

Cahier des charges SAE 3.02

Dans le cadre de ce projet SAE 3.02, les participants SORAGNA Quentin et OCANA Kevin seront amenés à développer des outils logiciels permettant l'échange d'informations au sein d'un environnement technologique. Le but de celui-ci est la gestion d'un hôtel R&T, qui à travers un formulaire, enregistrera les données inscrites dans un serveur BD. Une fois les données récupérées, de nombreuses règles permettent de savoir si la réservation est acceptée et le client enregistré. Un accusé de réception sera ensuite envoyé comprenant le prix à payer pour la réservation demandée par l'utilisateur. Après l'accord du client sur le prix, il pourra s'authentifier s'il souhaite revoir sa réservation.

Parties du projet :

- 1) Interface graphique pour que les clients puissent choisir quel type de chambres ils veulent, pendant combien de temps...
- 2) Ecriture d'un script Python
- 3) Création d'une base de donnée
- 4) Utilisation de Buildozer

Partie 1 :

Nous allons utiliser un module de python que nous avons déjà utilisé qui est kv. Kv commence à être un module python qui nous paraît de plus en plus familier de part nos TP dessus. Cependant, certaines utilisations comme changer d'interface lorsqu'on appuie sur un bouton ou l'utilisation d'un menu déroulant nous est encore inconnu.

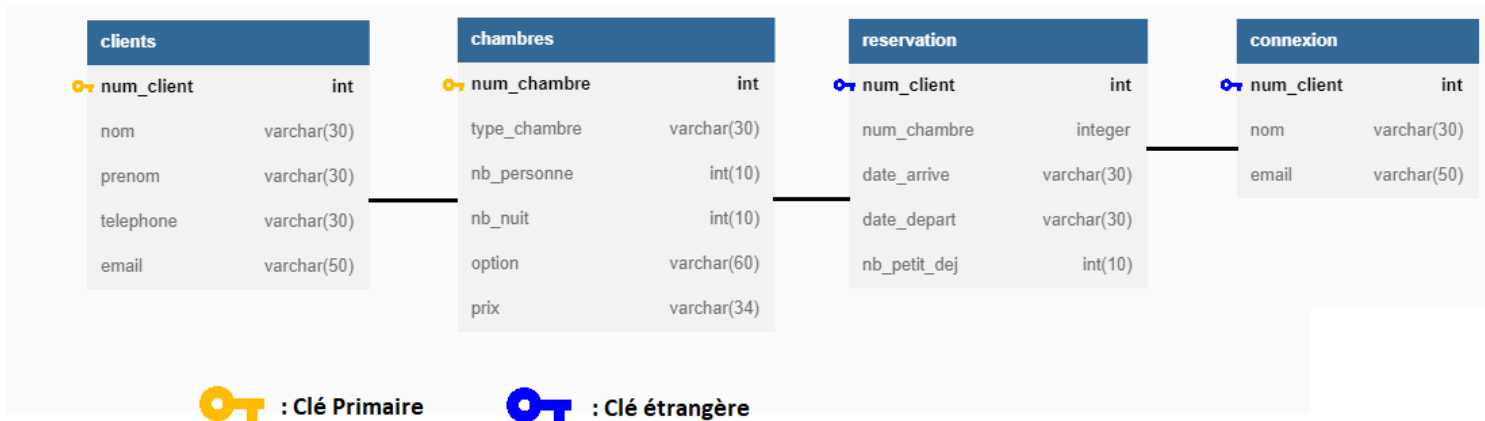
Partie 2 :

Nous allons utiliser l'interface précédemment créée et pouvoir mettre en place des conditions (pour que les dates d'arrivée et date de départ soient cohérentes et que le client soit alerté s'ils ont fait une erreur dans les informations qu'ils ont entrées).

