

# Projet final Tkinter

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

Année universitaire 2021-2022

## Table des matières

<b>1</b>	<b>Consigne</b>	<b>2</b>
1.1	Dépendances . . . . .	2
1.2	Cahier des charges . . . . .	2
<b>2</b>	<b>Code</b>	<b>4</b>
2.1	main.py . . . . .	4
2.2	db.py . . . . .	16
2.3	users.py . . . . .	17
2.4	stock.py . . . . .	18
2.5	stats.py . . . . .	20

*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 ⇒ **lisibilité accru pour les champs mal renseignés, l'ID n'est pas à renseigné car assigné automatiquement 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 désélectionner (caissier par défaut)**
  - Un histogramme présentant l'évolution des sommes totales des ventes journalières de la semaine passée d'un utilisateur ⇒ **accessible au double-clic 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 été 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

```
1 # Tkinter
2 from tkinter import Canvas, IntVar, Checkbutton, LabelFrame, PhotoImage, Scrollbar, Listbox, Entry,
   Button, Label, Frame, Tk, Toplevel
3 from tkinter.ttk import Combobox, Separator
4 from tkinter.messagebox import showerror, showinfo, showwarning, askyesno
5 from tkinter.filedialog import askopenfile, asksaveasfile
6 # Regex
7 from re import sub
8 # Date
9 from datetime import date
10
11 # Import des fichiers pour gérer la base de donnée et l'export en CSV
12 from users import Utilisateurs
13 from stock import Stock
14 from stats import Stats
15
16 class GesMag:
17     """Programme de Gestion d'une caisse de magasin."""
18     def __init__(self, presentation: bool = False) -> None:
19         """Instancie quelques variables pour plus de clarté."""
20         Utilisateurs().creationTable(presentation) # on créer la table utilisateurs si elle n'existe
   pas déjà
21         Stock().creationTable(presentation) # on créer la table du stock si elle n'existe pas déjà
22         Stats().creationCSV() # on créer le fichier CSV qui stockera les statistiques des
   utilisateurs
23
24         self.nomApp = "GesMag" # nom de l'application
25         self.parent = Tk() # fenêtre affichée à l'utilisateur
26         self.parent.resizable(False, False) # empêche la fenêtre d'être redimensionnée
27         self.f = Frame(self.parent) # 'Frame' "principale" affichée à l'écran
28         self.tableau = Frame() # 'Frame' qui va afficher le tableau des éléments présents dans le
   stock
29         self.imagesStock = [] # liste qui va contenir nos images pour l'affichage du stock
30         self.dossierImage = PhotoImage(file = "img/dossier.gif") # image pour l'icone de selection
31         self.panierAffichage = Frame() # 'Frame' qui va afficher le panier
32         self.panier = [] # liste des éléments "dans le panier"
33
34     def demarrer(self) -> None:
35         """Lance le programme GesMag."""
36         self.font = ("Comfortaa", 14) # police par défaut
37
38         self._interfaceConnexion() # on créer la variable 'self.f' qui est la frame a affiché
39         self.f.grid() # on affiche la frame
40
41         self.parent.mainloop() # on affiche la fenêtre
42
43     def motDePasseCorrect(self, motDPasse: str) -> tuple:
44         """Détermine si un mot de passe suit la politique du programme ou non."""
45         if len(motDPasse) == 0: # si le champs est vide
46             return (False, "Mot de passe incorrect.")
47         if len(motDPasse) < 8: # si le mot de passe est plus petit que 8 caractères
48             return (False, "Un mot de passe doit faire 8 caractères minimum.")
49         """
50         - Pour le regex, la fonction 'sub' élimine tout ce qui est donné en fonction
51           du pattern renseigné, alors si la fonction 'sub' renvoie pas exactement
52           la même chaîne de caractère alors c'est qu'il y avait un caractère interdit.
53         - J'utilises pas 'match' parce que je suis plus à l'aise avec 'sub'.
54         """
55         if not sub(r"[A-Z]", '', motDPasse) != motDPasse:
56             return (False, "Un mot de passe doit au moins contenir une lettre majuscule.")
57         if not sub(r"[a-z]", '', motDPasse) != motDPasse:
58             return (False, "Un mot de passe doit au moins contenir une lettre minuscule.")
59         if not sub(r" *?[\^w\s]+", '', motDPasse) != motDPasse:
60             return (False, "Un mot de passe doit au moins contenir un caractère spécial.")
61
62         return (True,) # si aucun des tests précédents n'est valide, alors le mot de passe est valide
63
64     def utilisateurCorrect(self, utilisateur: str) -> tuple:
65         """Détermine si un nom d'utilisateur suit la politique du programme ou non."""
66         """
67         Pour le nom d'utilisateur on vérifie si le champs n'est pas vide
68         et si il y a bien que des lettres et des chiffres.
69         """
70         if len(utilisateur) == 0:
71             return (False, "Utilisateur incorrect.")
72         if sub(r" *?[\^w\s]+", '', utilisateur) != utilisateur:
73             return (False, "Un nom d'utilisateur ne doit pas contenir de caractère spécial.")
74         return (True,)
75
```

```

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[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("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) -> None:
143     """Permet de changer les dimensions de la fenêtre parent et la place au centre de l'écran."""
144     largeur = fenetre.wininfo_screenwidth()
145     hauteur = fenetre.wininfo_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"{{fenetre}} de connexion {{self.nomApp}}")
157
158     # Suppression 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     """
181     if self.mdpVisible == False: # si mot de passe caché, alors on l'affiche
182         self.mdpVisible = True
183         motDpasse.config(show='')
184         boutonAffichageMDP.config(font=("Arial", 10, "overstrike"))
185     else: # inversement
186         self.mdpVisible = False
187         motDpasse.config(show='')
188         boutonAffichageMDP.config(font=("Arial", 10))
189
190 boutonAffichageMDP = Button(self.f, text='', command=lambda: __afficherMDP(self))
191 boutonAffichageMDP.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     # Suppression 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
232     # Cache un certain type de produit
233     fruitsLegumesVerif = IntVar(stock)
234     boulangerieVerif = IntVar(stock)
235     boucheriePoissonnerieVerif = IntVar(stock)
236     entretienVerif = IntVar(stock)
237
238     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("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 définit 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

