ENSAM
ÉCOLE NATIONALE SUPÉRIEURE D'ARTS ET MÉTIERS
UNIVERSITÉ HASSAN II DE CASABLANCA


# RAPPORT
# GESTION D'HOTEL

Réalisé par :

TIOULI SALMA
M'BARKI MAHMOUD

Encadré par :

Pr. M.Hain

Année scolaire
2025-2026

# Sommaire



Tiouli Salma
M'barki Mahmoud

# I. Introduction

Dans un contexte où la digitalisation occupe une place essentielle dans la gestion des services, les hôtels cherchent de plus en plus à automatiser leurs processus administratifs afin de gagner en efficacité et en fiabilité. La gestion manuelle des réservations, des chambres et des clients peut rapidement devenir complexe et source d'erreurs, surtout lorsque le volume d'informations augmente.

C'est dans cette optique que s'inscrit ce projet, dont l'objectif principal est de concevoir une **application de gestion d'hôtel** simple, intuitive et fonctionnelle. Cette application permet de gérer les **clients**, les **chambres** et les **réservations** à partir d'une interface unique, conviviale et responsive, réalisée à l'aide du module **Tkinter** de Python.

Le projet repose sur une **base de données SQLite**, assurant le stockage local des informations, tout en garantissant leur persistance entre les différentes sessions d'utilisation. L'interface graphique, divisée en plusieurs onglets, offre une navigation fluide et permet d'ajouter, de supprimer et de consulter les données en temps réel.

Ainsi, cette application constitue une solution pratique pour tout établissement hôtelier souhaitant moderniser sa gestion, en combinant **simplicité d'utilisation**, **automatisation des tâches** et **centralisation des informations** dans un seul outil.

# II. Description générale du projet

L'application développée dans le cadre de ce projet a pour objectif de centraliser et de simplifier la gestion quotidienne d'un hôtel. Elle est faites pour être utilisée par le personnel administratif afin de gérer efficacement les **clients**, les **chambres** et les **réservations**.

L'interface est organisée en **trois onglets principaux** correspondant aux fonctionnalités clés du système :

1. **Clients** : cet onglet permet ***d'ajouter*** de nouveaux clients, de ***visualiser*** la liste complète des clients existants et de ***supprimer*** des clients si nécessaire. Chaque client est identifié de manière unique par son **CIN**, et les informations de base comme le **nom** et le **numéro de téléphone** sont stockées.

2. **Chambres** : dans cet onglet, l'utilisateur peut ***enregistrer*** de nouvelles chambres en précisant le **type** (single, double ou suite), le **prix** et la **disponibilité** (1 chambre disponible, 0 chambre réservée ou sous travaux). La liste des chambres existantes peut être consultée et actualisée à tout moment.

3. **Réservations** : cet onglet permet ***d'effectuer*** des réservations pour les **clients** en choisissant la **chambre** et **les dates de séjour**. Le ***système vérifie automatiquement la disponibilité*** des chambres pour éviter les conflits de réservation. La liste des réservations est également consultable, et l'utilisateur peut ***supprimer*** une réservation existante si nécessaire.

L'application repose sur une **base de données SQLite**, qui stocke toutes les informations de manière structurée et persistante. Chaque table correspond à un élément clé du système : clients, chambres et réservations. Les relations entre les tables sont gérées via des clés primaires et étrangères, garantissant l'intégrité des données.

Grâce à une interface graphique intuitive réalisée avec **Tkinter**, l'application permet une interaction facile et rapide avec la base de données. Les actions comme l'ajout, la suppression ou la consultation des informations sont effectuées en quelques clics, offrant ainsi une expérience utilisateur fluide et efficace.

En résumé, ce projet constitue une solution complète et simple à mettre en œuvre pour la gestion d'un hôtel, combinant **centralisation des données**, **facilité d'utilisation** et **fiabilité**.