Partie 3:

La base de donnée que nous allons créer sera divisée en 3 tables, une pour les informations de chaque chambre, une pour stocker les informations des clients et enfin une dernière pour stocker les différentes réservations. Pour cela, nous utiliserons mariadb puis un connecteur python pour pouvoir gérer la base de donnée avec python.

Le schéma de nos tables est le suivant :



Partie 4:

Enfin, nous utiliserons Buildozer afin d'adapter notre projet sur un smartphone et que le client puisse donc réserver une chambre sur son téléphone portable.

Utiliser la bibliothèque Buildozer pour adapter notre projet aux smartphones est une tâche qui peut nous prendre du temps car nous ne connaissons que de nom cette bibliothèque python. Cela va être à nous de nous documenter sur celle-ci pour l'utiliser au mieux.

Exigences:

- Utilisation du langage Kivy, Python, kv, de base de donnée
- Utilisation de Buildozer pour rendre notre projet fonctionnel sur téléphone.

Pour réaliser ce projet un ordinateur est nécessaire, les différentes langages de programmations précédemment énumérés aussi. Les langages sont gratuits à l'utilisation, que ce soit pour développer ce projet et ensuite l'adapter pour téléphone.

Résultat attendu : Le résultat attendu de ce projet sera une interface graphique optimale pour une réservation avec la mise en place de méthodes permettant le bon fonctionnement de la gestion de l'hôtel. Les informations rentrées dans l'interface devront être envoyées dans une base de données puis nous devons récupérer ses informations pour les afficher lors de l'identification de l'utilisateur.

Planning

| |
|--|
| Séance 1 : regroupement des idées + commencer interface Kivy (part1) |
| Séance 2 : regroupement des idées + commencer interface Kivy (part2) |
| Séance 3 : regroupement des idées + commencer interface Kivy (part3) |
| Séance 4 : création interface Kivy (part1) |
| Séance 5 : création interface Kivy (part2) |
| Séance 6 : création interface Kivy (part3) |
| Séance 7 : Création base de données / utilisateur / tables (part1) |
| Séance 8 : Création base de données / utilisateur / tables (part2) |
| Séance 9 : Création base de données / utilisateur / tables (part3) |
| Séance 10 : regroupement interface Kivy base de données (part1) |
| Séance 11 : regroupement interface Kivy base de données (part2) |
| Séance 12 : regroupement interface Kivy base de données (part3) |
| Séance 13 : rédaction compte rendu + vérification du programme |

Suivi de projet SAE 3.02

Pour commencer ce projet, nous nous sommes penchés sur la création de la class **Hotel** qui contient tous les attributs qui nous permettront de réaliser le projet.

```
#Création de la class Hotel

class Hotel:
    def __init__(self):
        self.__Nom = ""
        self.__Prenom = ""
        self.__AdresseMail = ""
        self.__ConfirmAdresse = ""
        self.__Tel = 0
        self.__Datearrive = ""
        self.__Datedepart = ""
        self.__Nbrnuit = 0
        self.__TypeChambre = ""
        self.__Nbrpersonnes = 0
        self.__Nbrpetitdej = 0
        self.__Confort = ""
```

Nous avons ensuite créé les **Accesseurs-Mutateurs** pour stocker les données de nos attributs (sur cette photo, quelques exemples ont été donnés) ainsi que les sécuriser.

```
#Accesseurs-Mutateurs
def get_Nom(self):
    return self.__Nom

def set_Nom(self, Nom):
    self.__Nom = Nom

def get_Prenom(self):
    return self.__Prenom

def set_Prenom(self, Prenom):
    self.__Prenom = Prenom

def get_AdresseMail(self):
    return self.__AdresseMail
```