```

```

310 # Supprime toutes les valeurs 'None' de la liste
311 stockListe = list(filter(None, stockListe))
312
313 ecart = 10 # écart entre les champs
314
315 elementsParPage = 10 # on définit combien d'élément une page peut afficher au maximum
316 pageMax = -(len(stockListe) // elementsParPage) # on définit combien de page il y a au
maximum
317
318 if pageMax <= 1:
319     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
320
321 limiteIndex = elementsParPage * page # on définit une limite pour ne pas afficher plus d'
éléments qu'il n'en faut par page
322 if len(stockListe) > 0: # si stock non vide
323     # Définition des colonnes
324     Label(self.tableau, text="ID").grid(column=0, row=0, padx=ecart)
325     Label(self.tableau, text="Image").grid(column=1, row=0, padx=ecart)
326     Label(self.tableau, text="Type").grid(column=2, row=0, padx=ecart)
327     Label(self.tableau, text="Nom").grid(column=3, row=0, padx=ecart)
328     Label(self.tableau, text="Quantité").grid(column=4, row=0, padx=ecart)
329     Label(self.tableau, text="Prix unité").grid(column=5, row=0, padx=ecart)
330     Label(self.tableau, text="Action").grid(column=6, row=0, padx=ecart)
331     Separator(self.tableau).grid(column=0, row=0, colspan=7, sticky="sew")
332     Separator(self.tableau).grid(column=0, row=0, colspan=7, sticky="new")
333     for j in range(0, 8):
334         Separator(self.tableau, orient='vertical').grid(column=j, row=0, colspan=2,
sticky="nsw")
335
336     curseur = limiteIndex - elementsParPage # on commence à partir du curseur
337     i = 1 # on commence à 1 car il y a déjà le nom des colonnes en position 0
338     self.imagesStock = [] # on vide la liste si elle contient déjà des images
339     for element in stockListe[curseur:limiteIndex]: # on ignore les éléments avant le
curseur et après la limite
340         Label(self.tableau, text=element["id"]).grid(column=0, row=i, padx=ecart)
341
342         """
343         L'idée est que on a une liste 'images' qui permet de stocker toutes nos images
344         (c'est une limitation de tkinter que de garder nos images en mémoire)
345         Une fois ajouté à la liste, on l'affiche dans notre Label
346         """
347         if Stock().fichierExiste(element["image_url"]): # si l'image existe, utilisation
de la fonction de 'db.py'
348             self.imagesStock.append(PhotoImage(file = element["image_url"]))
349         else: # si l'image n'existe pas
350             self.imagesStock.append(PhotoImage(file = "img/default.gif")) # image par
défaut
351         Label(self.tableau, image=self.imagesStock[i - 1]).grid(column=1, row=i, padx=
ecart)
352
353         Label(self.tableau, text=element["type"].capitalize()).grid(column=2, row=i, padx
=ecart)
354         Label(self.tableau, text=element["nom"].capitalize()).grid(column=3, row=i, padx=
ecart)
355         Label(self.tableau, text=element["quantite"]).grid(column=4, row=i, padx=ecart)
356         Label(self.tableau, text=element["prix"]).grid(column=5, row=i,
padx=ecart)
357         # boutons d'actions pour le panier
358         Button(self.tableau, text='+', font=("Arial", 7, "bold"), command=lambda e =
element: __miseAJourPanier(e, True)).grid(column=6, row=i, sticky='n', padx=ecart)
359         Button(self.tableau, text='-', font=("Arial", 7, "bold"), command=lambda e =
element: __miseAJourPanier(e, False)).grid(column=6, row=i, sticky='s', pady=2)
360         for j in range(0, 8):
361             Separator(self.tableau, orient='vertical').grid(column=j, row=i, colspan
=2, sticky="nsw")
362             Separator(self.tableau).grid(column=j, row=i, colspan=2, sticky="sew")
363         curseur += 1
364         i += 1
365
366         # Information sur la page actuelle
367         Label(self.tableau, text=f"Page {page}/{pageMax}").grid(column=2, row=i, colspan
=3)
368
369         # Boutons
370         precedent = Button(self.tableau, text="Page précédente", command=lambda:
__affichageTableau(page - 1))
371         precedent.grid(column=0, row=i, colspan=2, sticky='w', padx=ecart, pady=ecart)
372         suivant = Button(self.tableau, text="Page suivante", command=lambda:
__affichageTableau(page + 1))
373         suivant.grid(column=5, row=i, colspan=2, sticky='e', padx=ecart)
374         if page == 1: # si on est à la première page on désactive le bouton précédent
375             precedent.config(state="disabled")
376         if page == pageMax: # si on est à la dernière page on désactive le bouton suivant
377             suivant.config(state="disabled")