# III. Présentation de l'application et des interfaces

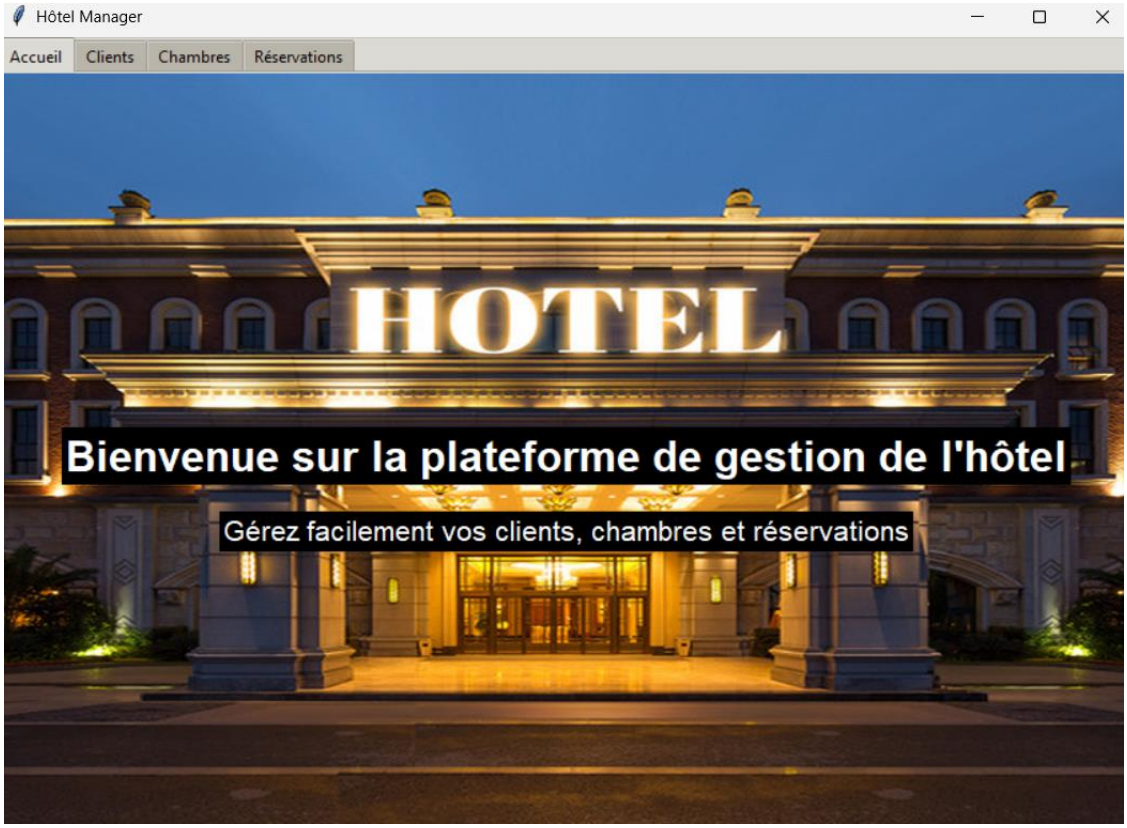
L'application "Hôtel Manager" est un logiciel de gestion simplifiée d'un hôtel. Elle permet de gérer trois éléments principaux : les *clients*, les *chambres* et les *réservations*. L'interface est conçue avec **Tkinter**, et se veut intuitive et visuelle grâce à l'utilisation de **Treeviews** et d'onglets.

## 1. Onglets et navigation

L'application est organisée en plusieurs onglets :

- **Accueil** : présente un écran de bienvenue avec un message d'introduction et une image de fond représentant l'hôtel.
- **Clients** : permet l'ajout, la consultation et la suppression des clients.
- **Chambres** : permet l'ajout, la consultation et la gestion de la disponibilité des chambres.
- **Réservations** : permet l'ajout, la consultation et la suppression des réservations, tout en vérifiant la disponibilité des chambres.