Puis, nous avons élaboré cette méthode qui nous permettra d'entrer avec des `inputs` nos données pour tester notre programme.

```
#Méthode permettant de rentrer nos données

def get_data(self):

    self.set_Nom(input("Nom: "))
    self.set_Prenom(input("Prenom: "))
    self.set_AdresseMail(input("AdresseMail: "))
    self.set_ConfirmAdresse(input("ConfirmAdresse: "))
    self.set_Tel(input("Tel: "))
    self.set_Datearrive(input("Datearrive: "))
    self.set_Datedepart(input("Datedepart: "))
    self.set_TypeChambre(input("Typechambre: "))
    self.set_Nbrpersonnes(input("Nbrpersonnes: "))
    self.set_Nbrpetitdej(input("NbrDeDeuj: "))
    self.set_Confort(input("Confort: "))
```

Nous avons ensuite créer la méthode pour savoir si la date d'arrivée est bien ultérieure à la date de départ :

1ère partie :

```
#Méthode permettant de tester la validité des dates

def calcul_date(self):
    i=0
    while i == 0:
        arrive_decompo=self.get_Datearrive().split("/")
        depart_decompo=self.get_Datedepart().split("/")

        #si l'année est inférieure, on recommence

        if int(depart_decompo[2]) < int(arrive_decompo[2]):
            print("la date d'arrivée est inférieure à la date de retour, recommencez : ")
            self.set_Datearrive(input("Datearrive: "))
            self.set_Datedepart(input("Datedepart: "))

        #si l'année est égale ou supérieure, on teste les mois et jours

        elif int(depart_decompo[2]) >= int(arrive_decompo[2]):

            #si le mois est inférieure

            if int(depart_decompo[1]) < int(arrive_decompo[1]):
                print("la date d'arrivée est inférieure à la date de retour, recommencez : ")
                self.set_Datearrive(input("Datearrive: "))
                self.set_Datedepart(input("Datedepart: "))
```

2ème partie :

```
#si le mois est égal

elif int(depart_decompo[1]) == int(arrive_decompo[1]) and int(depart_decompo[2]) == int(arrive_decompo[2]):

    if int(depart_decompo[0]) == int(arrive_decompo[0]):
        print("même date, recommencez : ")
        self.set_Datearrive(input("Datearrive: "))
        self.set_Datedepart(input("Datedepart: "))

    elif int(depart_decompo[0]) < int(arrive_decompo[0]):
        print("la date d'arrivée est inférieure à la date de retour, recommencez :")
        self.set_Datearrive(input("Datearrive: "))
        self.set_Datedepart(input("Datedepart: "))

    else:
        i=1

else:
    i=1

else:
    i=1
```

Pour ce programme nous avons créer une boucle `while` `i=0` `while i -- 0:` qui nous permettra de relancer les conditions tant qu'elles ne sont pas validées.

Nous avons ensuite décomposé notre chaîne avec un `split()` pour analyser chaque argument (jour, mois et année) et les comparer entre eux (conditions).

```
arrive_decompo=self.get_Datearrive().split("/")
depart_decompo=self.get_Datedepart().split("/")
```

Les conditions sont émises avec un `if` et nous commençons par comparer les années entre la date d'arrivée et la date de départ.

```
#si l'année est inférieure, on recommence

if int(depart_decompo[2]) < int(arrive_decompo[2]):
    print("la date d'arrivée est inférieure à la date de retour, recommencez : ")
    self.set_Datearrive(input("Datearrive: "))
    self.set_Datedepart(input("Datedepart: "))
```

```

#si l'année est égale ou supérieure, on teste les mois et jours

elif int(depart_decompo[2]) >= int(arrive_decompo[2]):

    #si le mois est inférieure

    if int(depart_decompo[1]) < int(arrive_decompo[1]):
        print("la date d'arrivée est inférieure à la date de retour, recommencez : ")
        self.set_Datearrive(input("Datearrive: "))
        self.set_Datedepart(input("Datedepart: "))

```

Puis, si l'année est égale ou supérieure, on teste les mois et les jours. Ici on commence par le mois.

```

#si le mois est égal

elif int(depart_decompo[1]) == int(arrive_decompo[1]) and int(depart_decompo[2]) == int(arrive_decompo[2]):

    if int(depart_decompo[0]) == int(arrive_decompo[0]):
        print("même date, recommencez : ")
        self.set_Datearrive(input("Datearrive: "))
        self.set_Datedepart(input("Datedepart: "))

