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`

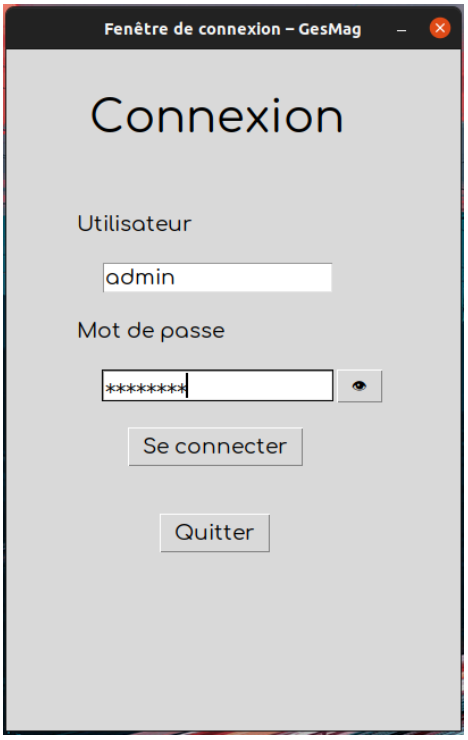


Figure 1: Page de login

☑ 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`

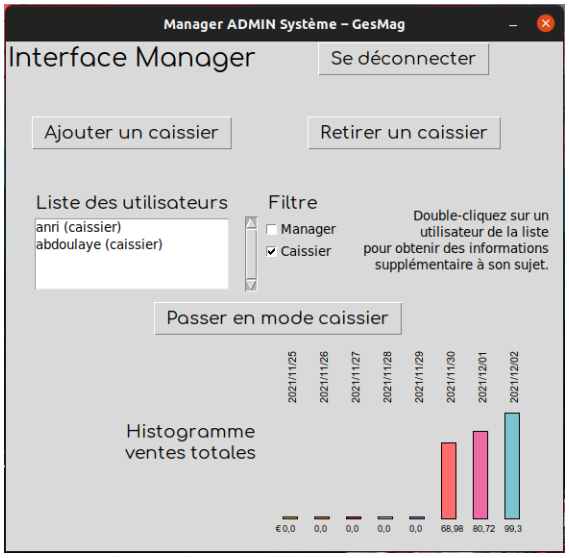


Figure 2: Page du Manager

- ☒ 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**

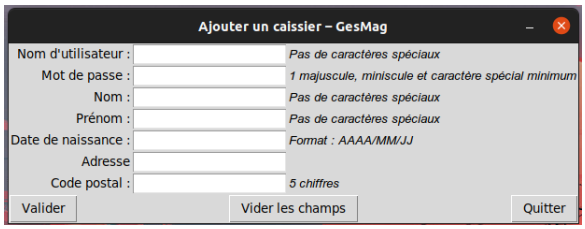


Figure 3: Ajout d'un caissier

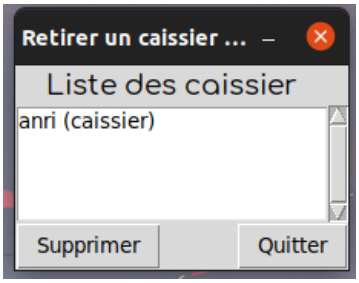


Figure 4: Suppression d'un caissier

- ☒ 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**

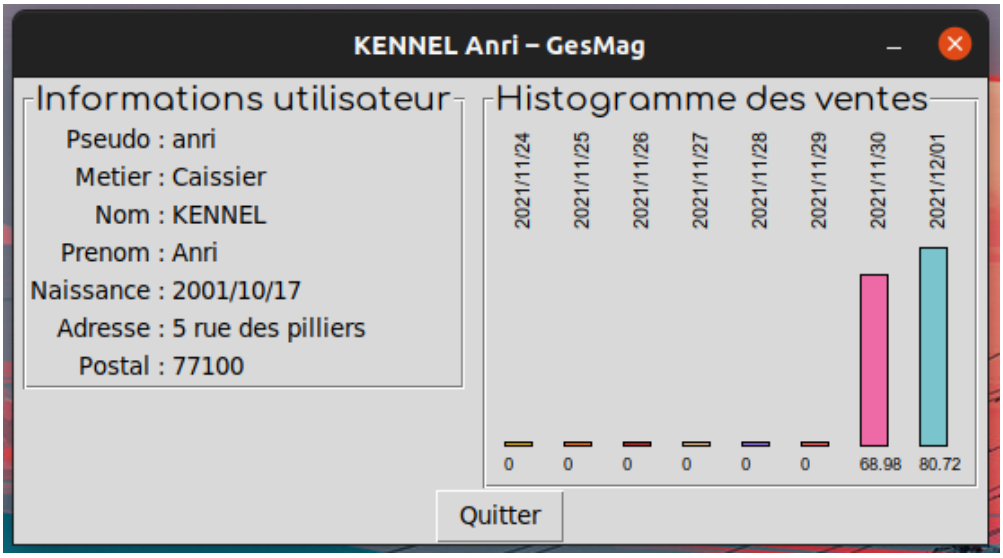


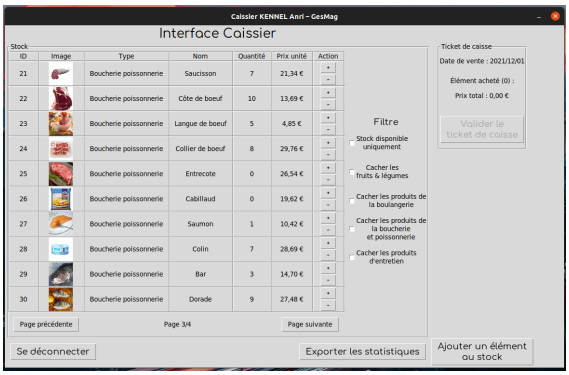
Figure 5: Informations étendues sur un utilisateur

☑ Un bouton pour vider tous les champs de saisie

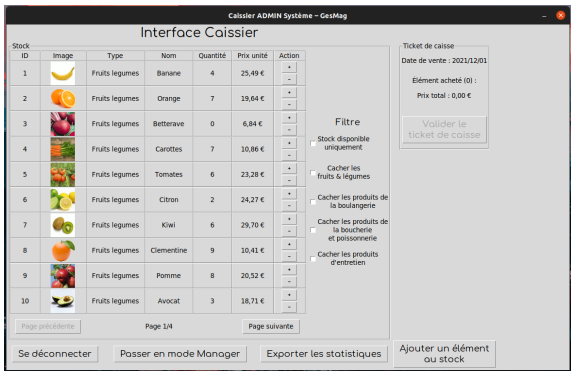
☒ Un bouton pour quitter l'application ⇒ **j'ai préféré mettre un bouton pour se déconnecter**

