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) -> None:
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(f"Erreur  {self.nomApp}",  pseudoOk[1])
124             return
125         mdpOk = self.motDePasseCorrect(motDePasse)
126         if not mdpOk[0]:
127             showerror(f"Erreur  {self.nomApp}", 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(f"Erreur  {self.nomApp}", "Une erreur est survenue : métier inconnue.")
139         else:
140             showerror(f"Erreur  {self.nomApp}", "Utilisateur ou mot de passe incorrect.")
141
142     def dimensionsFenetre(self, fenetre, nouveauX: int, nouveauY: int) -> None:
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) -> None:
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) -> None:
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) -> None:
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) -> None:
```

```
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) -> None:
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(f"Erreur  {self.nomApp}", "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(f"Erreur  {self.nomApp}", "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() -> None:
            """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() -> None:
            """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(f"Validation  {self.nomApp}", "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() -> None:
            """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.resizable(False, False)
```

```
454             enfant.title(f"Ajouter un élément au stock  {self.nomApp}")
455 
456             def ___verification() -> None:
457                 """Vérifie si les champs renseignées sont valides."""
458                 """
459                 La variable ‘ok‘ sert à savoir si la vérification est passée
460                 si elle vaut ‘True‘ alors tout est bon,
461                 Par contre si elle vaut ‘False‘ alors il y a eu une erreur.
462                 Les valeurs ‘Entry‘ qui ne sont pas passés seront dans
463                 la liste ‘mauvaisChamps‘.
464                 """
465                 ok = True
466                 mauvaisChamps = []
467                 # vérification pour l'image, on utilise la fonction du fichier ‘db.py‘
468                 if Stock().fichierExiste(image.get()) == False:
469                     ok = False
470                     mauvaisChamps.append(image)
471                 # vérification pour le type
472                 if typeElement.get() not in Stock().listeTypes():
473                     ok = False
474                     # Pas de coloration orange si le type est mauvais parce que on ne peut pas changé
     la couleur de fond d'une ComboBox
475                 # vérification pour le nom
476                 def ___nomValide(nom: str) -> bool:
477                     """
478                     Vérifie si un nom est valide pour le stock.
479                     (non vide et pas déjà présent dans la base de donnée)
480                     """
481                     if len(nom) <= 0:
482                         return False
483                     if Stock().stockExistant(nom) == True:
484                         return False
485                     return True
486                 if ___nomValide(nom.get()) == False:
487                     ok = False
488                     mauvaisChamps.append(nom)
489                 # vérification pour la quantité
490                 try:
491                     int(quantite.get()) # conversion en int
492                 except ValueError: # si la conversion a échoué
493                     ok = False
494                     mauvaisChamps.append(quantite)
495                 # vérification pour le prix
496                 try:
497                     float(prix.get()) # conversion en float
498                 except ValueError: # si la conversion a échoué
499                     ok = False
500                     mauvaisChamps.append(prix)
501 
502                 if ok == False:
503                     """
504                     Tous les champs qui n'ont pas réunies les conditions nécéssaires
505                     sont mis en orange pendant 3 secondes pour bien comprendre quelles champs
506                     sont à modifié.
507 
508                     La fonction lambda ‘remettreCouleur‘ permet de remettre la couleur initial
509                     après les 3 secondes.
510                     """
511                     remettreCouleur = lambda widget, ancienneCouleur: widget.configure(bg=
    ancienneCouleur)
512                     for champs in mauvaisChamps:
513                         couleur = champs["background"] # couleur d'avant changement
514                         champs.configure(bg="orange") # on change la couleur du champs en orange
515                         # dans 3 secondes on fait : ‘remettreCouleur(champs, couleur)‘
516                         champs.after(3000, remettreCouleur, champs, couleur)
517                 else:
518                     """
519                     Tous les tests sont passés, on peut ajouter l'utilisateur à la base de donnée
520                     Pas besoin de gérer les erreurs lors des casts car on a déjà vérifié que c'était
    bien les bons types avant
521                     """
522                     Stock().ajoutStock(
523                         typeElement.get(),
524                         nom.get(),
525                         int(quantite.get()),
526                         float(prix.get()),
527                         image.get()
528                     )
529                     __affichageTableau() # met à jour le tableau
530 
531             # Champs de saisie
532             # Image
533             Label(enfant, text="Image :").grid(column=0, row=0, sticky='e')
534             image = Entry(enfant)
535             image.grid(column=1, row=0, sticky='w')
```