```

Si le mois est égal, on va se pencher sur les jours.

```

        elif int(depart_decompo[0]) < int(arrive_decompo[0]):
            print("la date d'arrivée est inférieure à la date de retour, recommencez :")
            self.set_Datearrive(input("Datearrive: "))
            self.set_Datedepart(input("Datedepart: "))

        else:
            i=1

    else:
        i=1

else:
    i=1

```

Ici, nous testons les jours et si une des conditions est respectée pour la date, notre attribut prend la valeur 1 et la boucle s'arrête. Dans le cas contraire, le programme continue à l'infini tant qu'il n'est pas valide.

La méthode que nous avons ensuite faite est celle pour vérifier et confirmer l'adresse mail :

```
def Verif_Mail(self):
    i = 3
    while i !=4:

        if self.get_AdresseMail() != self.get_ConfirmAdresse():
            print("les adresses ne sont pas les mêmes, recommencez : ")
            self.set_AdresseMail(input("AdresseMail: "))
            self.set_ConfirmAdresse(input("ConfirmAdresse: "))

        email_decompo = self.get_ConfirmAdresse().split("@")

        if len(email_decompo) == 1:
            print("Votre adresse ne contient pas un @: ")
            self.set_AdresseMail(input("AdresseMail: "))
            +"Le tel est " + str(self.get_Tel())+ "\n"
            self.set_ConfirmAdresse(input("ConfirmAdresse: "))

        else:
            i = 4
```

Dans ce programme, nous avons aussi initialiser une boucle `while`

```
i = 3
while i !=4:
```

```
if self.get_AdresseMail() != self.get_ConfirmAdresse():
    print("les adresses ne sont pas les mêmes, recommencez : ")
    self.set_AdresseMail(input("AdresseMail: "))
    self.set_ConfirmAdresse(input("ConfirmAdresse: "))
```

La première condition sert à savoir si les adresses sont exactes lors de la saisie du texte.

```
email_decompo = self.get_ConfirmAdresse().split("@")
```

Nous décomposons ensuite la confirmation de l'adresse pour savoir avec un `.split("@")` pour savoir si l'adresse email contient bien un "@".


```

if len(email_decompo) == 1:
    print("Votre adresse ne contient pas un @: ")
    self.set_AdresseMail(input("AdresseMail: "))
    + "Le tel est " + str(self.get_Tel()) + "\n"
    self.set_ConfirmAdresse(input("ConfirmAdresse: "))

else:
    i = 4

```

Puis nous utilisons la longueur de la liste pour savoir s'il y a bien un "@" dans notre liste. Si elle est égale à 1, l'adresse n'est pas split donc elle ne contient pas de "@". Autrement, l'adresse est validée.

Nous avons ensuite fait la méthode pour calculer le nombre de jours en fonction des dates que l'on entre :

```

arrive_decompo=self.get_Datearrive().split("/")
depart_decompo=self.get_Datedepart().split("/")

self.set_Nbrnuit(int(depart_decompo[0]) - int(arrive_decompo[0]) + (int(depart_decompo[1])
- int(arrive_decompo[1]))*31 + (int(depart_decompo[2]) - int(arrive_decompo[2]))*365)

```

```

arrive_decompo=self.get_Datearrive().split("/")
depart_decompo=self.get_Datedepart().split("/")

```

Dans un premier temps, on split les "/" dans les dates pour récupérer les valeurs de la liste.

```

self.set_Nbrnuit(int(depart_decompo[0]) - int(arrive_decompo[0]) + (int(depart_decompo[1])
- int(arrive_decompo[1]))*31 + (int(depart_decompo[2]) - int(arrive_decompo[2]))*365)

```

Ensuite on soustrait les jours, les mois et les années pour savoir combien de nuit nous passons. Pour les mois, nous multiplions la valeur par 31, et pour les années, par 365 car nous calculons directement en jours.

