", ".join(recuperation)} FROM utilisateurs
                           WHERE pseudo = ?
                           """
            utilisateur = pseudo
        else: # si un id est renseigné, c'est ce qu'on va utilisé
            requete = f"""
                       SELECT {", ".join(recuperation)} FROM utilisateurs
                       WHERE id = ?
                       """
            utilisateur = id
        return self.affichageResultatDictionnaire(recuperation, self.requete(requete, utilisateur))

    def utilisateurExistant(self, utilisateur: str) -> bool:
        """Vérifie si l'utilisateur donnée existe déjà dans la base de donnée."""
        requete = """
                  SELECT EXISTS (
                      SELECT 1 FROM utilisateurs
                      WHERE pseudo = ?
                  )
                  """
        return True if self.affichageResultat(self.requete(requete, utilisateur.lower()))[0][0] == 1 else False
```

2.4 `stock.py`, implante la base de donnée pour le stock

```
from random import randint, uniform

from db import BaseDeDonnees

class Stock(BaseDeDonnees):
    """Gêre une table "stock" pour une base de donnée donné."""
    def __init__(self):
        super().__init__(r"db.sqlite3") # connexion à la base de donnée

    def creationTable(self, presentation: bool = False) -> None:
        """Créer la table qui stocker les stocks."""
        requete = """
                  CREATE TABLE IF NOT EXISTS stocks (
                      id INTEGER PRIMARY KEY,
                      type TEXT,
                      nom TEXT,
                      quantite INTEGER,
                      prix REAL,
                      image_url TEXT
                  );
                  """
        self.requete(requete)
        # Ajout d'un stock par défaut si aucun stock n'existe dans la base de donnée
        if len(self.listeStocks()) == 0 and presentation:
            # Créer un dictionnaire d'éléments pour mieux voir ce que l'on ajoute à la base de donnée
```

```
            defaut = {
                "fruits legumes": [
                    ("banane", "img/banane.gif"),
                    ("orange", "img/orange.gif"),
                    ("betterave", "img/betterave.gif"),
                    ("carottes", "img/carottes.gif"),
                    ("tomates", "img/tomates.gif"),
                    ("citron", "img/citron.gif"),
                    ("kiwi", "img/kiwi.gif"),
                    ("clementine", "img/clementine.gif"),
                    ("pomme", "img/pomme.gif"),
                    ("avocat", "img/avocat.gif")
                ],
                "boulangerie": [
                    ("brownie", "img/brownie.gif"),
                    ("baguette", "img/baguette.gif"),
                    ("pain au chocolat", "img/pain_au_chocolat.gif"),
                    ("croissant", "img/croissant.gif"),
                    ("macaron", "img/macaron.gif"),
                    ("millefeuille", "img/millefeuille.gif"),
                    ("paris-brest", "img/paris-brest.gif"),
                    ("opera", "img/opera.gif"),
                    ("fraisier", "img/fraisier.gif"),
                    ("eclair", "img/eclair.gif")
                ],
                "boucherie poissonnerie": [
                    ("saucisson", "img/saucisson.gif"),
                    ("côte de boeuf", "img/cote_de_boeuf.gif"),
                    ("langue de boeuf", "img/langue_de_boeuf.gif"),
                    ("collier de boeuf", "img/collier_de_boeuf.gif"),
                    ("entrecote", "img/entrecote.gif"),
                    ("cabillaud", "img/cabillaud.gif"),
                    ("saumon", "img/saumon.gif"),
                    ("colin", "img/colin.gif"),
                    ("bar", "img/bar.gif"),
                    ("dorade", "img/dorade.gif")
                ],
                "entretien": [
                    ("nettoyant air comprimé", "img/nettoyant_air_comprime.gif"),
                    ("nettoyage anti-bactérien", "img/nettoyage_anti-bacterien.gif"),
                    ("nettoyant pour écran", "img/nettoyant_pour_ecran.gif"),
                    ("nettoyant pour lunettes", "img/nettoyant_pour_lunettes.gif"),
                    ("pioche", "img/pioche.gif"),
                    ("pelle", "img/pelle.gif"),
                    ("lampe torche", "img/lampe_torche.gif"),
                    ("gants", "img/gants.gif"),
                    ("éponge", "img/eponge.gif"),
                    ("essuie-tout", "img/essuie-tout.gif")
                ]
            }