```
536             def ___selectionImage() -> None:
537                 """Fonction qui permet de choisir une image dans l'arborescence de fichiers de l'
    utilisateur."""
538                 try:
539                     chemin = askopenfile(title=f"Choisir une image  {self.nomApp}", filetypes=[("
    Image GIF", ".gif")])
540                     image.delete(0, "end")
541                     image.insert(0, chemin.name)
542                 except AttributeError: # si l'utilisateur n'a pas choisit d'image
543                     pass
544 
545             Button(enfant, image=self.dossierImage, command=___selectionImage).grid(column=1, row=0,
    sticky='e')
546             # Type (ComboBox)
547             Label(enfant, text="Type :").grid(column=0, row=1, sticky='e')
548             typeElement = Combobox(enfant, values=Stock().listeTypes())
549             # typeElement.current(0) # valeur 0 par défaut
550             typeElement.grid(column=1, row=1, sticky='w')
551             # Nom
552             Label(enfant, text="Nom :").grid(column=0, row=2, sticky='e')
553             nom = Entry(enfant)
554             nom.grid(column=1, row=2, sticky='w')
555             # Quantité
556             Label(enfant, text="Quantité :").grid(column=0, row=3, sticky='e')
557             quantite = Entry(enfant)
558             quantite.grid(column=1, row=3, sticky='w')
559             # Prix à l'unité
560             Label(enfant, text="Prix à l'unité :").grid(column=0, row=4, sticky='e')
561             prix = Entry(enfant)
562             prix.grid(column=1, row=4, sticky='w')
563 
564             def ___viderChamps() -> None:
565                 """Vide tout les champs de leur contenu"""
566                 # On récupère toutes les 'Entry' de la fenêtre et on change leur contenu
567                 for champ in [widget for typeElement, widget in enfant.children.items() if "entry" in
     typeElement]:
568                     champ.delete(0, "end")
569                     champ.update()
570 
571             # Boutons
572             Button(enfant, text="Valider", command=___verification).grid(column=0, row=8, columnspan
    =3, sticky='w')
573             Button(enfant, text="Vider les champs", command=___viderChamps).grid(column=0, row=8,
    columnspan=3)
574             Button(enfant, text="Quitter", command=enfant.destroy).grid(column=0, row=8, columnspan
    =3, sticky='e')
575 
576         Button(self.f, text="Ajouter un élément\nau stock", font=self.font, command=
    __ajouterElementStock).grid(column=1, row=2)
577 
578         # -> Partie export des statistiques
579         def __exportation() -> None:
580             """Exporte dans un fichier choisie par l'utilisateur ses statistiques de la journée."""
581             chemin = asksaveasfile(title=f"Exportation des statistiques de {caissier['nom']} {
    caissier['prenom']}  {self.nomApp}", filetypes=[("Fichier CSV", ".csv")])
582             if chemin == None: # si rien n'a été spécifie on arrête l'exportation
583                 return
584             Stats().exporteCSV(chemin.name, id)
585         Button(self.f, text="Exporter les statistiques", font=self.font, command=__exportation).grid(
    column=0, row=2, sticky='e', padx=ecart)
586 
587         Button(self.f, text="Se déconnecter", font=self.font, command=self._interfaceConnexion).grid(
    column=0, row=2, sticky='w', padx=ecart)
588 
589         # -> Boutton pour passer en mode manager si la personne est un manager
590         if caissier["metier"] == 0:
591             Button(self.f, text="Passer en mode Manager", font=self.font, command=lambda: self.
    _interfaceManager(id)).grid(column=0, row=2, sticky='w', padx=220)
592 
593     def _interfaceManager(self, id: int) -> None:
594         """Affiche l'interface du manager."""
595         manager = Utilisateurs().recuperationUtilisateur(id=id)
596         # Dans le cas où un utilisateur réussi à trouvé cette interface alors qu'il n'a pas le droit,
     il sera bloqué
597         if manager["metier"] != 0:
598             showerror(f"Erreur  {self.nomApp}", "Vous ne pouvez pas accéder à cette interface.")
599             return
600         self.parent.title(f"Manager {manager['nom']} {manager['prenom']}  {self.nomApp}")
601         self.dimensionsFenetre(self.parent, 580, 310)
602 
603         # Suppresssion de la dernière Frame
604         self.f.destroy()
605         # Instanciation d'une nouvelle Frame, on va donc ajouter tout nos widgets à cet Frame
606         self.f = Frame(self.parent)
607         self.f.grid()
```