☑ 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



(a) En tant que Caissier



(b) En tant que Manager

Figure 6: Page du Caissier

☑ 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

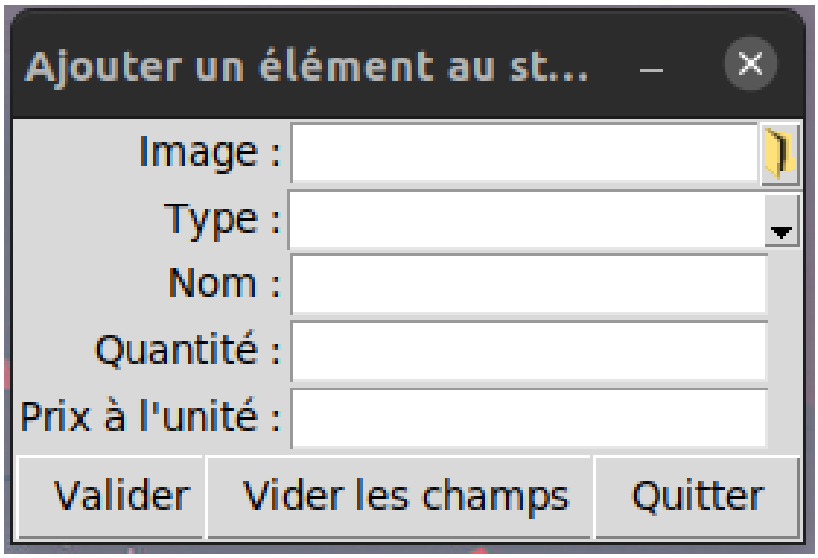


Figure 7: Ajout d'éléments au stock

☑ Affichage d'un ticket de caisse

Figure 8: Ticket de caisse avec 2 éléments au panier

- ☑ 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="Connexion", font=(self.font[0], 30)).grid(column=1, row=0, columnspan=2)#
     titre
170         Label(self.f, text="Utilisateur", font=self.font).grid(column=0, row=1, columnspan=2, padx=
    ecart - 20, pady=20, sticky='w')
171         utilisateur = Entry(self.f, font=self.font, width=18)
172         utilisateur.grid(column=1, row=2, columnspan=2, padx=ecart)
173 
174         Label(self.f, text="Mot de passe", font=self.font).grid(column=0, row=3, columnspan=2, padx=
    ecart - 20, pady=20, sticky='w')
175         motDpasse = Entry(self.f, font=self.font, show='', width=18)
176         motDpasse.grid(column=1, row=4, columnspan=2, padx=ecart)
177         motDpasse.bind("<Return>", tentativeDeConnexion)
178 
179         def __afficherMDP(self) -> None:
180             """Permet de gérer l'affichage du mot de passe dans le champs sur la page de connexion.
    """
181             if self.mdpVisible == False: # si mot de passe caché, alors on l'affiche
182                 self.mdpVisible = True
183                 motDpasse.config(show='')
184                 bouttonAffichageMDP.config(font=("Arial", 10, "overstrike"))
185 
186             else: # inversement
187                 self.mdpVisible = False
188                 motDpasse.config(show='')
189                 bouttonAffichageMDP.config(font=("Arial", 10))
190 
191         bouttonAffichageMDP = Button(self.f, text='', command=lambda: __afficherMDP(self))
192         bouttonAffichageMDP.grid(column=2, row=4, columnspan=2)
193         self.mdpVisible = False
194 
195         bouton = Button(self.f, text="Se connecter", font=self.font, command=tentativeDeConnexion)
196         bouton.grid(column=0, row=5, columnspan=3, padx=ecart, pady=20)
197         bouton.bind("<Return>", tentativeDeConnexion)
198 
199         Button(self.f, text="Quitter", font=self.font, command=quit).grid(column=0, row=6, columnspan
    =4, pady=20)
200 
201     def _interfaceCaissier(self, id: int) -> None:
202         """Affiche l'interface du caissier."""
203         caissier = Utilisateurs().recuperationUtilisateur(id=id)
204         self.parent.title(f"Caissier {caissier['nom']} {caissier['prenom']}  {self.nomApp}")
205         self.dimensionsFenetre(self.parent, 1160, 710)
206 
207         self.panier = [] # remet le panier à 0
208 
209         # Suppresssion de la dernière Frame
210         self.f.destroy()
211         # Instanciation d'une nouvelle Frame, on va donc ajouter tout nos widgets à cet Frame
212         self.f = Frame(self.parent)
213         self.f.grid()
214 
215         Label(self.f, text="Interface Caissier", font=(self.font[0], 20)).grid(column=0, row=0) #
    titre de l'interface
216 
217         def __formatPrix(prix: str) -> str:
218             """
219             Renvoie un string pour un meilleur affichage du prix :
220                 - ',' au lieu de '.'
221                 - Symbole ''
222                 - 2 chiffres après la virgule
223             """
224             return f"{float(prix):.2f} ".replace('.', ',')
225 
226         # -> Partie affichage du Stock
227         stock = LabelFrame(self.f, text="Stock")
228         stock.grid(column=0, row=1, sticky='n', padx=5)
229 
230         # Variables pour les filtres du tableau
231         stockDisponibleVerif = IntVar(stock) # controle si on affiche que les éléments en stocks ou
    non
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:
239             """Fonction qui va actualiser le tableau avec une page donnée (par défaut affiche la
    première page)."""
240             # On supprime et refais la frame qui va stocker notre tableau
241             self.tableau.destroy()
242             self.tableau = Frame(stock)
243             self.tableau.grid(column=0, row=1, columnspan=7)
244 
245             # Filtre pour le tableau
246             filtres = Frame(stock) # Morceau qui va contenir nos checkbutton
247             ecartFiltre = 10 # écart entre les champs des filtres
248             Label(filtres, text="Filtre", font=self.font).grid(column=0, row=0) # titre