            # Ajoute le dictionnaire précédemment créer à la base de donnée avec un prix et une quantité aléatoire
            for type in defaut:
                for element in defaut[type]:
                    self.ajoutStock(type, element[0], randint(0, 10), round(uniform(2., 30.), 2), element[1])

        def ajoutStock(self, typeElement: str, nom: str, quantite: int, prix: float, imageURL: str) -> list:
            """Ajoute un élément dans le stock et retourne l'ID de ce dernier."""
            requete = """
                      INSERT INTO stocks (
                          type, nom, quantite, prix, image_url
                      ) VALUES (
                          ?, ?, ?, ?, ?
                      );
                      """
            self.requete(requete, [typeElement.lower(), nom.lower(), quantite, prix, imageURL])
            return self.affichageResultat(self.requete("SELECT last_insert_rowid();"))

        def reduitQuantiteStock(self, id: int, quantiteARetirer: int) -> None:
            """Retire une quantité d'un élément du stock et met-à-jour la base de donnée."""
            requeteA = """
                       SELECT quantite FROM stocks
                       WHERE id = ?
                       """
            quantiteActuelle: int = self.affichageResultat(self.requete(requeteA, id))[0][0]
            if quantiteActuelle <= quantiteARetirer: # il ne reste plus rien
                quantiteFinale = 0
            else: # il reste quelque chose
                quantiteFinale = quantiteActuelle - quantiteARetirer
            # On met à jour la quantité de l'élément dans la base de donnée
            requeteB = """
                       UPDATE stocks
```

```
                    SET quantite = ?
                    WHERE id = ?
                    """
        self.requete(requeteB, [quantiteFinale, id])

    def listeStocks(self) -> list:
        """Retourne la liste des éléments en stock sous forme de dictionnaire."""
        recuperation = [
            "id",
            "type",
            "nom",
            "quantite",
            "prix",
            "image_url"
        ]
        requete = f"""
                    SELECT {", ".join(recuperation)} FROM stocks
                    """
        return [dict(zip(recuperation, element)) for element in self.affichageResultat(self.requete(
requete))]

    def stockExistant(self, stock: str) -> bool:
        """Vérifie si le stock donnée existe déjà dans la base de donnée."""
        requete = """
                    SELECT EXISTS (
                        SELECT 1 FROM stocks
                        WHERE nom = ?
                    )
                    """
        return True if self.affichageResultat(self.requete(requete, stock.lower()))[0][0] == 1 else
False

    def listeTypes(self) -> list:
        """Renvoie la liste des types disponibles dans la base de donnée."""
        requete = """
                    SELECT type FROM stocks
                    """
        res = []
        for i in self.affichageResultat(self.requete(requete)):
            if i[0] not in res:
                res.append(i[0])
        return res
```

2.5 `stats.py`, implante la gestion des statistiques et son export en format `CSV`

```
import csv

from datetime import date, timedelta

from users import Utilisateurs

class Stats():
    """Gère les statistiques et son export en format CSV."""
    def __init__(self):
        self.formatDate = "%Y/%m/%d"

    def datesDisponibles(self) -> list:
        """Renvoie les dates disponibles pour l'entête du fichier `CSV`."""
        datesPossibles = []
        dateAujourdHui = date.today() - timedelta(days=7)
        for _ in range(0, 8):
            datesPossibles.append(dateAujourdHui.strftime(self.formatDate))
            dateAujourdHui = dateAujourdHui + timedelta(days=1)
        return datesPossibles

    def creationCSV(self, force: bool = False):
        """
        Créer le fichier `CSV` qui stockera les statistiques pour tous les utilisateurs.