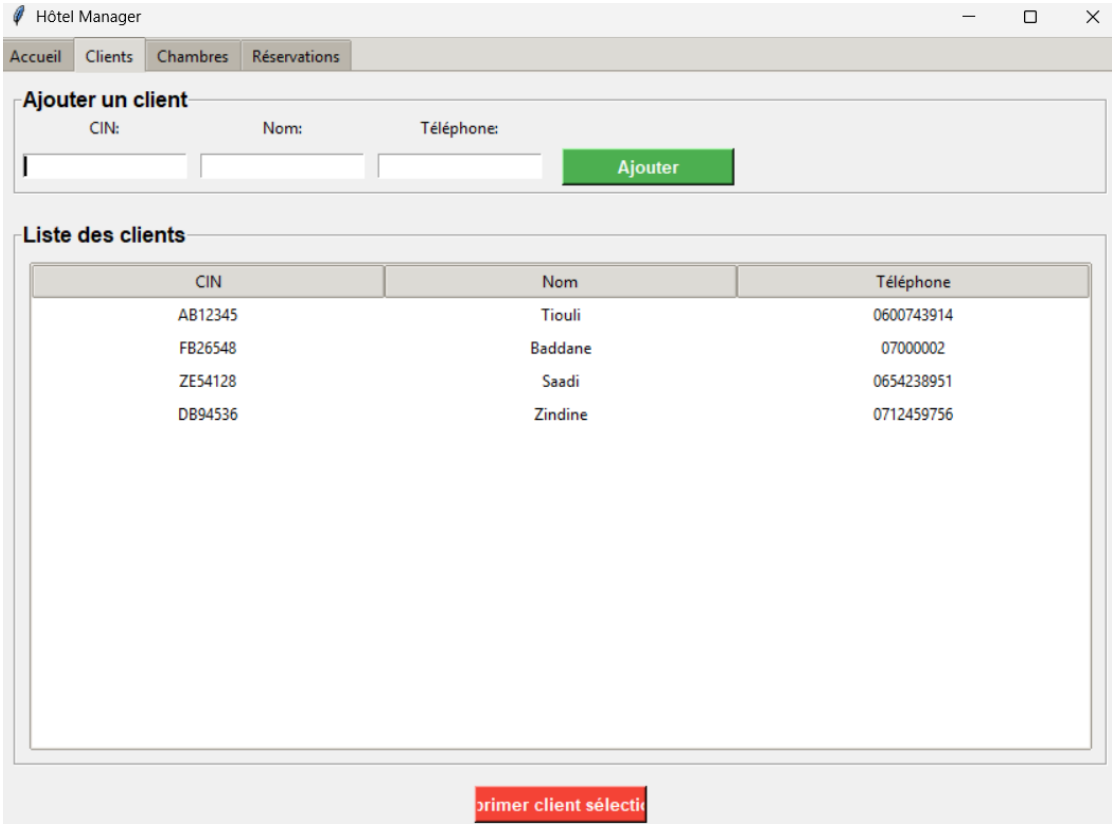
La navigation entre les onglets se fait facilement via un **menu de type Notebook**, permettant d'accéder rapidement à chaque fonctionnalité.




## 2. Gestion des clients

Dans l'onglet Clients :

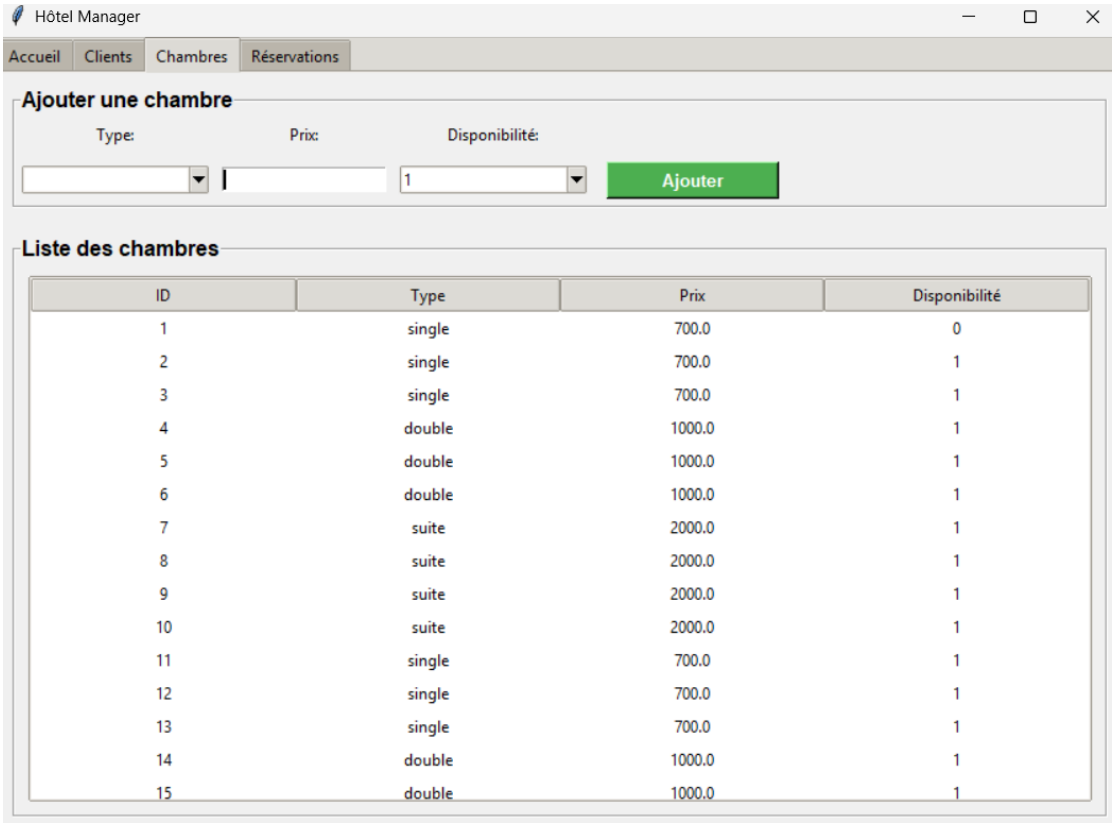
- Un formulaire permet d'ajouter un client en saisissant son CIN, son nom et son téléphone.
- Une **liste (Treeview)** affiche tous les clients présents dans la base de données.
- Un bouton permet de supprimer le client sélectionné.