249             Checkbutton(filtres, text="Stock disponible\nuniquement", variable=stockDisponibleVerif,
    command=__affichageTableau).grid(sticky='w', pady=ecartFiltre)
250             Checkbutton(filtres, text="Cacher les\nfruits & légumes", variable=fruitsLegumesVerif,
    command=__affichageTableau).grid(sticky='w', pady=ecartFiltre)
251             Checkbutton(filtres, text="Cacher les produits de\nla boulangerie", variable=
    boulangerieVerif, command=__affichageTableau).grid(sticky='w', pady=ecartFiltre)
252             Checkbutton(filtres, text="Cacher les produits de\nla boucherie\net poissonnerie",
    variable=boucheriePoissonnerieVerif, command=__affichageTableau).grid(sticky='w')
253             Checkbutton(filtres, text="Cacher les produits\nd'entretien", variable=entretienVerif,
    command=__affichageTableau).grid(sticky='w', pady=ecartFiltre)
254             filtres.grid(column=7, row=1, sticky='w')
255 
256             stockListe = Stock().listeStocks() # stock récupéré de la base de données
257 
258             def ___miseAJourPanier(element: dict, action: bool) -> None:
259                 """
260                 Permet d'ajouter ou de retirer des éléments au panier
261                 -> Action
262                     -> Vrai : Ajout
263                     -> Faux : Retire
264                 """
265                 # On compte combien de fois l'élément est présent dans le panier
266                 nombreDeFoisPresentDansLePanier = 0
267                 index = None
268                 for idx, elementDansLePanier in enumerate(self.panier):
269                     if elementDansLePanier[0] == element:
270                         index = idx # On met à jour l'index
271                         nombreDeFoisPresentDansLePanier = elementDansLePanier[1]
272                         break # on peut quitter la boucle car on a trouvé notre élément
273 
274                 # On vérifie que on peut encore l'ajouter/retirer
275                 if nombreDeFoisPresentDansLePanier == 0 and not action: # pop-up seulement si on veut
     retirer un élément pas présent
276                     showerror(f"Erreur  {self.nomApp}", "Impossible de retirer cet élément au panier
    .\nNon présent dans le panier.")
277                     return
278                 if nombreDeFoisPresentDansLePanier >= element["quantite"] and action: # pop-up
    seulement si on veut en rajouter
279                     showerror(f"Erreur  {self.nomApp}", "Impossible de rajouter cet élément au panier
    .\nLimite excédée.")
280                     return
281 
282                 if index != None: # on retire l'ancienne valeur du panier si déjà présente dans le
    panier
283                     self.panier.pop(index)
284                 else: # sinon on définie un index pour pouvoir ajouté la nouvelle valeur à la fin de
    la liste
285                     index = len(self.panier)
286 
287                 # On change la valeur dans le panier
288                 if action: # si on ajoute
289                     nombreDeFoisPresentDansLePanier += 1
290                 else: # si on retire
291                     nombreDeFoisPresentDansLePanier -= 1
292 
293                 # On rajoute l'élément avec sa nouvelle quantité seulement s'il y en a
294                 if nombreDeFoisPresentDansLePanier > 0:
295                     self.panier.insert(index, (element, nombreDeFoisPresentDansLePanier))
296 
297                 __affichagePanier() # met-à-jour le panier
298 
299             for i in range(0, len(stockListe)): # on retire les éléments plus présent dans la liste
300                 if stockDisponibleVerif.get() == 1 and stockListe[i]["quantite"] < 1:
301                     stockListe[i] = None
302                 elif fruitsLegumesVerif.get() == 1 and stockListe[i]["type"] == "fruits legumes":
303                     stockListe[i] = None
304                 elif boulangerieVerif.get() == 1 and stockListe[i]["type"] == "boulangerie":
305                     stockListe[i] = None
306                 elif boucheriePoissonnerieVerif.get() == 1 and stockListe[i]["type"] == "boucherie
    poissonnerie":
307                     stockListe[i] = None
308                 elif entretienVerif.get() == 1 and stockListe[i]["type"] == "entretien":
309                     stockListe[i] = None
```

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

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