        Possibilité de forcer la création (c-à-d même si le fichier existe déjà) en renseignant
        `force = True`
        """
        if not Utilisateurs().fichierExiste("stats.csv") or force:
            with open("stats.csv", 'w') as f:
                fichier = csv.writer(f)
                fichier.writerow(["id", "pseudo"] + self.datesDisponibles())

    def miseAJourStatsUtilisateur(self, utilisateurID: int, prix: float):
        """
        Récupère le prix d'une transaction et l'ajoute au total d'un utilisateur.

        - s'il y a déjà une valeur dans la base de donnée correspondant à la date du jour,
```

```
            on met à jour cette valeur en l'additionnant avec le nouveaux prix
        - s'il n'avait pas de valeur à cette date:
            - si l'utilisateur n'est pas dans le fichier, on rajoute une ligne avec le prix
            - si l'utilisateur est présent mais aucun prix n'est fixé pour la date du jour,
              on rajoute le prix sur la ligne de l'utilisateur déjà existante

        On remplie les espaces vides par la valeur '0' (car aucun chiffre n'a été fait ce jour là,
        car aucune information n'était renseignée).
        """
        self.miseAJourDatesCSV() # met-à-jour les dates du fichier 'CSV'

        # Mets-à-jour le 'CSV' avec le nouveau prix...
        aujourdHui = date.today().strftime(self.formatDate)
        with open("stats.csv", 'r') as f:
            fichier = list(csv.reader(f))
            # On récupère la colonne pour aujourd'hui
            index = 0
            locationDate = None # note l'index de la colonne de la date dans le fichier
            for nomColonne in fichier[0]: # on regarde l'entente
                if nomColonne == aujourdHui: # on regarde si la colonne correspond à la date du jour
                    locationDate = index # on note l'entête
                index += 1
            if locationDate == None: # ne devrait pas arrivé car on mets à jour les dates du 'CSV'
    avant de lire le fichier
                raise IndexError("Date du jour non trouvé dans le fichier csv.")

            utilisateur = Utilisateurs().recuperationUtilisateur(utilisateurID) # on récupère les
    infos de l'utilisateur
            # Vérification si l'utilisateur est déjà présent dans le fichier 'CSV'
            locationUtilisateur = None # note l'index de la ligne de l'utilisateur dans le fichier
            for idx, location in enumerate(fichier):
                if location[0] == str(utilisateurID):
                    locationUtilisateur = idx
            if locationUtilisateur == None: # si l'utilisateur n'est pas présent dans le fichier
                # on rajoute la ligne
                fichier += [[utilisateurID, utilisateur["pseudo"]] + ['0' for _ in range(0,
    locationDate - 2)] + [prix]]
            else: # si déjà présent dans le fichier
                try:
                    ancienPrix = float(fichier[locationUtilisateur][locationDate]) # on récupère l'
    ancien prix
                except IndexError: # si il n'y avait pas de prix définie avant
                    ancienPrix = 0
                    # On rajoute la case
                    fichier[locationUtilisateur] += ['0' for _ in range(0, locationDate - 1)]
                ancienPrix += prix # on y ajoute le nouveaux prix
                fichier[locationUtilisateur][locationDate] = f"{float(ancienPrix):.2f}" # on met à
    jour le fichier

        with open("stats.csv", 'w') as f: # on applique les changements
            ecriture = csv.writer(f)
            ecriture.writerows(fichier)