```



```

378         else:
379             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)
380
381         __affichageTableau() # affichage du tableau
382
383         # -> Partie affichage du ticket de caisse
384         ecart = 10
385         ticket = LabelFrame(self.f, text="Ticket de caisse")
386         ticket.grid(column=1, row=1, sticky='n', padx=5)
387
388         Label(ticket, text=f"Date de vente : {date.today().strftime('%Y/%m/%d')}").grid(column=0, row
=0, pady=ecart)
389
390         def __affichagePanier() -> None:
391             """Affiche le panier actuel dans le ticket de caisse."""
392             self.panierAffichage.destroy()
393             self.panierAffichage = Frame(ticket)
394             self.panierAffichage.grid(column=0, row=1, pady=ecart)
395             elementsAchetes = Label(self.panierAffichage)
396             elementsAchetes.grid(column=0, columnspan=2)
397             prixTotal = 0
398             compteurElements = 0
399             for idx, element in enumerate(self.panier):
400                 Label(self.panierAffichage, text=f"[{element[0]['id']} -]").grid(column=0, row=idx +
1, sticky='e')
401                 if element[1] > 1:
402                     message = f"{element[1]}x {element[0]['nom'].capitalize()} ({__formatPrix(element
[0]['prix'])} | total: {__formatPrix(element[0]['prix'] * element[1])})"
403                     else:
404                         message = f"{element[1]}x {element[0]['nom'].capitalize()} ({__formatPrix(element
[0]['prix'])})"
405                     Label(self.panierAffichage, text=message).grid(column=1, row=idx + 1, sticky='w')
406                     prixTotal += (element[0]["prix"] * element[1]) # ajout du prix
407                     compteurElements += element[1]
408
409                     elementsAchetes.config(text=f"Élément{'s' if compteurElements > 1 else ''} acheté{'s' if
compteurElements > 1 else ''} ({compteurElements}) :")
410
411                 try: # désactive le bouton si rien n'est dans le panier
412                     if len(self.panier) <= 0:
413                         validationTicketDeCaisseBouton.config(state="disabled")
414                     else:
415                         validationTicketDeCaisseBouton.config(state="active")
416                 except NameError: # si pas renseigné, alors = panier vide, déjà désactiver
417                     pass
418
419                 Label(self.panierAffichage, text=f"Prix total : {__formatPrix(prixTotal)}").grid(column
=0, pady=ecart, columnspan=2)
420
421         __affichagePanier()
422
423         def __validationTicketDeCaisse() -> None:
424             """Lance plusieurs méthodes pour valider le ticket de caisse."""
425             # Met à jour la valeur dans le fichier 'CSV' (statistiques)
426             Stats().miseAJourStatsUtilisateur(id, sum([element[0]["prix"] * element[1] for element in
self.panier]))
427
428             # Informe l'utilisateur que tout est validé
429             showinfo("Validation", "Ticket de caisse validé !")
430
431             # Retire les éléments renseigné dans le panier du stock
432             for element in self.panier:
433                 Stock().reduitQuantiteStock(element[0]["id"], element[1])
434
435             # Remet le panier à 0
436             self.panier = []
437
438             # Met-à-jour le panier et le tableau du stock
439             __affichagePanier()
440             __affichageTableau()
441
442             validationTicketDeCaisseBouton = Button(ticket, text="Valider le\nticket de caisse", font=
self.font, command=__validationTicketDeCaisse, state="disabled")
443             validationTicketDeCaisseBouton.grid(column=0, pady=ecart)
444
445             # -> Partie ajout élément au stock
446             def __ajouterElementStock() -> None:
447                 """Ouvre une fenêtre qui permet d'ajouter un nouvel élément à la base de donnée."""
448                 ""
449                 L'enfant ('TopLevel') dépend de la 'Frame' et non du parent ('Tk')
450                 pour éviter de resté ouverte meme lorsque le caissier se déconnecte.
451                 ""
452                 enfant = Toplevel(self.f)
453                 enfant.title(f"Ajouter un élément au stock {self.nomApp}")