```
536             image.grid(column=1, row=0, sticky='w')
537             def ___selectionImage() -> None:
538                 """Fonction qui permet de choisir une image dans l'arborescence de fichiers de l'
    utilisateur."""
539                 try:
540                     chemin = askopenfile(title=f"Choisir une image  {self.nomApp}", filetypes=[("
    Image GIF", ".gif")])
541                     image.delete(0, "end")
542                     image.insert(0, chemin.name)
543                 except AttributeError: # si l'utilisateur n'a pas choisit d'image
544                     pass
545 
546             Button(enfant, image=self.dossierImage, command=___selectionImage).grid(column=1, row=0,
    sticky='e')
547             # Type (ComboBox)
548             Label(enfant, text="Type :").grid(column=0, row=1, sticky='e')
549             typeElement = Combobox(enfant, values=Stock().listeTypes())
550             # typeElement.current(0) # valeur 0 par défaut
551             typeElement.grid(column=1, row=1, sticky='w')
552             # Nom
553             Label(enfant, text="Nom :").grid(column=0, row=2, sticky='e')
554             nom = Entry(enfant)
555             nom.grid(column=1, row=2, sticky='w')
556             # Quantité
557             Label(enfant, text="Quantité :").grid(column=0, row=3, sticky='e')
558             quantite = Entry(enfant)
559             quantite.grid(column=1, row=3, sticky='w')
560             # Prix à l'unité
561             Label(enfant, text="Prix à l'unité :").grid(column=0, row=4, sticky='e')
562             prix = Entry(enfant)
563             prix.grid(column=1, row=4, sticky='w')
564 
565             def ___viderChamps() -> None:
566                 """Vide tout les champs de leur contenu"""
567                 # On récupère toutes les 'Entry' de la fenêtre et on change leur contenu
568                 for champ in [widget for typeElement, widget in enfant.children.items() if "entry" in
     typeElement]:
569                     champ.delete(0, "end")
570                     champ.update()
571 
572             # Boutons
573             Button(enfant, text="Valider", command=___verification).grid(column=0, row=8, columnspan
    =3, sticky='w')
574             Button(enfant, text="Vider les champs", command=___viderChamps).grid(column=0, row=8,
    columnspan=3)
575             Button(enfant, text="Quitter", command=enfant.destroy).grid(column=0, row=8, columnspan
    =3, sticky='e')
576 
577         Button(self.f, text="Ajouter un élément\nau stock", font=self.font, command=
    __ajouterElementStock).grid(column=1, row=2)
578 
579         # -> Partie export des statistiques
580         def __exportation() -> None:
581             """Exporte dans un fichier choisie par l'utilisateur ses statistiques de la journée."""
582             chemin = asksaveasfile(title=f"Exportation des statistiques de {caissier['nom']} {
    caissier['prenom']}  {self.nomApp}", filetypes=[("Fichier CSV", ".csv")])
583             if chemin == None: # si rien n'a été spécifie on arrête l'exportation
584                 return
585             Stats().exporteCSV(chemin.name, id)
586         Button(self.f, text="Exporter les statistiques", font=self.font, command=__exportation).grid(
    column=0, row=2, sticky='e', padx=ecart)
587 
588         Button(self.f, text="Se déconnecter", font=self.font, command=self._interfaceConnexion).grid(
    column=0, row=2, sticky='w', padx=ecart)
589 
590         # -> Boutton pour passer en mode manager si la personne est un manager
591         if caissier["metier"] == 0:
592             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)
593 
594     def _interfaceManager(self, id: int) -> None:
595         """Affiche l'interface du manager."""
596         manager = Utilisateurs().recuperationUtilisateur(id=id)
597         # Dans le cas où un utilisateur réussi à trouvé cette interface alors qu'il n'a pas le droit,
     il sera bloqué
598         if manager["metier"] != 0:
599             showerror(f"Erreur  {self.nomApp}", "Vous ne pouvez pas accéder à cette interface.")
600             return
601         self.parent.title(f"Manager {manager['nom']} {manager['prenom']}  {self.nomApp}")
602         self.dimensionsFenetre(self.parent, 580, 530)
603 
604         # Suppresssion de la dernière Frame
605         self.f.destroy()
606         # Instanciation d'une nouvelle Frame, on va donc ajouter tout nos widgets à cet Frame
607         self.f = Frame(self.parent)
```

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

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