Enfin, nous avons la méthode qui va afficher les résultats et les exécutions pour ce programme :

```
def Affiche(self):  
  
    return ("Votre nom est: " + self.get_Nom() + "\n"  
            + "Votré prénom: " + self.get_Nom() + "\n"  
  
            + "Adresse Mail: " + self.get_AdresseMail() + "\n"  
            + "Confirmation: " + self.get_ConfirmAdresse() + "\n"  
  
            + "Le tel est " + str(self.get_Tel()) + "\n"  
  
            + "Dates d'arrivée: " + str(self.get_Datearrive()) + "\n"  
            + "Dates de départ: " + str(self.get_Datedepart()) + "\n"  
            + "Nombre de nuits: " + str(self.get_Nbrnuit()) + "\n"  
            + "Type de chambre: " + str(self.get_TypeChambre()) + "\n"  
            + "Nombre de personnes: " + str(self.get_Nbrpersonnes()) + "\n"  
            + "Nombre de déjeuners: " + str(self.get_Nbrpetitdej()) + "\n"  
            + "Conforts: " + str(self.get_Confort()) + "\n")
```

Ici avec cette fonction, nous récupérons toutes les valeurs avec des `get()` pour ensuite les afficher.

```
Nom: Soragna  
Prenom: Quentin  
AdresseMail: quentin.soragna@etu.univ-amu.fr  
ConfirmAdresse: quentin.soragna@etu.univ-amu.fr  
Tel: 07831919494  
Datearrive: 02/04/2023  
Datedepart: 03/04/2023  
Typechambre: 1 chambre  
Nbrpersonnes: 3  
NbrDeDeuj: 2  
Confort: TV  
  
Votre nom est: Soragna  
Votré prénom: Soragna  
Adresse Mail: quentin.soragna@etu.univ-amu.fr  
Confirmation: quentin.soragna@etu.univ-amu.fr  
Le tel est 07831919494  
Dates d'arrivée: 02/04/2023  
Dates de départ: 03/04/2023  
Nombre de nuits: 1  
Type de chambre: 1 chambre  
Nombre de personnes: 3  
Nombre de déjeuners: 2  
Conforts: TV
```

```

h = Hotel()
h.get_data()
print(" ")
h.calcul_date()
h.Verif_Mail()
h.Calcul_nuit()
print(h.Affiche())

```

Ensuite, ceci représente les exécutions des méthodes pour le bon fonctionnement de notre class `Hotel`.

Suite à la partie python, nous allons commencer la création de notre interface graphique avec `Kivy/Kv`.

La première chose que nous avons faite est d'importer les modules nécessaires pour créer notre interface :

```

import kivy
from kivy.app import App
from kivy.uix.boxlayout import BoxLayout
from kivy.lang import Builder
from kivy.uix.image import Image
from kivy.config import Config
from kivy.uix.textinput import TextInput
from kivy.uix.widget import Widget
from kivy.uix.button import Button
from kivy.uix.label import Label
from kivy.properties import ObjectProperty
from kivy.core.window import Window
from kivy.uix.screenmanager import ScreenManager, Screen

import sys
from kivy.uix.floatlayout import FloatLayout
from Prog_Hotel import Hotel

```

```

Window.size = (800,950)

ProgHotel=Hotel()

```

Par la suite, je définis ma classe `Hotel` avec la variable `ProgHotel`.

```

class SousClasseHotel(Screen):

    nom=ObjectProperty(None)
    prenom=ObjectProperty(None)
    eMail=ObjectProperty(None)
    confirm=ObjectProperty(None)
    tel=ObjectProperty(None)
    date_arrive=ObjectProperty(None)
    date_depart=ObjectProperty(None)
    nbr_nuit=ObjectProperty(None)
    typechambres=ObjectProperty(None)
    nbr_personnes=ObjectProperty(None)
    nbr_petitdej=ObjectProperty(None)
    confort=ObjectProperty(None)
    erreur_email=ObjectProperty(None)
    erreur_date=ObjectProperty(None)
    erreur_info=ObjectProperty(None)