    def exporteCSV(self, chemin: str, utilisateurID: int):
        """
        Exporte les statistiques d'un utilisateur dans un fichier 'CSV'.
        - N'exporte que les statistiques du jour.
        """
        donnees = self.recuperationDonneesCSV(utilisateurID)
        aujourdHui = date.today().strftime(self.formatDate)
        with open(chemin, 'w') as f:
            fichier = csv.writer(f)
            fichier.writerow(["ID Utilisateur", f"Totales des ventes du jour ({aujourdHui})"])
            if len(donnees) > 0: # si il y a des données enregistrées
                fichier.writerow([utilisateurID, donnees[aujourdHui]])
            else:
                fichier.writerow([utilisateurID, "Aucune ventes enregistrée"])

    def recuperationDonneesCSV(self, utilisateurID: int) -> dict:
        """Renvoie les informations contenu dans le fichier 'CSV' globale."""
        self.miseAJourDatesCSV() # met à jour les dates du fichier 'CSV'
        with open("stats.csv", 'r') as f:
            fichier = list(csv.DictReader(f)) # lecture du fichier sous forme d'une liste de
    dictionnaire
            for utilisateur in fichier: # on regarde tous les utilisateurs stockés dans le fichier
                if utilisateur["id"] == str(utilisateurID): # si utilisateur trouvé
                    return utilisateur # renvoie des infos de l'utilisateur
        return {} # ne retourne rien si l'utilisateur n'était pas présent dans le fichier

    def miseAJourDatesCSV(self):
        """
        Mets-à-jour les dates trop anciennes du fichier globales 'CSV'.

        On remplie les espaces vides par la valeur '0' pour les jours qui remplacent les dates
```

```
        trop vieilles (âgées de plus d'une semaine) car soit aucun chiffre n'a été fait ce jour là,
        car aucune information n'était renseignée.
        """
        besoinDeMofication = False
        with open("stats.csv", 'r') as f:
            fichier = list(csv.reader(f))
            if len(fichier) == 1: # si fichier ne contient que l'entête
                self.creationCSV(True) # on recréer le fichier dans le doute (avec les bonnes dates)
            else:
                index = 2 # variable qui permet de savoir quel index on regarde (on commence à 2 car on ignore les 2 premières valeurs [id et pseudo])
                mauvaisIndex = [] # liste qui va stocker les index trop vieux (> 1 semaine)
                datesPresentes = [] # liste qui stock les dates valides présentes dans le fichier (< 1 semaine)
                datesDisponibles = self.datesDisponibles() # liste des bonnes dates
                for dateFichier in fichier[0][index:]: # on regarde toutes les dates du fichier
                    if dateFichier not in datesDisponibles: # si trop vieux
                        mauvaisIndex.append(index) # on ajoute l'index à la liste
                        besoinDeMofication = True
                    else:
                        datesPresentes.append(dateFichier) # on ajoute la date à la liste
                    index += 1

                if not besoinDeMofication: # vérification si on a besoin de rien faire
                    return # on quitte la fonction

                datesARajouter = [date for date in datesDisponibles if date not in datesPresentes] # liste des dates à rajouter
                if len(datesARajouter) != len(mauvaisIndex): # ne devrais pas arrivé mais on sait jamais
                    raise IndexError("Problème, pas autant de dates à rajouter que de de dates périmés dans le fichier.")
                for idx in mauvaisIndex: # pour tous les mauvais index
                    for numLigne, ligne in enumerate(fichier): # on regarde toutes les lignes du fichier
                        if idx < len(ligne): # s'il y a un élément dans la ligne à l'index donnée ou si elle est vide de toute façon
                            if numLigne == 0: # si c'est la ligne d'entête
                                ligne[idx] = datesARajouter[0] # on change la ligne avec la nouvelle date
                                datesARajouter.pop(0)
                            else: # si c'est une ligne de donnée
                                ligne[idx] = '0' # on change la ligne avec une valeur vide
                            fichier[numLigne] = ligne # on applique les changements
        if besoinDeMofication: # vérification si on a besoin de faire des changements
            with open("stats.csv", 'w') as f: # on applique les changements
                ecriture = csv.writer(f)
                ecriture.writerows(fichier)
```