```
758                     utilisateur = listeUtilisateurs_retirer.get(listeUtilisateurs_retirer.
    curselection()[0]).split('(')[0][:-1]
759                     reponse = askyesno(f"Confirmation  {self.nomApp}", f"Voulez vous supprimer {
    utilisateur} ?")
760                     if reponse == True:
761                         Utilisateurs().suppressionUtilisateurs(utilisateur)
762                         __ajouterUtilisateursListe(listeUtilisateurs_retirer) # met à jour la liste
    dans la fenêtre de suppression
763                         __ajouterUtilisateursListe(listeUtilisateurs) # met à jour la liste dans l'
    interface principale
764                         showinfo(f"Information  {self.nomApp}", f"Utilisateur {utilisateur} supprimé.
    ")
765 
766             # Boutons
767             Button(enfant, text="Supprimer", command=___suppressionUtilisateur).grid(column=0, row=8,
     columnspan=3, sticky='w')
768             Button(enfant, text="Quitter", command=enfant.destroy).grid(column=0, row=8, columnspan
    =3, sticky='e')
769 
770         def __actualisationCanvas(canvas: Canvas, donnees: dict) -> None:
771             """Affiche l'histogramme des vente d'un utilisateur dans un canvas."""
772             if len(donnees) <= 0:
773                 canvas.create_text(10, 10, anchor='w', text="Aucun résultat récemment enregistré")
774             else:
775                 # Les dates dans le fichier CSV ne sont pas dans l'ordre
776                 # On retire l'ID et le pseudo du dictionnaire si présent dedans
777                 donnees.pop("id", None)
778                 donnees.pop("pseudo", None)
779                 ecart = 10
780                 couleurs = [
781                     "CadetBlue3",
782                     "HotPink2",
783                     "IndianRed1",
784                     "MediumPurple2",
785                     "burlywood2",
786                     "brown3",
787                     "chocolate1",
788                     "goldenrod2"
789                 ]
790                 # On récupère la plus grosse vente
791                 maxVente = 0.1 # par défaut la meilleur vente est de 0.1 (pas 0 car on ne peut pas
    divisier par 0, cf. calculs en dessous)
792                 for prix in donnees.values():
793                     prix = float(prix)
794                     if prix > maxVente: # si on trouve une valeure plus grande
795                         maxVente = prix
796                 canvas.create_text(3, 190, anchor='w', text="", font=("Arial", 7)) # affichage de la
    devise
797                 for date in sorted(donnees.keys()): # on regarde les dates dans l'ordre
798                     # Affichage de la date
799                     canvas.create_text(ecart + 10, 60, anchor='w', text=date, font=("Arial", 8),
    angle=90)
800                     # Affichage de la barre
801                     hauteur = 190 - (float(donnees[date]) * 100) / maxVente # calcul de la hauteur en
     fonction de la plus grosse vente
802                     # On fait `- 20` au résultat pour allonger la barre, aussi on met une barre de
    `2` pixel quand valeur petite
803                     canvas.create_rectangle(ecart, 180, ecart + 15, hauteur - 20 if hauteur < 180
    else 178, fill=couleurs.pop())
804                     # Affichage du montant
805                     canvas.create_text(ecart, 190, anchor='w', text=str(donnees[date]).replace('.', '
    ,'), font=("Arial", 7))
806                     ecart += 33
807 
808         def __afficherInformationsUtilisateur(_) -> None:
809             """Permet d'afficher les informations d'un utilisateur"""
810             element = listeUtilisateurs.curselection()
811             if len(element) == 0: # si aucun élément n'est selectionné
812                 return
813             """
814             On split le champs car dans la liste on affiche le métier entre
815             parenthèses et on doit donner que le nom d'utilisateur à
816             la fonction `recuperationUtilisateur`, aussi on retire le dernièr
817             charactère avec [:-1] car c'est un espace.
818             """
819             utilisateur = Utilisateurs().recuperationUtilisateur(pseudo=listeUtilisateurs.get(element
    [0]).split('(')[0][:-1])
820             enfant = Toplevel(self.f) # cf. l'explication dans `__ajouterUtilisateur`
821             enfant.resizable(False, False)
822             enfant.title(f"{utilisateur['nom']} {utilisateur['prenom']}  {self.nomApp}")
823 
824             # Informations sur l'utilisateur
825             frameInfos = LabelFrame(enfant, text="Informations utilisateur", font=self.font)
826             frameInfos.grid(column=0, row=0, sticky='n', padx=5)
827             utilisateur["metier"] = "Manager" if utilisateur["metier"] == 0 else "Caissier"
```

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