```

Ensuite, je définis les `ObjectProperty(None)` qui seront utiles pour stocker des informations dans ces variables pour ensuite les afficher sur l'interface graphique.

```

<SousClasseHotel>:
    canvas.before:
        Color:
            rgba: (.6, .07, .07, .6)

        Rectangle:
            pos: self.pos
            size: self.size

    title: 'Hotel'
    auto_dismiss: False
    id: SAE
    name: "premier"

    nom: name
    prenom: prenom
    eMail: email
    confirm: confirm
    tel: tel
    typechambres: cham
    date_arrive: datea
    date_depart: dated
    nbr_nuit: nuit
    nbr_petitdej: dej
    nbr_personnes: pers
    confort: conf
    erreur_email: erreur_e
    erreur_date: erreur_d
    erreur_info: erreur_i

```

La première chose à faire suite à cela est de créer une fenêtre graphique pour pouvoir commencer à placer nos informations dessus.

```
canvas.before:
    Color:
        rgba:(.6,.07,.07,.6)

    Rectangle:
        pos: self.pos
        size: self.size
```

Ici, nous avons définis notre couleur de fond d'écran et les positions et la taille avec `self.pos` et `self.size`

```
title: 'Hotel'
auto_dismiss: False
id: SAE
name: "premier"

nom: name
prenom: prenom
eMail: email
confirm: confirm
tel: tel
typechambres: cham
date_arrive: datea
date_depart: dated
nbr_nuit: nuit
nbr_petitdej: dej
nbr_personnes: pers
confort: conf
erreur_email: erreur_e
erreur_date: erreur_d
erreur_info: erreur_i
```

Nous avons ensuite le titre de la fenêtre, le nom de celle-ci, son id et aussi les `ObjetProperty(None)` liés à notre classe.

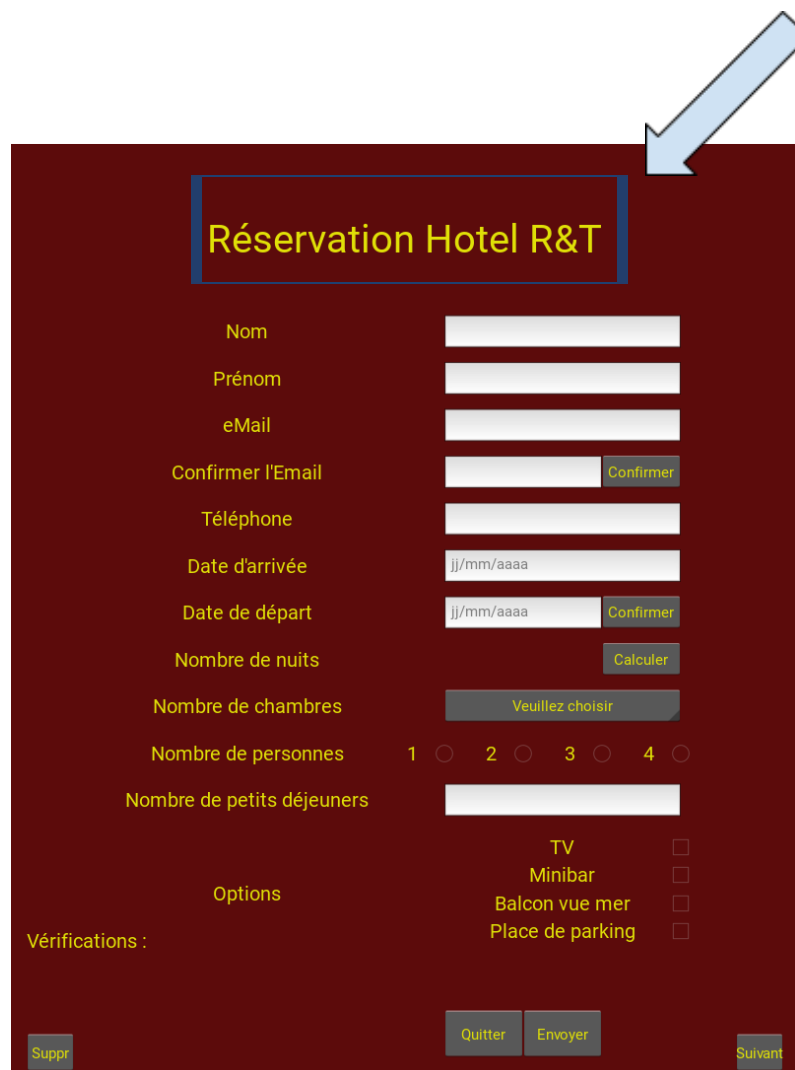
```
RelativeLayout:
    orientation:'vertical'

    pos : self.pos
    size : root.size

    Label:
        text: ("[color=e0e007]Réservation Hotel R&T[/color]")
        markup: True
        font_size:40
        size_hint: 0.2,0.2
        pos_hint:{'center_x':.5, 'center_y':.9}
```