```

```

454
455 def __verification() -> None:
456     """Vérifie si les champs renseignés 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
474     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 nécessaires
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=
512     ancienneCouleur)
513     for champs in mauvaisChamps:
514         couleur = champs["background"] # couleur d'avant changement
515         champs.configure(bg="orange") # on change la couleur du champs en orange
516         # dans 3 secondes on fait : 'remettreCouleur(champs, couleur)'
517         champs.after(3000, remettreCouleur, champs, couleur)
518     else:
519         """
520         Tous les tests sont passés, on peut ajouter l'utilisateur à la base de donnée
521         Pas besoin de gérer les erreurs lors des casts car on a déjà vérifié que c'était
522         bien les bons types avant
523         """
524     Stock().ajoutStock(
525         typeElement.get(),
526         nom.get(),
527         int(quantite.get()),
528         float(prix.get()),
529         image.get()
530     )
531     __affichageTableau() # met à jour le tableau
532
533     # Champs de saisie
534     # Image
535     Label(enfant, text="Image :").grid(column=0, row=0, sticky='e')
536     image = Entry(enfant)
537     image.grid(column=1, row=0, sticky='w')
538     def __selectionImage() -> None:

```

```

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

```

```

608 Label(self.f, text="Interface Manager", font=(self.font[0], 20)).grid(column=0, row=0)
609
610 Button(self.f, text="Se déconnecter", font=self.font, command=self._interfaceConnexion).grid(
column=1, row=0, padx=50)
611
612 Label(self.f).grid(row = 1, pady=10) # séparateur
613
614 def __ajouterUtilisateur(metier: int) -> None:
615     """Permet de créer un nouvel utilisateur, manager ('metier = 0') et caissier ('metier =
1')."""
616     """
617     L'enfant ('TopLevel') dépend de la 'Frame' et non du parent ('Tk')
618     pour éviter de resté ouverte meme lorsque le manager se déconnecte.
619     """
620     enfant = Toplevel(self.f)
621     enfant.title(f"Ajouter un {'manager' if metier == 0 else 'caissier'} {self.nomApp}")
622
623     def __verification() -> None:
624         """Vérifie si les champs renseignées sont valides."""
625         """
626         Les valeurs 'Entry' qui ne sont pas passés seront dans
627         la liste 'mauvaisChamps'.
628         Si la liste 'mauvaisChamps' contient un élément alors un test n'est pas ok.
629         """
630         mauvaisChamps = []
631         # vérification pour le nom d'utilisateur
632         if self.utilisateurCorrect(pseudo.get())[0] == False or Utilisateurs().
utilisateurExistant(pseudo.get()) == True:
633             mauvaisChamps.append(pseudo)
634         # vérification pour le mot de passe
635         if self.motDePasseCorrect(passe.get())[0] == False:
636             mauvaisChamps.append(passe)
637         # vérification pour le nom
638         if self.nomCorrect(nom.get()) == False:
639             mauvaisChamps.append(nom)
640         # vérification pour le prénom
641         if self.prenomCorrect(prenom.get()) == False:
642             mauvaisChamps.append(prenom)
643         # vérification pour la date de naissance
644         if self.naissanceCorrect(naissance.get()) == False:
645             mauvaisChamps.append(naissance)
646         # vérification pour l'adresse
647         if self.adresseCorrect(adresse.get()) == False:
648             mauvaisChamps.append(adresse)
649         # vérification pour le code postal
650         if self.postalCorrect(postal.get()) == False:
651             mauvaisChamps.append(postal)
652
653         if len(mauvaisChamps) != 0:
654             """
655             Tous les champs qui n'ont pas réunies les conditions nécessaires
656             sont mis en orange pendant 3 secondes pour bien comprendre quelles champs
657             sont à modifié.
658
659             La fonction lambda 'remettreCouleur' permet de remettre la couleur initial
660             après les 3 secondes.
661             """
662             remettreCouleur = lambda widget, ancienneCouleur: widget.configure(bg=
ancienneCouleur)
663             for champs in mauvaisChamps:
664                 couleur = champs["background"] # couleur d'avant changement
665                 champs.configure(bg="orange") # on change la couleur du champs en orange
666                 # dans 3 secondes on fait : 'remettreCouleur(champs, couleur)'
667                 champs.after(3000, remettreCouleur, champs, couleur)
668             else:
669                 # Tous les tests sont passés, on peut ajouter l'utilisateur à la base de donnée
670                 Utilisateurs().ajoutUtilisateur(
671                     pseudo.get(),
672                     passe.get().strip(),
673                     metier,
674                     nom.get(),
675                     prenom.get(),
676                     naissance.get(),
677                     adresse.get(),
678                     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
679                 )
680                 __ajouterUtilisateursListe(listeUtilisateurs) # met à jour la liste
681
682             # Champs de saisie
683             # Nom d'utilisateurs
684             Label(enfant, text="Nom d'utilisateur :").grid(column=0, row=0, sticky='e')
685             Label(enfant, text="Pas de caractères spéciaux", font=("Arial", 10, "italic")).grid(
column=2, row=0, sticky='w')
686             pseudo = Entry(enfant)