```
903             # On définit une barre pour pouvoir scroller dans la liste
904             scroll = Scrollbar(self.f)
905             scroll.grid(column=0, row=5, sticky="nse")
906             # On définit notre liste et on la lie à notre ‘Scrollbar‘
907             listeUtilisateurs = Listbox(self.f, width=25, height=4, yscrollcommand=scroll.set)
908             scroll.config(command=listeUtilisateurs.yview) # scroll à la verticale dans notre liste
909             # On ajoute nos utilisateurs à notre liste
910             __ajouterUtilisateursListe(listeUtilisateurs)
911             listeUtilisateurs.grid(column=0, row=5)
912             listeUtilisateurs.bind('<Double-Button>', __afficherInformationsUtilisateur) # on affiche l'
        utilisateur quand on double-clique dessus
913 
914             # Filtre pour la liste
915             Label(self.f, text="Filtre", font=self.font).grid(column=1, row=4, sticky='w', padx=10) #
        titre
916             filtres = Frame(self.f) # Morceau qui va contenir nos checkbutton
917             filtres.grid(column=1, row=4, rowspan=2, sticky='w')
918             Checkbutton(filtres, text="Manager", variable=managerVerif, command=lambda:
        __ajouterUtilisateursListe(listeUtilisateurs)).grid(sticky='w')
919             Checkbutton(filtres, text="Caissier", variable=caissierVerif, command=lambda:
        __ajouterUtilisateursListe(listeUtilisateurs)).grid(sticky='w')
920 
921             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)
922 
923             # Histogramme global
924             Label(self.f, text="Histogramme\nventes totales", font=self.font).grid(column=0, row=7,
        sticky='e')
925             histogrammeGlobale = Canvas(self.f, width=270, height=200)
926             histogrammeGlobale.grid(column=1, row=7, columnspan=2)
927 
928             donneesGlobales = Stats().recuperationDonneesCSV()
929             __actualisationCanvas(histogrammeGlobale, donneesGlobales)
930 
931 if __name__ == "__main__":
932     """Application "GesMag" pour le module de Programmation d'interfaces (2021-2022)"""
933     """
934     Si presentation = True alors une base de donnée par défaut sera généré.
935     Si presentation = False ou n'est même pas mentionné, alors aucune base de donnée par défaut ne
        sera généré.
936     """
937     GesMag(presentation = True).demarrer()
```

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

```
1  import sqlite3
2  
3  class BaseDeDonnees:
4      """Gère la base de donnée."""
5      def __init__(self, urlBaseDeDonnee: str):
6          self.connexion = self.creerConnexion(urlBaseDeDonnee)
7  
8      def creerConnexion(self, path: str):
9          """Connexion à une base de donnée SQLite."""
10         if not self.fichierExiste(path): # si l base de donnée n'existe pas
11             open(path, 'x') # on la créer
12         try:
13             connnexion = sqlite3.connect(path)
14         except sqlite3.Error as e:
15             print(e) # on affiche l'erreur
16             connnexion = None # et renvoie None
17         return connnexion
18 
19     def fichierExiste(self, path: str) -> bool:
20         """Vérifie qu'un fichier existe."""
21         try: # on essaie d'ouvrir le fichier
22             open(path, 'r')
23         except FileNotFoundError: # si le fichier n'existe pas
24             return False
25         else: # si le fichier existe
26             return True
27 
28     def requete(self, requete: str, valeurs = None) -> 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
        charactère (par exemple), alors la converti en liste
34                     valeurs = [valeurs]
35                 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();"))

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

    def verificationIdentifiants(self, pseudo: str, motDePasse: str) -> tuple:
        """
        Retourne l'ID de l'utilisateur si trouvé dans la base de donnée ainsi
        que son métier (`tuple`), sinon renvoie `(0,)`.
        """
        requete = """
                  SELECT id, metier FROM utilisateurs
                  WHERE pseudo = ? AND passe = ?
                  """
        reponseBaseDeDonnee = self.affichageResultat(self.requete(requete, [pseudo.lower(), motDePasse]))
        if len(reponseBaseDeDonnee) == 0: # si les identifiants renseignés sont mauvais
            return (0,)
        return reponseBaseDeDonnee[0]