```
608
609             Label(self.f, text="Interface Manager", font=(self.font[0], 20)).grid(column=0, row=0)
610
611             Button(self.f, text="Se déconnecter", font=self.font, command=self._interfaceConnexion).grid(
        column=1, row=0, padx=50)
612
613             Label(self.f).grid(row = 1, pady=10) # séparateur
614
615             def __ajouterUtilisateur(metier: int) -> None:
616                 """Permet de créer un nouvel utilisateur, manager (‘metier = 0‘) et caissier (‘metier =
        1‘)."""
617                 """
618                 L’enfant (‘TopLevel‘) dépend de la ‘Frame‘ et non du parent (‘Tk‘)
619                 pour éviter de resté ouverte meme lorsque le manager se déconnecte.
620                 """
621                 enfant = Toplevel(self.f)
622                 enfant.resizable(False, False)
623                 enfant.title(f"Ajouter un {’manager’ if metier == 0 else ’caissier’}  {self.nomApp}")
624
625                 def ___verification() -> None:
626                     """Vérifie si les champs renseignées sont valides."""
627                     """
628                     Les valeurs ‘Entry‘ qui ne sont pas passés seront dans
629                     la liste ‘mauvaisChamps‘.
630                     Si la liste ‘mauvaisChamps‘ contient un élément alors un test n’est pas ok.
631                     """
632                     mauvaisChamps = []
633                     # vérification pour le nom d’utilisateur
634                     if self.utilisateurCorrect(pseudo.get())[0] == False or Utilisateurs().
        utilisateurExistant(pseudo.get()) == True:
635                         mauvaisChamps.append(pseudo)
636                     # vérification pour le mot de passe
637                     if self.motDePasseCorrect(passe.get())[0] == False:
638                         mauvaisChamps.append(passe)
639                     # vérification pour le nom
640                     if self.nomCorrect(nom.get()) == False:
641                         mauvaisChamps.append(nom)
642                     # vérification pour le prénom
643                     if self.prenomCorrect(prenom.get()) == False:
644                         mauvaisChamps.append(prenom)
645                     # vérification pour la date de naissance
646                     if self.naissanceCorrect(naissance.get()) == False:
647                         mauvaisChamps.append(naissance)
648                     # vérification pour l’adresse
649                     if self.adresseCorrect(adresse.get()) == False:
650                         mauvaisChamps.append(adresse)
651                     # vérification pour le code postal
652                     if self.postalCorrect(postal.get()) == False:
653                         mauvaisChamps.append(postal)
654
655                     if len(mauvaisChamps) != 0:
656                         """
657                         Tous les champs qui n’ont pas réunies les conditions nécéssaires
658                         sont mis en orange pendant 3 secondes pour bien comprendre quelles champs
659                         sont à modifié.
660
661                         La fonction lambda ‘remettreCouleur‘ permet de remettre la couleur initial
662                         après les 3 secondes.
663                         """
664                         remettreCouleur = lambda widget, ancienneCouleur: widget.configure(bg=
        ancienneCouleur)
665                         for champs in mauvaisChamps:
666                             couleur = champs["background"] # couleur d’avant changement
667                             champs.configure(bg="orange") # on change la couleur du champs en orange
668                             # dans 3 secondes on fait : ‘remettreCouleur(champs, couleur)‘
669                             champs.after(3000, remettreCouleur, champs, couleur)
670                     else:
671                         # Tous les tests sont passés, on peut ajouter l’utilisateur à la base de donnée
672                         Utilisateurs().ajoutUtilisateur(
673                             pseudo.get(),
674                             passe.get().strip(),
675                             metier,
676                             nom.get(),
677                             prenom.get(),
678                             naissance.get(),
679                             adresse.get(),
680                             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
681                         )
682                         __ajouterUtilisateursListe(listeUtilisateurs) # met à jour la liste
683
684                 # Champs de saisie
685                 # Nom d’utilisateurs
686                 Label(enfant, text="Nom d’utilisateur :").grid(column=0, row=0, sticky=’e’)
```

```
687             Label(enfant, text="Pas de caractères spéciaux", font=("Arial", 10, "italic")).grid(
    column=2, row=0, sticky='w')
688             pseudo = Entry(enfant)
689             pseudo.grid(column=1, row=0, sticky='w')
690             # Mot de passe
691             Label(enfant, text="Mot de passe :").grid(column=0, row=1, sticky='e')
692             Label(enfant, text="1 majuscule, miniscule et caractère spécial minimum", font=("Arial",
    10, "italic")).grid(column=2, row=1, sticky='w')
693             passe = Entry(enfant)
694             passe.grid(column=1, row=1, sticky='w')
695             # Nom
696             Label(enfant, text="Nom :").grid(column=0, row=2, sticky='e')
697             Label(enfant, text="Pas de caractères spéciaux", font=("Arial", 10, "italic")).grid(
    column=2, row=2, sticky='w')
698             nom = Entry(enfant)
699             nom.grid(column=1, row=2, sticky='w')
700             # Prénom
701             Label(enfant, text="Prénom :").grid(column=0, row=3, sticky='e')
702             Label(enfant, text="Pas de caractères spéciaux", font=("Arial", 10, "italic")).grid(
    column=2, row=3, sticky='w')
703             prenom = Entry(enfant)
704             prenom.grid(column=1, row=3, sticky='w')
705             # Date de naissance
706             Label(enfant, text="Date de naissance :").grid(column=0, row=4, sticky='e')
707             Label(enfant, text="Format : AAAA/MM/JJ", font=("Arial", 10, "italic")).grid(column=2,
    row=4, sticky='w')
708             naissance = Entry(enfant)
709             naissance.grid(column=1, row=4, sticky='w')
710             # Adresse
711             Label(enfant, text="Adresse").grid(column=0, row=5, sticky='e')
712             adresse = Entry(enfant)
713             adresse.grid(column=1, row=5, sticky='w')
714             # Code postal
715             Label(enfant, text="Code postal :").grid(column=0, row=6, sticky='e')
716             Label(enfant, text="5 chiffres", font=("Arial", 10, "italic")).grid(column=2, row=6,
    sticky='w')
717             postal = Entry(enfant)
718             postal.grid(column=1, row=6, sticky='w')
719 
720             def ___viderChamps() -> None:
721                 """Vide tout les champs de leur contenu"""
722                 # On récupère toutes les `Entry` de la fenêtre et on change leur contenu
723                 for champ in [widget for typeElement, widget in enfant.children.items() if "entry" in
     typeElement]:
724                     champ.delete(0, "end")
725                     champ.update()
726 
727             # Boutons
728             Button(enfant, text="Valider", command=___verification).grid(column=0, row=8, columnspan
    =3, sticky='w')
729             Button(enfant, text="Vider les champs", command=___viderChamps).grid(column=0, row=8,
    columnspan=3)
730             Button(enfant, text="Quitter", command=enfant.destroy).grid(column=0, row=8, columnspan
    =3, sticky='e')
731 
732         def __retirerUtilisateur(metier: int) -> None:
733             """Permet de supprimer un utilisateur existant, manager (`metier = 0`) et caissier (`
    metier = 1`)."""
734             enfant = Toplevel(self.f) # cf. l'explication dans `__ajouterUtilisateur`
735             enfant.resizable(False, False)
736             enfant.title(f"Retirer un {'manager' if metier == 0 else 'caissier'}  {self.nomApp}")
737 
738             # Liste des utilisateurs
739             Label(enfant, text=f"Liste des {'manager' if metier == 0 else 'caissier'}", font=self.
    font).grid(column=0, row=0) # titre
740             # On définit une barre pour pouvoir scroller dans la liste
741             scroll_retirer = Scrollbar(enfant)
742             scroll_retirer.grid(column=1, row=1, sticky="nse")
743             # On définit notre liste et on la lie à notre `Scrollbar`
744             listeUtilisateurs_retirer = Listbox(enfant, width=25, height=4, yscrollcommand=
    scroll_retirer.set)
745             scroll_retirer.config(command=listeUtilisateurs_retirer.yview) # scroll à la verticale
    dans notre liste
746             # On ajoute nos utilisateurs à notre liste
747             __ajouterUtilisateursListe(listeUtilisateurs_retirer, metier)
748             listeUtilisateurs_retirer.grid(column=0, row=1)
749             # On affiche l'utilisateur quand on double-clique dessus
750 
751             def ___suppressionUtilisateur() -> None:
752                 """Supprime l'utilisateur actuellement sélectionné dans la liste"""
753                 element = listeUtilisateurs_retirer.curselection()
754                 if len(element) == 0: # si aucun élément n'est selectionné
755                     showwarning(f"Attention  {self.nomApp}", "Aucun utilisateur n'a été selectionné."
    )
756                 else:
```