```

```

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

```

```

758         __ajouterUtilisateursListe(listeUtilisateurs_retirer) # met à jour la liste
dans la fenêtre de suppression
759         __ajouterUtilisateursListe(listeUtilisateurs) # met à jour la liste dans l'
interface principale
760         showinfo("Information", f"Utilisateur {utilisateur} supprimé.")
761
762         # Boutons
763         Button(enfant, text="Supprimer", command=__suppressionUtilisateur).grid(column=0, row=8,
columnspan=3, sticky='w')
764         Button(enfant, text="Quitter", command=enfant.destroy).grid(column=0, row=8, columnspan
=3, sticky='e')
765
766     def __afficherInformationsUtilisateur(_) -> None:
767         """Permet d'afficher les informations d'un utilisateur"""
768         element = listeUtilisateurs.curselection()
769         if len(element) == 0: # si aucun élément n'est sélectionné
770             return
771         """
772         On split le champs car dans la liste on affiche le métier entre
773         parenthèses et on doit donner que le nom d'utilisateur à
774         la fonction 'recuperationUtilisateur', aussi on retire le dernier
775         caractère avec[:-1] car c'est un espace.
776         """
777         utilisateur = Utilisateurs().recuperationUtilisateur(pseudo=listeUtilisateurs.get(element
[0]).split('(')[0][:-1])
778         enfant = TopLevel(self.f) # cf. l'explication dans '__ajouterUtilisateur'
779         enfant.title(f"{utilisateur['nom']} {utilisateur['prenom']} {self.nomApp}")
780
781         # Informations sur l'utilisateur
782         frameInfos = LabelFrame(enfant, text="Informations utilisateur", font=self.font)
783         frameInfos.grid(column=0, row=0, sticky='n', padx=5)
784         utilisateur["metier"] = "Manager" if utilisateur["metier"] == 0 else "Caissier"
785         del utilisateur["passe"] # le manager ne doit pas connaître le mot de passe de l'
utilisateur
786         for idx, cle in enumerate(utilisateur):
787             if cle == "id": # on ignore l'ID
788                 continue
789             cleAffichage = cle.capitalize()
790             Label(frameInfos, text=f"{cleAffichage} :").grid(column=0, row=idx + 1, sticky='e')
791             Label(frameInfos, text=utilisateur[cle]).grid(column=1, row=idx + 1, sticky='w')
792
793         frameSuivi = LabelFrame(enfant, text="Histogramme des ventes", font=self.font)
794         frameSuivi.grid(column=1, row=0, sticky="ns", padx=5)
795
796     def __actualisationCanvas() -> None:
797         """Affiche l'histogramme des vente d'un utilisateur dans un canvas."""
798         donnees = Stats().recuperationDonneesCSV(utilisateur['id'])
799         if len(donnees) <= 0:
800             histogramme.create_text(10, 10, anchor='w', text="Aucun résultat récemment
enregistré")
801         else:
802             # Les dates dans le fichier CSV ne sont pas dans l'ordre
803             # On retire l'ID et le pseudo du dictionnaire
804             donnees.pop("id")
805             donnees.pop("pseudo")
806             ecart = 10
807             couleurs = [
808                 "CadetBlue3",
809                 "HotPink2",
810                 "IndianRed1",
811                 "MediumPurple2",
812                 "burlywood2",
813                 "brown3",
814                 "chocolate1",
815                 "goldenrod2"
816             ]
817             maxVente = 0 # par défaut la meilleur vente est de 0
818             for prix in donnees.values():
819                 prix = float(prix)
820                 if prix > maxVente: # si on trouve une valeur plus grande
821                     maxVente = prix
822
823             for date in sorted(donnees.keys()): # on regarde les dates dans l'ordre
824                 # Affichage de la date
825                 histogramme.create_text(ecart + 10, 60, anchor='w', text=date, font=("Arial",
8), angle=90)
826                 # Affichage de la barre
827                 hauteur = 190 - (float(donnees[date]) * 100) / maxVente # calcul de la
hauteur en fonction de la plus grosse vente
828                 # On fait '- 20' au résultat pour allonger la barre, aussi on met une barre
de '2' pixel quand valeur petite
829                 histogramme.create_rectangle(ecart, 180, ecart + 15, hauteur - 20 if hauteur
< 180 else 178, fill=couleurs.pop())
830                 # Affichage du montant