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

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

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

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

```
from random import randint, uniform

from db import BaseDeDonnees

```

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

    def creationTable(self, presentation: bool = False) -> None:
        """Créer la table qui stocker les stocks."""
        requete = """
                  CREATE TABLE IF NOT EXISTS stocks (
                      id INTEGER PRIMARY KEY,
                      type TEXT,
                      nom TEXT,
                      quantite INTEGER,
                      prix REAL,
                      image_url TEXT
                  );
                  """
        self.requete(requete)
        # Ajout d'un stock par défaut si aucun stock n'existe dans la base de donnée
        if len(self.listeStocks()) == 0 and presentation:
            # Créer un dictionnaire d'éléments pour mieux voir ce que l'on ajoute à la base de donnée
            defaut = {
                "fruits legumes": [
                    ("banane", "img/banane.gif"),
                    ("orange", "img/orange.gif"),
                    ("betterave", "img/betterave.gif"),
                    ("carottes", "img/carottes.gif"),
                    ("tomates", "img/tomates.gif"),
                    ("citron", "img/citron.gif"),
                    ("kiwi", "img/kiwi.gif"),
                    ("clementine", "img/clementine.gif"),
                    ("pomme", "img/pomme.gif"),
                    ("avocat", "img/avocat.gif")
                ],
                "boulangerie": [
                    ("brownie", "img/brownie.gif"),
                    ("baguette", "img/baguette.gif"),
                    ("pain au chocolat", "img/pain_au_chocolat.gif"),
                    ("croissant", "img/croissant.gif"),
                    ("macaron", "img/macaron.gif"),
                    ("millefeuille", "img/millefeuille.gif"),
                    ("paris-brest", "img/paris-brest.gif"),
                    ("opera", "img/opera.gif"),
                    ("fraisier", "img/fraisier.gif"),
                    ("eclair", "img/eclair.gif")
                ],
                "boucherie poissonnerie": [
                    ("saucisson", "img/saucisson.gif"),
                    ("côte de boeuf", "img/cote_de_boeuf.gif"),
                    ("langue de boeuf", "img/langue_de_boeuf.gif"),
                    ("collier de boeuf", "img/collier_de_boeuf.gif"),
                    ("entrecote", "img/entrecote.gif"),
                    ("cabillaud", "img/cabillaud.gif"),
                    ("saumon", "img/saumon.gif"),
                    ("colin", "img/colin.gif"),
                    ("bar", "img/bar.gif"),
                    ("dorade", "img/dorade.gif")
                ],
                "entretien": [
                    ("nettoyant air comprimé", "img/nettoyant_air_comprime.gif"),
                    ("nettoyage anti-bactérien", "img/nettoyage_anti-bacterien.gif"),
                    ("nettoyant pour écran", "img/nettoyant_pour_ecran.gif"),
                    ("nettoyant pour lunettes", "img/nettoyant_pour_lunettes.gif"),
                    ("pioche", "img/pioche.gif"),
                    ("pelle", "img/pelle.gif"),
                    ("lampe torche", "img/lampe_torche.gif"),
                    ("gants", "img/gants.gif"),
                    ("éponge", "img/eponge.gif"),
                    ("essuie-tout", "img/essuie-tout.gif")
                ]
            }

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

    def ajoutStock(self, typeElement: str, nom: str, quantite: int, prix: float, imageURL: str) -> list:
        """Ajoute un élément dans le stock et retourne l'ID de ce dernier."""
        requete = """
                  INSERT INTO stocks (
                      type, nom, quantite, prix, image_url
```

```
                    ) VALUES (
                        ?, ?, ?, ?, ?
                    );
                    """
        self.requete(requete, [typeElement.lower(), nom.lower(), quantite, prix, imageURL])
        return self.affichageResultat(self.requete("SELECT last_insert_rowid();"))

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

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

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

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

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

```
import csv

from datetime import date, timedelta

from users import Utilisateurs

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

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

```
            datesPossibles.append(dateAujourdHui.strftime(self.formatDate))
            dateAujourdHui = dateAujourdHui + timedelta(days=1)
        return datesPossibles

    def creationCSV(self, force: bool = False) -> None:
        """
        Créer le fichier ‘CSV‘ qui stockera les statistiques pour tous les utilisateurs.

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

    def miseAJourStatsUtilisateur(self, utilisateurID: int, prix: float) -> None:
        """
        Récupère le prix d'une transaction et l'ajoute au total d'un utilisateur.