```
                    utilisateur = listeUtilisateurs_retirer.get(listeUtilisateurs_retirer.curselection()[0]).split('(')[0][:-1]
                    reponse = askyesno(f"Confirmation  {self.nomApp}", f"Voulez vous supprimer {utilisateur} ?")
                    if reponse == True:
                        Utilisateurs().suppressionUtilisateurs(utilisateur)
                        __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(f"Information  {self.nomApp}", 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(_) -> None:
        """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.resizable(False, False)
        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() -> None:
            """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"
                ]

                # On remplace tous les prix `None` par `0.` (au cas où il y est des valeurs vide dans le `CSV`)
                for date, prix in donnees.items():
                    if prix == None:
                        donnees[date] = 0.

                # On récupère la plus grosse vente
                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", 7))
                        ecart += 33

            histogramme = Canvas(frameSuivi, width=270, height=200)
            histogramme.grid()

            ___actualisationCanvas()

            Button(enfant, text="Quitter", command=enfant.destroy).grid(column=0, row=1, columnspan
    =2)

        Button(self.f, text="Ajouter un caissier", font=self.font, command=lambda:
    __ajouterUtilisateur(1)).grid(column=0, row=2)
        Button(self.f, text="Retirer un caissier", font=self.font, command=lambda:
    __retirerUtilisateur(1)).grid(column=1, row=2)

        Label(self.f).grid(row = 3, pady=10) # séparateur

        # Liste des utilisateurs
        managerVerif = IntVar(self.f) # filtre pour afficher ou non les managers dans la liste
        caissierVerif = IntVar(self.f) # filtre pour afficher ou non les caissiers ou non dans la
    liste

        caissierVerif.set(1) # par défaut on affiche que les caissiers

        def __ajouterUtilisateursListe(liste: Listbox, force: int = None) -> None:
            """
            Ajoute des utilisateurs à la liste du Manager.
                -> metier =  0 : manager uniquement
                -> metier =  1 : caissier uniquement
                -> metier =  2 : manager et caissier
            """
            liste.delete(0, "end") # vidé la liste des utilisateurs
            if force: # si `force` n'est pas `None`, alors on force l'utilisation d'un métier
                metier = force
            else: # sinon on fait une vérification normale en fonction des filtres de l'interface
    manager
                if managerVerif.get() == 1:
                    if caissierVerif.get() == 1:
                        metier = None # on affiche les 2
                    else:
                        metier = 0 # on affiche seulement les managers
                else:
                    metier = 1 # on affiche les caissiers
                    if caissierVerif.get() == 0: # rien est coché, on revient à la configuration par
    défaut (caissiers uniquement)
                        metier = 1
                        caissierVerif.set(1)