```

```

831         histogramme.create_text(ecart, 190, anchor='w', text=donnees[date], font=("
Arial", 8))
832         ecart += 33
833
834         histogramme = Canvas(frameSuivi, width=270, height=200)
835         histogramme.grid()
836
837         __actualisationCanvas()
838
839         Button(enfant, text="Quitter", command=enfant.destroy).grid(column=0, row=1, columnspan
=2)
840
841         Button(self.f, text="Ajouter un caissier", font=self.font, command=lambda:
__ajouterUtilisateur(1)).grid(column=0, row=2)
842         Button(self.f, text="Retirer un caissier", font=self.font, command=lambda:
__retirerUtilisateur(1)).grid(column=1, row=2)
843
844         Label(self.f).grid(row = 3, pady=10) # séparateur
845
846         # Liste des utilisateurs
847         managerVerif = IntVar(self.f) # filtre pour afficher ou non les managers dans la liste
848         caissierVerif = IntVar(self.f) # filtre pour afficher ou non les caissiers ou non dans la
liste
849
850         caissierVerif.set(1) # par défaut on affiche que les caissiers
851
852         def __ajouterUtilisateursListe(liste: Listbox, force: int = None) -> None:
853             """
854             Ajoute des utilisateurs à la liste du Manager.
855             -> metier = 0 : manager uniquement
856             -> metier = 1 : caissier uniquement
857             -> metier = 2 : manager et caissier
858             """
859             liste.delete(0, "end") # vidé la liste des utilisateurs
860             if force: # si 'force' n'est pas 'None', alors on force l'utilisation d'un métier
861                 metier = force
862             else: # sinon on fait une vérification normale en fonction des filtres de l'interface
manager
863                 if managerVerif.get() == 1:
864                     if caissierVerif.get() == 1:
865                         metier = None # on affiche les 2
866                     else:
867                         metier = 0 # on affiche seulement les managers
868                 else:
869                     metier = 1 # on affiche les caissiers
870                 if caissierVerif.get() == 0: # rien est coché, on revient à la configuration par
défaut (caissiers uniquement)
871                     metier = 1
872                     caissierVerif.set(1)
873
874                 if metier == None: # on ajoute tous les utilisateurs
875                     for idx, utilisateur in enumerate(Utilisateurs().listUtilisateurs()):
876                         liste.insert(idx, f"{utilisateur[0]} ({'manager' if utilisateur[1] == 0 else '
caissier'})")
877                 elif metier == 0: # on ajoute que les managers
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                 elif metier == 1: # on ajoute que les caissiers
882                     for idx, utilisateur in enumerate(Utilisateurs().listUtilisateurs()):
883                         if utilisateur[1] == metier:
884                             liste.insert(idx, f"{utilisateur[0]} ({'manager' if utilisateur[1] == 0 else
'caissier'})")
885                 else: # ce cas est là au cas où mais n'est pas sensé être appelé
886                     raise NameError("Métier inconnu.")
887
888             # Label d'information
889             Label(self.f, text="""
890                 Double-cliquez sur un
891                 utilisateur de la liste
892                 pour obtenir des informations
893                 supplémentaire à son sujet.
894                 """, justify="right").grid(column=1, row=4, rowspan=2, sticky="e")
895
896             Label(self.f, text="Liste des utilisateurs", font=self.font).grid(column=0, row=4) # titre
897             # On définit une barre pour pouvoir scroller dans la liste
898             scroll = Scrollbar(self.f)
899             scroll.grid(column=0, row=5, sticky="nse")
900             # On définit notre liste et on la lie à notre 'Scrollbar'
901             listeUtilisateurs = Listbox(self.f, width=25, height=4, yscrollcommand=scroll.set)
902             scroll.config(command=listeUtilisateurs.yview) # scroll à la verticale dans notre liste
903             # On ajoute nos utilisateurs à notre liste
904             __ajouterUtilisateursListe(listeUtilisateurs)
905             listeUtilisateurs.grid(column=0, row=5)