## 3. Gestion des chambres

Dans l'onglet Chambres :

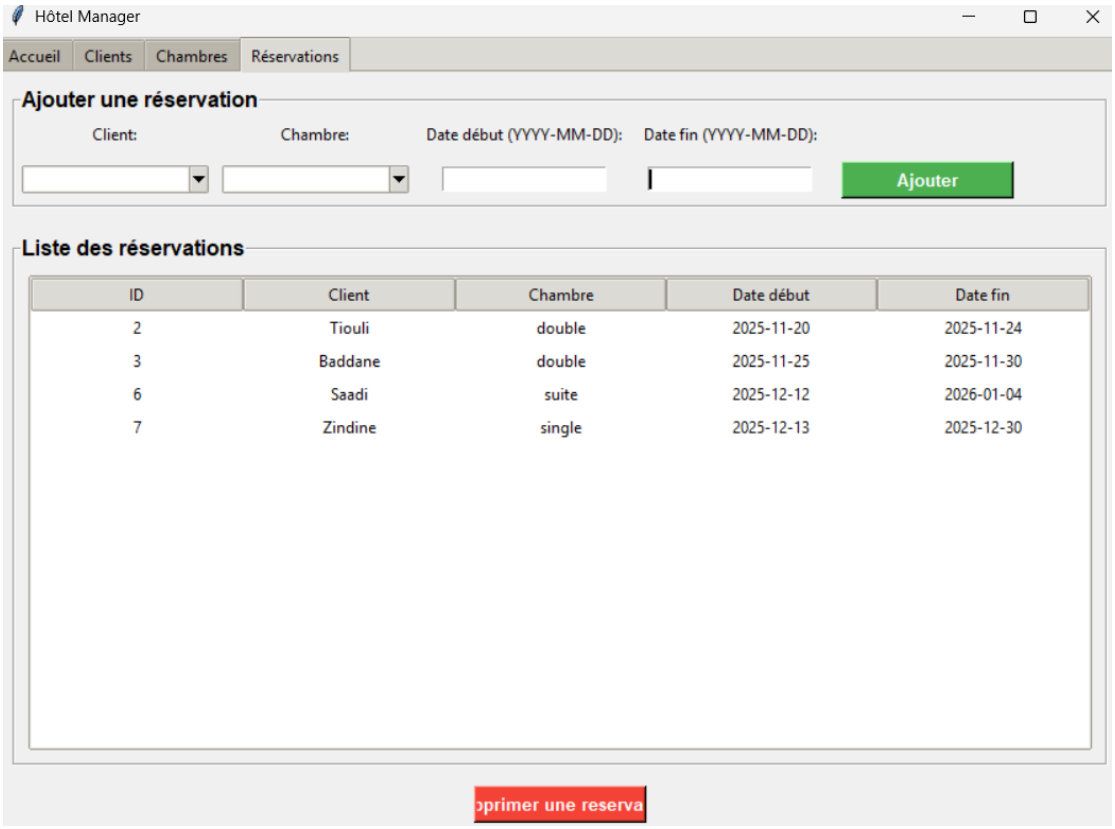
- Les chambres peuvent être ajoutées avec leur type, leur prix et leur disponibilité.
- Une liste affiche toutes les chambres avec leurs informations.
- Les chambres disponibles sont automatiquement prises en compte lors de la création d'une réservation.