            if metier == None: # on ajoute tous les utilisateurs
                for idx, utilisateur in enumerate(Utilisateurs().listUtilisateurs()):
                    liste.insert(idx, f"{utilisateur[0]} ({'manager' if utilisateur[1] == 0 else '
    caissier'})")
            elif metier == 0: # on ajoute que les managers
                for idx, utilisateur in enumerate(Utilisateurs().listUtilisateurs()):
                    if utilisateur[1] == metier:
                        liste.insert(idx, f"{utilisateur[0]} ({'manager' if utilisateur[1] == 0 else
    'caissier'})")
            elif metier == 1: # on ajoute que les caissiers
                for idx, utilisateur in enumerate(Utilisateurs().listUtilisateurs()):
                    if utilisateur[1] == metier:
                        liste.insert(idx, f"{utilisateur[0]} ({'manager' if utilisateur[1] == 0 else
    'caissier'})")
            else: # ce cas est là au cas où mais n'est pas sensé être appellé
                raise NameError("Métier inconnu.")

        # Label d'information
        Label(self.f, text="""
                            Double-cliquez sur un
```

```
                            utilisateur de la liste
                            pour obtenir des informations
                            supplémentaire à son sujet.
                            """, justify="right").grid(column=1, row=4, rowspan=2, sticky="e")

        Label(self.f, text="Liste des utilisateurs", font=self.font).grid(column=0, row=4) # titre
        # On définit une barre pour pouvoir scroller dans la liste
        scroll = Scrollbar(self.f)
        scroll.grid(column=0, row=5, sticky="nse")
        # On définit notre liste et on la lie à notre 'Scrollbar'
        listeUtilisateurs = Listbox(self.f, width=25, height=4, yscrollcommand=scroll.set)
        scroll.config(command=listeUtilisateurs.yview) # scroll à la verticale dans notre liste
        # On ajoute nos utilisateurs à notre liste
        __ajouterUtilisateursListe(listeUtilisateurs)
        listeUtilisateurs.grid(column=0, row=5)
        listeUtilisateurs.bind('<Double-Button>', __afficherInformationsUtilisateur) # on affiche l'
    utilisateur quand on double-clique dessus

        # Filtre pour la liste
        Label(self.f, text="Filtre", font=self.font).grid(column=1, row=4, sticky='w', padx=10) #
    titre
        filtres = Frame(self.f) # Morceau qui va contenir nos checkbutton
        filtres.grid(column=1, row=4, rowspan=2, sticky='w')
        Checkbutton(filtres, text="Manager", variable=managerVerif, command=lambda:
    __ajouterUtilisateursListe(listeUtilisateurs)).grid(sticky='w')
        Checkbutton(filtres, text="Caissier", variable=caissierVerif, command=lambda:
    __ajouterUtilisateursListe(listeUtilisateurs)).grid(sticky='w')

        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)

if __name__ == "__main__":
    """Application "GesMag" pour le module de Programmation d'interfaces (2021-2022)"""
    """
    Si presentation = True alors une base de donnée par défaut sera généré.
    Si presentation = False ou n'est même pas mentionné, alors aucune base de donnée par défaut ne
     sera généré.
    """
    GesMag(presentation = True).demarrer()
```

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

```
import sqlite3

class BaseDeDonnees:
    """Gère la base de donnée."""
    def __init__(self, urlBaseDeDonnee: str):
        self.connexion = self.creerConnexion(urlBaseDeDonnee)

    def creerConnexion(self, path: str):
        """Connexion à une base de donnée SQLite."""
        if not self.fichierExiste(path): # si l base de donnée n'existe pas
            open(path, 'x') # on la créer
        try:
            connnexion = sqlite3.connect(path)
        except sqlite3.Error as e:
            print(e) # on affiche l'erreur
            connnexion = None # et renvoie None
        return connnexion

    def fichierExiste(self, path: str) -> bool:
        """Vérifie qu'un fichier existe."""
        try: # on essaie d'ouvrir le fichier
            open(path, 'r')
        except FileNotFoundError: # si le fichier n'existe pas
            return False
        else: # si le fichier existe
            return True

    def requete(self, requete: str, valeurs = None) -> tuple:
        """Envois une requête vers la base de données."""
        try:
            curseur = self.connexion.cursor()
            if valeurs: # s'il y a des valeurs alors on lance la commande 'execute' avec ses
    dernières
                if type(valeurs) not in [list, tuple]: # si la valeur c'est juste une chaîne de
    charactère (par exemple), alors la converti en liste
                    valeurs = [valeurs]
                curseur.execute(requete, valeurs)
            else: # sinon on lance juste la requête
                curseur.execute(requete)
            self.connexion.commit() # applique les changements à la base de donnée
```

```
            return (curseur, curseur.lastrowid) # renvoie le curseur et l'ID de l'élément modifié
        except sqlite3.Error as e: # s'il y a eu une erreur SQLite
            print(e)

    def affichageResultat(self, curseur: tuple) -> list:
        """Affiche le résultat d'une requête."""
        tableau = []
        if curseur == None: # si le curseur est vide il n'y a rien a affiché (tableau vide)
            return tableau
        lignes = curseur[0].fetchall() # sinon on récupère les éléments
        for ligne in lignes:
            tableau.append(ligne) # on les ajoute au tableau
        return tableau # on le renvoie

    def affichageResultatDictionnaire(self, cles, curseur: sqlite3.Cursor) -> dict:
        """
        Même but que `affichageResultat()` mais avec
        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();"))
```