```

```

906     listeUtilisateurs.bind('<Double-Button>', __afficherInformationsUtilisateur) # on affiche l'
    utilisateur quand on double-clique dessus
907
908     # Filtre pour la liste
909     Label(self.f, text="Filtre", font=self.font).grid(column=1, row=4, sticky='w', padx=10) #
    titre
910     filtres = Frame(self.f) # Morceau qui va contenir nos checkbutton
911     filtres.grid(column=1, row=4, rowspan=2, sticky='w')
912     Checkbutton(filtres, text="Manager", variable=managerVerif, command=lambda:
    __ajouterUtilisateursListe(listeUtilisateurs)).grid(sticky='w')
913     Checkbutton(filtres, text="Caissier", variable=caissierVerif, command=lambda:
    __ajouterUtilisateursListe(listeUtilisateurs)).grid(sticky='w')
914
915     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)
916
917 if __name__ == "__main__":
918     """Application "GesMag" pour le module de Programmation d'interfaces (2021-2022)"""
919     """
920     Si presentation = True alors une base de donnée par défaut sera généré.
921     Si presentation = False ou n'est même pas mentionné, alors aucune base de donnée par défaut ne
    sera généré.
922     """
923     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             connexion = sqlite3.connect(path)
14         except sqlite3.Error as e:
15             print(e) # on affiche l'erreur
16             connexion = None # et renvoie None
17         return connexion
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) -> tuple:
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
    caractère (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.
57     """
58     valeurs = self.affichageResultat(curseur)
59     if len(valeurs) == 0:
60         valeurs = []
61     else:
62         valeurs = valeurs[0]
63     if type(cles) not in [list, tuple]:
64         cles = [cles]
65     if len(cles) != len(valeurs):
66         raise IndexError # il y a pas autant de clés que de valeurs
67     return dict(zip(cles, valeurs))

```

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

```

1  from db import BaseDeDonnees
2
3  class Utilisateurs(BaseDeDonnees):
4      """Gère une table "utilisateurs" pour une base de donnée donné."""
5      def __init__(self):
6          super().__init__(r"db.sqlite3") # connexion à la base de donnée
7
8      def creationTable(self, presentation: bool = False) -> None:
9          """Créer la table qui stocker les utilisateurs."""
10         requete = """
11             CREATE TABLE IF NOT EXISTS utilisateurs (
12                 id INTEGER PRIMARY KEY,
13                 pseudo TEXT,
14                 passe TEXT,
15                 metier INTEGER,
16                 nom TEXT,
17                 prenom TEXT,
18                 naissance TEXT,
19                 adresse TEXT,
20                 postal INTEGER
21             );
22         """
23         self.requete(requete)
24         # Ajout d'un utilisateur par défaut si aucun utilisateur n'existe dans la base de donnée
25         if len(self.listUtilisateurs()) == 0 and presentation:
26             self.ajoutUtilisateur(
27                 pseudo="admin",
28                 passe="P@ssword",
29                 metier=0,
30                 nom="Admin",
31                 prenom="Système",
32                 naissance="2000/10/09",
33                 adresse="12 Rue de Montmartre",
34                 postal=46800
35             )
36             print("-- Compte par défaut --\nNom d'utilisateur: admin\nMot de passe: P@ssword")
37
38         def ajoutUtilisateur(self, pseudo: str, passe: str, metier: int, nom: str, prenom: str, naissance
39         : str, adresse: str, postal: int) -> list:
40             """Ajoute un utilisateur et retourne l'ID de ce dernier."""
41             requete = """
42                 INSERT INTO utilisateurs (
43                     pseudo, passe, metier, nom, prenom, naissance, adresse, postal
44                 ) VALUES (
45                     ?, ?, ?, ?, ?, ?, ?, ?
46                 );
47             """
48             self.requete(requete, [pseudo.lower(), passe, metier, nom.upper(), prenom.capitalize(),
49             naissance, adresse, postal])
50             return self.affichageResultat(self.requete("SELECT last_insert_rowid();"))
51
52         def suppressionUtilisateurs(self, pseudo: str) -> None:
53             """Supprime un utilisateur."""
54             requete = """
55                 DELETE FROM utilisateurs
56                 WHERE pseudo = ?
57             """
58             self.requete(requete, pseudo)
59
60         def verificationIdentifiants(self, pseudo: str, motDePasse: str) -> tuple:
61             """
62             Retourne l'ID de l'utilisateur si trouvé dans la base de donnée ainsi
63             que son métier ('tuple'), sinon renvoie '(0,)'
64             """
65             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         """
82         recuperation = [
83             "id",
84             "pseudo",
85             "passe",
86             "metier",
87             "nom",
88             "prenom",
89             "naissance",
90             "adresse",
91             "postal",
92         ]
93         if not id: # si la variable 'id' n'est pas définie
94             if not pseudo: # si seul la variable 'pseudo' n'est pas définie
95                 raise ValueError # Aucun utilisateur renseigné
96             else: # si un pseudo est renseigné, c'est ce qu'on va utilisé
97                 requete = f"""
98                     SELECT {", ".join(recuperation)} FROM utilisateurs
99                     WHERE pseudo = ?
100                 """
101             utilisateur = pseudo
102         else: # si un id est renseigné, c'est ce qu'on va utilisé
103             requete = f"""
104                 SELECT {", ".join(recuperation)} FROM utilisateurs
105                 WHERE id = ?
106                 """
107             utilisateur = id
108         return self.affichageResultatDictionnaire(recuperation, self.requete(requete, utilisateur))
109
110     def utilisateurExistant(self, utilisateur: str) -> bool:
111         """Vérifie si l'utilisateur donnée existe déjà dans la base de donnée."""
112         requete = """
113             SELECT EXISTS (
114                 SELECT 1 FROM utilisateurs
115                 WHERE pseudo = ?
116             )
117             """
118         return True if self.affichageResultat(self.requete(requete, utilisateur.lower()))[0][0] == 1
119         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     default = {
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
78     # quantité aléatoire
79     for type in default:
80         for element in default[type]:
81             self.ajoutStock(type, element[0], randint(0, 10), round(uniform(2., 30.), 2),
82                             element[1])
83
84 def ajoutStock(self, typeElement: str, nom: str, quantite: int, prix: float, imageURL: str) ->
85 list:
86     """Ajoute un élément dans le stock et retourne l'ID de ce dernier."""
87     requete = """
88     INSERT INTO stocks (
89         type, nom, quantite, prix, image_url
90     ) VALUES (
91         ?, ?, ?, ?, ?
92     );
93     """
94     self.requete(requete, [typeElement.lower(), nom.lower(), quantite, prix, imageURL])
95     return self.affichageResultat(self.requete("SELECT last_insert_rowid();"))
96
97 def reduitQuantiteStock(self, id: int, quantiteARetirer: int) -> None:
98     """Retire une quantité d'un élément du stock et met-à-jour la base de donnée."""
99     requeteA = """
100     SELECT quantite FROM stocks
101     WHERE id = ?
102     """
103     quantiteActuelle: int = self.affichageResultat(self.requete(requeteA, id))[0][0]
104     if quantiteActuelle <= quantiteARetirer: # il ne reste plus rien
105         quantiteFinale = 0
106     else: # il reste quelque chose
107         quantiteFinale = quantiteActuelle - quantiteARetirer