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

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

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

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

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


    def exporteCSV(self, chemin: str, utilisateurID: int) -> None:
        """
        Exporte les statistiques d'un utilisateur dans un fichier ‘CSV‘.
        - N'exporte que les statistiques du jour.
        """
        donnees = self.recuperationDonneesCSV(utilisateurID)
        aujourdHui = date.today().strftime(self.formatDate)
        with open(chemin, 'w') as f:
            fichier = csv.writer(f)
            fichier.writerow(["ID Utilisateur", f"Totales des ventes du jour ({aujourdHui})"])
```

```
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 = None) -> dict:
103         """
104         Renvoie les informations contenu dans le fichier ‘CSV‘ globale.
105
106         Si un argument est renseignée pour ‘utilisateurID‘ alors seul l’utilisateur
107         renseigné sera renvoyé.
108         """
109         self.miseAJourDatesCSV() # met à jour les dates du fichier ‘CSV‘
110
111         with open("stats.csv", ’r’) as f:
112             fichier = list(csv.DictReader(f)) # lecture du fichier sous forme d’une liste de
    dictionnaire
113             if utilisateurID == None: # si on renvoie les stats globales
114                 dates = dict.fromkeys(self.datesDisponibles(), 0.)
115                 for date in dates:
116                     for utilisateur in fichier:
117                         if utilisateur[date] == None: # on remplace les date sans prix par ‘0.‘
118                             utilisateur[date] = 0.
119                         dates[date] += float(utilisateur[date])
120                 return dates
121             else: # si on renvoie les stats spécifique à un utilisateur
122                 for utilisateur in fichier: # on regarde tous les utilisateurs stockés dans le
    fichier
123                     if utilisateur["id"] == str(utilisateurID): # si utilisateur trouvé
124                         for date, prix in utilisateur.items(): # on remplace les date sans prix par
    ‘0.‘
125                             if prix == None:
126                                 utilisateur[date] = 0.
127                         return utilisateur # renvoie des infos de l’utilisateur
128         return {} # ne retourne rien si l’utilisateur n’était pas présent dans le fichier
129
130     def miseAJourDatesCSV(self) -> None:
131         """
132         Mets-à-jour les dates trop anciennes du fichier globales ‘CSV‘.
133
134         On remplie les espaces vides par la valeur ‘0‘ pour les jours qui remplacent les dates
135         trop vieilles (âgées de plus d’une semaine) car soit aucun chiffre n’a été fait ce jour là,
136         car aucune information n’était renseignée.
137         """
138         besoinDeMofication = False
139         with open("stats.csv", ’r’) as f:
140             fichier = list(csv.reader(f))
141             if len(fichier) == 1: # si fichier ne contient que l’entête
142                 self.creationCSV(True) # on recréer le fichier dans le doute (avec les bonnes dates)
143             else:
144                 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])
145                 mauvaisIndex = [] # liste qui va stocker les index trop vieux (> 1 semaine)
146                 datesPresentes = [] # liste qui stock les dates valides présentes dans le fichier (<
    1 semaine)
147                 datesDisponibles = self.datesDisponibles() # liste des bonnes dates
148                 for dateFichier in fichier[0][index:]: # on regarde toutes les dates du fichier
149                     if dateFichier not in datesDisponibles: # si trop vieux
150                         mauvaisIndex.append(index) # on ajoute l’index à la liste
151                         besoinDeMofication = True
152                     else:
153                         datesPresentes.append(dateFichier) # on ajoute la date à la liste
154                     index += 1
155
156                 for i in range(1, len(fichier)): # on regarde tous les éléments présent dans notre
    fichier (sauf l’entête)
157                     while len(fichier[0]) < len(fichier[i]): # s’il y a plus d’éléments renseignés
    que d’élément dans l’entête
158                         besoinDeMofication = True # on note qu’on a besoin de modifier le fichier
159                         del fichier[i][-1] # on retire le dernier élément "en trop"
160
161                 if not besoinDeMofication: # vérification si on a besoin de rien faire
162                     return # on quitte la fonction
163
164                 datesARajouter = [date for date in datesDisponibles if date not in datesPresentes] #
    liste des dates à rajouter
165                 if len(datesARajouter) != len(mauvaisIndex): # ne devrais pas arrivé mais on sait
    jamais
166                     raise IndexError("Problème, pas autant de dates à rajouter que de de dates
    périmés dans le fichier.")
167                 for idx in mauvaisIndex: # pour tous les mauvais index
168                     for numLigne, ligne in enumerate(fichier): # on regarde toutes les lignes du
    fichier
169                         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
```

```
170                             if numLigne == 0: # si c'est la ligne d'entête
171                                 ligne[idx] = datesARajouter[0] # on change la ligne avec la nouvelle
        date
172                                 datesARajouter.pop(0)
173                             else: # si c'est une ligne de donnée
174                                 ligne[idx] = '0' # on change la ligne avec une valeur vide
175                             fichier[numLigne] = ligne # on applique les changements
176 
177         if besoinDeMofication: # vérification si on a besoin de faire des changements
178             with open("stats.csv", 'w') as f: # on applique les changements
179                 ecriture = csv.writer(f)
180                 ecriture.writerows(fichier)
```