```
49
50      def suppressionUtilisateurs(self, pseudo: str) -> None:
51          """Supprime un utilisateur."""
52          requete = """
53                    DELETE FROM utilisateurs
54                    WHERE pseudo = ?
55                    """
56          self.requete(requete, pseudo)
57
58      def verificationIdentifiants(self, pseudo: str, motDePasse: str) -> tuple:
59          """
60          Retourne l'ID de l'utilisateur si trouvé dans la base de donnée ainsi
61          que son métier (`tuple`), sinon renvoie `(0,)`.
62          """
63          requete = """
64                    SELECT id, metier FROM utilisateurs
65                    WHERE pseudo = ? AND passe = ?
66                    """
67          reponseBaseDeDonnee = self.affichageResultat(self.requete(requete, [pseudo.lower(),
        motDePasse]))
68          if len(reponseBaseDeDonnee) == 0: # si les identifiants renseignés sont mauvais
69              return (0,)
70          return reponseBaseDeDonnee[0]
71
72      def listUtilisateurs(self) -> list:
73          """Retourne la liste des nom d'utilisateurs (avec leur métier)."""
74          requete = """
75                    SELECT pseudo, metier FROM utilisateurs
76                    """
77          return self.affichageResultat(self.requete(requete))
78
79      def recuperationUtilisateur(self, id: int = None, pseudo: str = None) -> dict:
80          """Retourne les informations d'un utilisateur grâce à son ID ou son pseudo (ID en priorité).
        """
81          recuperation = [
82              "id",
83              "pseudo",
84              "passe",
85              "metier",
86              "nom",
87              "prenom",
88              "naissance",
89              "adresse",
90              "postal",
91          ]
92          if not id: # si la variable `id` n'est pas définie
93              if not pseudo: # si seul la variable `pseudo` n'est pas définie
94                  raise ValueError # Aucun utilisateur renseigné
95              else: # si un pseudo est renseigné, c'est ce qu'on va utilisé
96                  requete = f"""
97                            SELECT {", ".join(recuperation)} FROM utilisateurs
98                            WHERE pseudo = ?
99                            """
100             utilisateur = pseudo
101         else: # si un id est renseigné, c'est ce qu'on va utilisé
102             requete = f"""
103                        SELECT {", ".join(recuperation)} FROM utilisateurs
104                        WHERE id = ?
105                        """
106             utilisateur = id
107         return self.affichageResultatDictionnaire(recuperation, self.requete(requete, utilisateur))
108
109     def utilisateurExistant(self, utilisateur: str) -> bool:
110         """Vérifie si l'utilisateur donnée existe déjà dans la base de donnée."""
111         requete = """
112                   SELECT EXISTS (
113                       SELECT 1 FROM utilisateurs
114                       WHERE pseudo = ?
115                   )
116                   """
117         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

```
1   from random import randint, uniform
2
3   from db import BaseDeDonnees
4
5   class Stock(BaseDeDonnees):
6       """Gère une table "stock" pour une base de donnée donné."""
7       def __init__(self):
```

```
8           super().__init__(r"db.sqlite3") # connexion à la base de donnée
9
10      def creationTable(self, presentation: bool = False) -> None:
11          """Créer la table qui stocker les stocks."""
12          requete = """
13                    CREATE TABLE IF NOT EXISTS stocks (
14                        id INTEGER PRIMARY KEY,
15                        type TEXT,
16                        nom TEXT,
17                        quantite INTEGER,
18                        prix REAL,
19                        image_url TEXT
20                    );
21                    """
22          self.requete(requete)
23          # Ajout d'un stock par défaut si aucun stock n'existe dans la base de donnée
24          if len(self.listeStocks()) == 0 and presentation:
25              # Créer un dictionnaire d'éléments pour mieux voir ce que l'on ajoute à la base de donnée
26              defaut = {
27                  "fruits legumes": [
28                      ("banane", "img/banane.gif"),
29                      ("orange", "img/orange.gif"),
30                      ("betterave", "img/betterave.gif"),
31                      ("carottes", "img/carottes.gif"),
32                      ("tomates", "img/tomates.gif"),
33                      ("citron", "img/citron.gif"),
34                      ("kiwi", "img/kiwi.gif"),
35                      ("clementine", "img/clementine.gif"),
36                      ("pomme", "img/pomme.gif"),
37                      ("avocat", "img/avocat.gif")
38                  ],
39                  "boulangerie": [
40                      ("brownie", "img/brownie.gif"),
41                      ("baguette", "img/baguette.gif"),
42                      ("pain au chocolat", "img/pain_au_chocolat.gif"),
43                      ("croissant", "img/croissant.gif"),
44                      ("macaron", "img/macaron.gif"),
45                      ("millefeuille", "img/millefeuille.gif"),
46                      ("paris-brest", "img/paris-brest.gif"),
47                      ("opera", "img/opera.gif"),
48                      ("fraisier", "img/fraisier.gif"),
49                      ("eclair", "img/eclair.gif")
50                  ],