```

```

105 # On met à jour la quantité de l'élément dans la base de donnée
106 requeteB = """
107     UPDATE stocks
108     SET quantite = ?
109     WHERE id = ?
110     """
111 self.requete(requeteB, [quantiteFinale, id])
112
113 def listeStocks(self) -> list:
114     """Retourne la liste des éléments en stock sous forme de dictionnaire."""
115     recuperation = [
116         "id",
117         "type",
118         "nom",
119         "quantite",
120         "prix",
121         "image_url"
122     ]
123     requete = f"""
124         SELECT {", ".join(recuperation)} FROM stocks
125         """
126     return [dict(zip(recuperation, element)) for element in self.affichageResultat(self.requete(
127         requete))]
128
129 def stockExistant(self, stock: str) -> bool:
130     """Vérifie si le stock donnée existe déjà dans la base de donnée."""
131     requete = """
132         SELECT EXISTS (
133             SELECT 1 FROM stocks
134             WHERE nom = ?
135         )
136         """
137     return True if self.affichageResultat(self.requete(requete, stock.lower()))[0][0] == 1 else
138     False
139
140 def listeTypes(self) -> list:
141     """Renvoie la liste des types disponibles dans la base de donnée."""
142     requete = """
143         SELECT type FROM stocks
144         """
145     res = []
146     for i in self.affichageResultat(self.requete(requete)):
147         if i[0] not in res:
148             res.append(i[0])
149     return res

```

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

```

1 import csv
2
3 from datetime import date, timedelta
4
5 from users import Utilisateurs
6
7 class Stats():
8     """Gère les statistiques et son export en format CSV."""
9     def __init__(self):
10         self.formatDate = "%Y/%m/%d"
11
12     def datesDisponibles(self) -> list:
13         """Renvoie les dates disponibles pour l'entête du fichier 'CSV'."""
14         datesPossibles = []
15         dateAujourdHui = date.today() - timedelta(days=7)
16         for _ in range(0, 8):
17             datesPossibles.append(dateAujourdHui.strftime(self.formatDate))
18             dateAujourdHui = dateAujourdHui + timedelta(days=1)
19         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 remplit 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'entête
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 remplit 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     besoinDeModification = 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 recrée 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                     besoinDeModification = True
134             else:
135                 datesPresentes.append(dateFichier) # on ajoute la date à la liste
136                 index += 1
137
138             if not besoinDeModification: # 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 besoinDeModification: # 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)

```