## 4. Gestion des réservations

Dans l'onglet Réservations :

- Il est possible d'ajouter une réservation en sélectionnant un client, une chambre disponible, et en définissant les dates de début et de fin.
- Une vérification automatique empêche de réserver une chambre déjà occupée sur la période sélectionnée.
- La liste des réservations permet de consulter et de supprimer les réservations existantes




# IV. Fonctionnalités principales et fonctionnement du code

L'application "Hôtel Manager" est organisée autour de trois fonctionnalités principales : **gestion des clients**, **gestion des chambres** et **gestion des réservations**. Chaque fonctionnalité est liée à un onglet dans l'interface et interagit directement avec la base de données SQLite.

## 1. Gestion des clients

- **Ajouter un client** :

L'utilisateur saisit le CIN, le nom et le téléphone dans le formulaire de l'onglet Clients.
Le bouton "Ajouter" déclenche la fonction `action_ajouter_client()`, qui récupère les informations avec `entry_cin.get()`, `entry_nom.get()` et `entry_tel.get()`, puis appelle la fonction `ajouter_client(cin, nom, tel)` du module de base de données pour insérer le client dans la table `clients`.
Après ajout, les champs sont vidés avec `entry.delete()` et la liste des clients est actualisée avec `charger_clients()`.

```python
def action_ajouter_client():  1 usage
    cin = entry_cin.get()
    nom = entry_nom.get()
    tel = entry_tel.get()
    if not cin or not nom:
        messagebox.showwarning( title: "Attention",  message: "CIN et Nom sont obligatoires !")
        return
    try:
        ajouter_client(cin, nom, tel)
        messagebox.showinfo( title: "Succès",  message: "Client ajouté avec succès ")
        entry_cin.delete( first: 0, tk.END)
        entry_nom.delete( first: 0, tk.END)
        entry_tel.delete( first: 0, tk.END)
        charger_clients()
        entry_client['values'] = [c[0] + " - " + c[1] for c in afficher_clients()]

    except Exception as e:
        messagebox.showerror( title: "Erreur",  message: f"Impossible d'ajouter le client : {e}")
```

*Fig 1 : Fonction « action_ajouter_client() » du fichier main.py*

```python
btn_add_client = tk.Button(frame_add_client, text="Ajouter", command=action_ajouter_client, bg="#4CAF50", **btn_style)
btn_add_client.grid(row=1, column=3, padx=10, pady=5)
```

*Fig 2 : Bouton « Ajouter »*

```python
def ajouter_client(CIN, nom, telephone):  1 usage
    db = connexion()
    cur = db.cursor()
    cur.execute( sql: "INSERT INTO clients(CIN, nom, telephone) VALUES (?, ?, ?)", parameters: (CIN, nom, telephone))
    db.commit()
    db.close()
```

*Fig 3 : Fonction « Ajouter client » du fichier database.py*

- **Afficher les clients** :

La liste des clients est affichée dans un `Treeview`. La fonction `charger_clients()` commence par supprimer les anciennes lignes puis insère les clients récupérés depuis la base de données :

```python
frame_table_client = tk.LabelFrame(tab_clients, text="Liste des clients", font=("Arial", 12, "bold"))
frame_table_client.pack(padx=10, pady=10, fill="both", expand=True)

cols_client = ("CIN", "Nom", "Téléphone")
tree_client = ttk.Treeview(frame_table_client, columns=cols_client, show="headings")
for col in cols_client:
    tree_client.heading(col, text=col)
    tree_client.column(col, width=200, anchor="center")
tree_client.pack(fill="both", expand=True, padx=10, pady=10)

def charger_clients():  3 usages
    for i in tree_client.get_children():  #sert a ne pas repeter toutes les lignes une autre fois lors
        tree_client.delete(i)
    for c in afficher_clients():
        tree_client.insert( parent: "",  index: "end", values=c)
```

*Fig 4 : extrait du code main.py permettant d'afficher la liste des clients dans un treeview*

- **Supprimer un client** :