51                  "boucherie poissonnerie": [
52                      ("saucisson", "img/saucisson.gif"),
53                      ("côte de boeuf", "img/cote_de_boeuf.gif"),
54                      ("langue de boeuf", "img/langue_de_boeuf.gif"),
55                      ("collier de boeuf", "img/collier_de_boeuf.gif"),
56                      ("entrecote", "img/entrecote.gif"),
57                      ("cabillaud", "img/cabillaud.gif"),
58                      ("saumon", "img/saumon.gif"),
59                      ("colin", "img/colin.gif"),
60                      ("bar", "img/bar.gif"),
61                      ("dorade", "img/dorade.gif")
62                  ],
63                  "entretien": [
64                      ("nettoyant air comprimé", "img/nettoyant_air_comprime.gif"),
65                      ("nettoyage anti-bactérien", "img/nettoyage_anti-bacterien.gif"),
66                      ("nettoyant pour écran", "img/nettoyant_pour_ecran.gif"),
67                      ("nettoyant pour lunettes", "img/nettoyant_pour_lunettes.gif"),
68                      ("pioche", "img/pioche.gif"),
69                      ("pelle", "img/pelle.gif"),
70                      ("lampe torche", "img/lampe_torche.gif"),
71                      ("gants", "img/gants.gif"),
72                      ("éponge", "img/eponge.gif"),
73                      ("essuie-tout", "img/essuie-tout.gif")
74                  ]
75              }
76
77              # Ajoute le dictionnaire précédemment créer à la base de donnée avec un prix et une
    quantité aléatoire
78              for type in defaut:
79                  for element in defaut[type]:
80                      self.ajoutStock(type, element[0], randint(0, 10), round(uniform(2., 30.), 2),
    element[1])
81
82      def ajoutStock(self, typeElement: str, nom: str, quantite: int, prix: float, imageURL: str) ->
    list:
83          """Ajoute un élément dans le stock et retourne l'ID de ce dernier."""
84          requete = """
85                    INSERT INTO stocks (
86                        type, nom, quantite, prix, image_url
87                    ) VALUES (
88                        ?, ?, ?, ?, ?
89                    );
```

```
                    """
        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
```

```
20
21      def creationCSV(self, force: bool = False) -> None:
22          """
23          Créer le fichier ‘CSV‘ qui stockera les statistiques pour tous les utilisateurs.
24
25          Possibilité de forcer la création (c-à-d même si le fichier existe déjà) en renseignant
26          ‘force = True‘
27          """
28          if not Utilisateurs().fichierExiste("stats.csv") or force:
29              with open("stats.csv", 'w') as f:
30                  fichier = csv.writer(f)
31                  fichier.writerow(["id", "pseudo"] + self.datesDisponibles())
32
33      def miseAJourStatsUtilisateur(self, utilisateurID: int, prix: float) -> None:
34          """
35          Récupère le prix d'une transaction et l'ajoute au total d'un utilisateur.
36
37          - s'il y a déjà une valeur dans la base de donnée correspondant à la date du jour,
38            on met à jour cette valeur en l'additionnant avec le nouveaux prix
39          - s'il n'avait pas de valeur à cette date:
40              - si l'utilisateur n'est pas dans le fichier, on rajoute une ligne avec le prix
41              - si l'utilisateur est présent mais aucun prix n'est fixé pour la date du jour,
42                on rajoute le prix sur la ligne de l'utilisateur déjà existante
43
44          On remplie les espaces vides par la valeur ‘0‘ (car aucun chiffre n'a été fait ce jour là,
45          car aucune information n'était renseignée).
46          """
47          self.miseAJourDatesCSV() # met-à-jour les dates du fichier ‘CSV‘
48
49          # Mets-à-jour le ‘CSV‘ avec le nouveau prix...
50          aujourdHui = date.today().strftime(self.formatDate)
51          with open("stats.csv", 'r') as f:
52              fichier = list(csv.reader(f))
53              # On récupère la colonne pour aujourd'hui
54              index = 0
55              locationDate = None # note l'index de la colonne de la date dans le fichier
56              for nomColonne in fichier[0]: # on regarde l'entente
57                  if nomColonne == aujourdHui: # on regarde si la colonne correspond à la date du jour
58                      locationDate = index # on note l'entête
59                  index += 1
60              if locationDate == None: # ne devrait pas arrivé car on mets à jour les dates du ‘CSV‘
    avant de lire le fichier
61                  raise IndexError("Date du jour non trouvé dans le fichier csv.")
62
63              utilisateur = Utilisateurs().recuperationUtilisateur(utilisateurID) # on récupère les
    infos de l'utilisateur
64              # Vérification si l'utilisateur est déjà présent dans le fichier ‘CSV‘
65              locationUtilisateur = None # note l'index de la ligne de l'utilisateur dans le fichier
66              for idx, location in enumerate(fichier):
67                  if location[0] == str(utilisateurID):
68                      locationUtilisateur = idx
69              if locationUtilisateur == None: # si l'utilisateur n'est pas présent dans le fichier
70                  # on rajoute la ligne
71                  fichier += [[utilisateurID, utilisateur["pseudo"]] + ['0' for _ in range(0,
    locationDate - 2)] + [prix]]
72              else: # si déjà présent dans le fichier