L'utilisateur sélectionne un client dans la liste et clique sur le bouton "Supprimer". La fonction `action_supprimer_clients()` récupère l'identifiant du client sélectionné et appelle `supprimer_clients(id_cl)`. La liste est ensuite actualisée pour refléter la suppression.

```
def action_supprimer_clients():  1 usage
    selected = tree_client.selection()
    if not selected:
        messagebox.showwarning( title: "Attention",  message: "Sélectionnez un client à supprimer !")
        return
    id_cl = tree_client.item(selected[0])["values"][0]
    confirmer = messagebox.askyesno( title: "Confirmer",  message: f"Supprimer le client {id_cl} ?")
    if confirmer:
        supprimer_clients(id_cl)
        charger_clients()

btn_supprimer_clients = tk.Button(tab_clients, text="Supprimer client sélectionné", command=action_supprimer_clients, bg="#f44336", **btn_style)
btn_supprimer_clients.pack(pady=5)
```

*Fig 5 : extrait du main.py pour supprimer un client*

```
def supprimer_clients(id_client):  1 usage
    db = connexion()
    cur = db.cursor()
    cur.execute( sql: "DELETE FROM clients WHERE CIN=?",  parameters: (id_client,))
    db.commit()
    db.close()
```

*Fig 6 : extrait du code database.py pour supprimer un client de la liste*

## 2. Gestion des chambres

- **Ajouter une chambre** :

Le formulaire comprend le type de chambre, le prix et la disponibilité. Le bouton "Ajouter" déclenche `action_ajouter_chambre()`, qui convertit les valeurs en types appropriés (`float` pour le prix, `int` pour la disponibilité) puis appelle `ajouter_chambre(type_ch, prix, dispo)` pour enregistrer la chambre dans la table `chambres`.

```
def ajouter_chambre(type_chambre, prix, dispo):  1 usage
    db = connexion()
    cur = db.cursor()
    cur.execute( sql: "INSERT INTO chambres(type, prix, disponibilite) VALUES (?, ?, ?)",  parameters: (type_chambre, prix, dispo))
    db.commit()
    db.close()
```

*Fig 7 : extrait de d'une fct du code database.py utiliser dans le main pour ajouter une chambre*

```python
def action_ajouter_chambre():  1 usage
    type_ch = entry_type.get()
    prix = entry_prix.get()
    dispo = entry_dispo.get()
    if not type_ch or not prix:
        messagebox.showwarning( title: "Attention",  message: "Type et prix obligatoires !")
        return
    try:
        prix = float(prix)
        dispo = int(dispo)
        ajouter_chambre(type_ch, prix, dispo)
        messagebox.showinfo( title: "Succès",  message: "Chambre ajoutée ")
        entry_type.set('')
        entry_prix.delete( first: 0, tk.END)
        entry_dispo.set(1)
        charger_chambres()
        entry_chambre['values'] = chambres_disponibles()
    except Exception as e:
        messagebox.showerror( title: "Erreur",  message: f"Impossible d'ajouter la chambre : {e}")

btn_add_chambre = tk.Button(frame_add_chambre, text="Ajouter", command=action_ajouter_chambre, bg="#4CAF50", **btn_style)
btn_add_chambre.grid(row=1, column=3, padx=10, pady=5)
```

*Fig 8 : extrait du main.py pour ajouter un chambre*

- **Afficher les chambres** :

La fonction `charger_chambres()` fonctionne comme pour les clients : elle supprime d'abord les anciennes lignes puis insère toutes les chambres existantes dans le Treeview.

```python
def charger_chambres():  2 usages
    for i in tree_chambre.get_children():
        tree_chambre.delete(i)
    for c in afficher_chambres():
        tree_chambre.insert( parent: "",  index: "end", values=c)
```

*Fig 9 : Fonction afficher les chambres du fichier main.py*

- **Disponibilité des chambres** :

La fonction `chambres_disponibles()` filtre les chambres dont la colonne `disponibilite` vaut 1, afin que seules les chambres libres soient affichées lors de la création d'une réservation.

```python
def chambre_disponible(id_chambre, date_debut, date_fin):  1 usage
    db = connexion()
    cur = db.cursor()
    query = """
        SELECT COUNT(*)
        FROM reservations
        WHERE id_chambre = ?
        AND (
            (date_debut <= ? AND date_fin >= ?) OR
            (date_debut <= ? AND date_fin >= ?) OR
            (? <= date_debut AND ? >= date_fin)
        )
    """
    cur.execute(query,  parameters: (id_chambre, date_debut, date_debut, date_fin, date_fin, date_debut, date_fin))
    count = cur.fetchone()[0]
    db.close()
    return count == 0  # True si aucune réservation en conflit
```

*Fig 10 : Fonction permettant de filtrer les chambres disponibles du fichier database.py*

## 3. Gestion des réservations

- **Ajouter une réservation** :

L'utilisateur sélectionne un client et une chambre disponibles, puis saisit les dates de début et de fin. Le bouton "Ajouter" déclenche `action_ajouter_reservation()`.
Le code vérifie d'abord que la chambre est libre avec la fonction `chambre_disponible(id_chambre, date_debut, date_fin)` qui compte les conflits dans la table `reservations` :

```
def action_ajouter_reservation():  1 usage
    client_sel = entry_client.get()
    chambre_sel = entry_chambre.get()
    date_debut = entry_date_debut.get()
    date_fin = entry_date_fin.get()

    if not client_sel or not chambre_sel or not date_debut or not date_fin:
        messagebox.showwarning( title: "Attention",  message: "Tous les champs sont obligatoires !")
        return

    id_client = client_sel.split(" - ")[0]
    id_chambre = int(chambre_sel.split(" - ")[0])
    if not chambre_disponible(id_chambre, date_debut, date_fin):
        messagebox.showwarning( title: "Attention",  message: "Cette chambre est déjà réservée pour cette période !")
        return

    try:
        ajouter_reservation(id_client, id_chambre, date_debut, date_fin)
        messagebox.showinfo( title: "Succès",  message: "Réservation ajoutée ")
        entry_client.set('')
        entry_chambre.set('')
        entry_date_debut.delete( first: 0, tk.END)
        entry_date_fin.delete( first: 0, tk.END)
        charger_reservations()
        # Actualiser la liste des chambres disponibles
        entry_chambre['values'] = chambres_disponibles()
    except Exception as e:
        messagebox.showerror( title: "Erreur",  message: f"Impossible d'ajouter la réservation : {e}")
```

*Fig 11 : Fonction d'ajout d'une réservation du fichier main.py*

Si la chambre est disponible, la réservation est ajoutée avec `ajouter_reservation(id_client, id_chambre, date_debut, date_fin)` et la liste des réservations est mise à jour avec `charger_reservations()`.

- **Afficher les réservations** :

La fonction `charger_reservations()` remplit le Treeview avec les réservations existantes, en joignant les tables `clients` et `chambres` pour afficher les noms et types au lieu des identifiants.

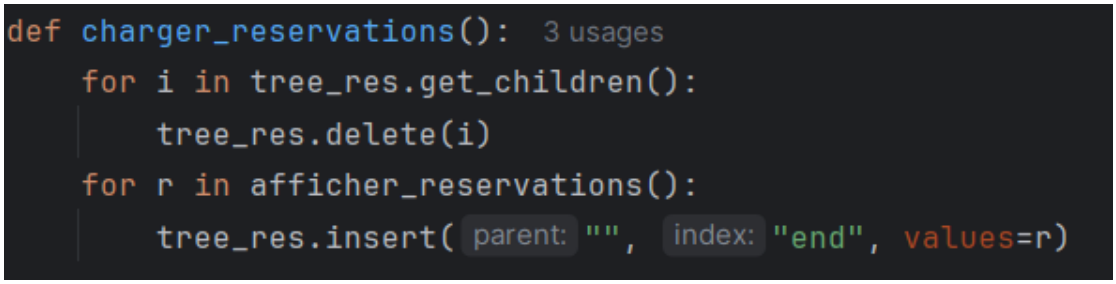



*Fig 12 : Fonction afficher les réservations du fichier main.py*

- **Supprimer une réservation** :

Similaire à la suppression de clients, l'utilisateur sélectionne une réservation et clique sur "Supprimer". La fonction `action_supprimer_reservation()` supprime la réservation dans la base et actualise la liste.


*Fig 13 : extrait du main.py pour supprimer une réservation*

# V. Conclusion

En conclusion, ce projet m'a permis de concevoir une application complète de **gestion d'hôtel** en Python, utilisant **Tkinter** pour l'interface graphique et **SQLite** pour la gestion de la base de données. L'application permet de **gérer efficacement les clients, les chambres et les réservations**, tout en assurant la cohérence des données et la disponibilité des chambres.

Ce travail m'a donné l'occasion de mettre en pratique mes compétences en **programmation, organisation des données et conception d'interface utilisateur**, tout en consolidant ma compréhension des bases de données relationnelles et des interactions entre l'interface graphique et la logique métier.

Je tiens à exprimer mes **remerciements à mon professeur, M.Hain** pour ses conseils précieux, son accompagnement et sa disponibilité, qui ont grandement contribué à la réussite de ce projet.