73                  try:
74                      ancienPrix = float(fichier[locationUtilisateur][locationDate]) # on récupère l'
    ancien prix
75                  except IndexError: # si il n'y avait pas de prix définie avant
76                      ancienPrix = 0
77                      # On rajoute la case
78                      fichier[locationUtilisateur] += ['0' for _ in range(0, locationDate - 1)]
79                  ancienPrix += prix # on y ajoute le nouveaux prix
80                  fichier[locationUtilisateur][locationDate] = f"{float(ancienPrix):.2f}" # on met à
    jour le fichier
81
82          with open("stats.csv", 'w') as f: # on applique les changements
83              ecriture = csv.writer(f)
84              ecriture.writerows(fichier)
85
86
87      def exporteCSV(self, chemin: str, utilisateurID: int) -> None:
88          """
89          Exporte les statistiques d'un utilisateur dans un fichier ‘CSV‘.
90          - N'exporte que les statistiques du jour.
91          """
92          donnees = self.recuperationDonneesCSV(utilisateurID)
93          aujourdHui = date.today().strftime(self.formatDate)
94          with open(chemin, 'w') as f:
95              fichier = csv.writer(f)
96              fichier.writerow(["ID Utilisateur", f"Totales des ventes du jour ({aujourdHui})"])
97              if len(donnees) > 0: # si il y a des données enregistrées
98                  fichier.writerow([utilisateurID, donnees[aujourdHui]])
99              else:
```

```
100                 fichier.writerow([utilisateurID, "Aucune ventes enregistrée"])
101
102     def recuperationDonneesCSV(self, utilisateurID: int) -> dict:
103         """Renvoie les informations contenu dans le fichier ‘CSV‘ globale."""
104         self.miseAJourDatesCSV() # met à jour les dates du fichier ‘CSV‘
105         with open("stats.csv", 'r') as f:
106             fichier = list(csv.DictReader(f)) # lecture du fichier sous forme d'une liste de
    dictionnaire
107             for utilisateur in fichier: # on regarde tous les utilisateurs stockés dans le fichier
108                 if utilisateur["id"] == str(utilisateurID): # si utilisateur trouvé
109                     return utilisateur # renvoie des infos de l'utilisateur
110         return {} # ne retourne rien si l'utilisateur n'était pas présent dans le fichier
111
112     def miseAJourDatesCSV(self) -> None:
113         """
114         Mets-à-jour les dates trop anciennes du fichier globales ‘CSV‘.
115
116         On remplie les espaces vides par la valeur ‘0‘ pour les jours qui remplacent les dates
117         trop vieilles (âgées de plus d'une semaine) car soit aucun chiffre n'a été fait ce jour là,
118         car aucune information n'était renseignée.
119         """
120         besoinDeMofication = False
121         with open("stats.csv", 'r') as f:
122             fichier = list(csv.reader(f))
123             if len(fichier) == 1: # si fichier ne contient que l'entête
124                 self.creationCSV(True) # on recréer le fichier dans le doute (avec les bonnes dates)
125             else:
126                 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])
127                 mauvaisIndex = [] # liste qui va stocker les index trop vieux (> 1 semaine)
128                 datesPresentes = [] # liste qui stock les dates valides présentes dans le fichier (<
    1 semaine)
129                 datesDisponibles = self.datesDisponibles() # liste des bonnes dates
130                 for dateFichier in fichier[0][index:]: # on regarde toutes les dates du fichier
131                     if dateFichier not in datesDisponibles: # si trop vieux
132                         mauvaisIndex.append(index) # on ajoute l'index à la liste
133                         besoinDeMofication = True
134                     else:
135                         datesPresentes.append(dateFichier) # on ajoute la date à la liste
136                     index += 1
137
138                 if not besoinDeMofication: # vérification si on a besoin de rien faire
139                     return # on quitte la fonction
140
141                 datesARajouter = [date for date in datesDisponibles if date not in datesPresentes] #
    liste des dates à rajouter
142                 if len(datesARajouter) != len(mauvaisIndex): # ne devrais pas arrivé mais on sait
    jamais
143                     raise IndexError("Problème, pas autant de dates à rajouter que de de dates
    périmés dans le fichier.")
144                 for idx in mauvaisIndex: # pour tous les mauvais index
145                     for numLigne, ligne in enumerate(fichier): # on regarde toutes les lignes du
    fichier
146                         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
147                             if numLigne == 0: # si c'est la ligne d'entête
148                                 ligne[idx] = datesARajouter[0] # on change la ligne avec la nouvelle
    date
149                                 datesARajouter.pop(0)
150                             else: # si c'est une ligne de donnée
151                                 ligne[idx] = '0' # on change la ligne avec une valeur vide
152                             fichier[numLigne] = ligne # on applique les changements
153         if besoinDeMofication: # vérification si on a besoin de faire des changements
154             with open("stats.csv", 'w') as f: # on applique les changements
155                 ecriture = csv.writer(f)
156                 ecriture.writerows(fichier)
```