La présentation de la page et le placement de données se fait en `RelativeLayout` pour pouvoir poser manuellement et facilement tout ce dont nous avons besoin.

Nous avons aussi un exemple de Label Pour afficher le titre



The screenshot shows a web form for hotel reservations. At the top, a yellow title 'Réservation Hotel R&T' is centered in a blue-bordered box. A blue arrow points to this title. Below the title, the form is organized into two columns. The left column contains labels for 'Nom', 'Prénom', 'eMail', 'Confirmer l'Email', 'Téléphone', 'Date d'arrivée', 'Date de départ', 'Nombre de nuits', 'Nombre de chambres', 'Nombre de personnes', and 'Nombre de petits déjeuners'. The right column contains corresponding input fields: text boxes for name, first name, email, and phone; a date picker for arrival; a date picker for departure with a 'Confirmer' button; a dropdown for the number of nights; a dropdown for the number of rooms; radio buttons for the number of people (1, 2, 3, 4); and a text box for the number of breakfasts. Below these, there are checkboxes for 'Options' including 'TV', 'Minibar', 'Balcon vue mer', and 'Place de parking'. At the bottom, there are buttons for 'Suppr', 'Quitter', 'Envoyer', and 'Suivant'.

```

Label:
  text: ("[color=e0e007]Nom[/color]")
  markup: True
  font_size:20
  size_hint: 0.1,0.1
  pos_hint:{'center_x':.3, 'center_y':.8}

Label:
  text: ("[color=e0e007]Prénom[/color]")
  markup: True
  font_size:20
  size_hint: 0.1,0.1
  pos_hint:{'center_x':.3, 'center_y':.75}

Label:
  text: ("[color=e0e007]eMail[/color]")
  markup: True
  font_size:20
  size_hint: 0.1,0.1
  pos_hint:{'center_x':.3, 'center_y':.70}

```



Réservation Hotel R&T

Nom

Prénom

eMail

Confirmer l'Email

Téléphone

Date d'arrivée

Date de départ

Nombre de nuits

Nombre de chambres

Nombre de personnes

Nombre de petits déjeuners

Options

Confirmer

 Confirmer

1 ☐ 2 ☐ 3 ☐ 4 ☐

TV ☐

Minibar ☐

Balcon vue mer ☐

Place de parking ☐

Vérifications :

Cette partie du programme est affichée par des labels qui seront placés avec `pos_hint:`

```
TextInput:
  id: name
  multiline: False
  size_hint: 0.3,0.035
  pos_hint: {'center_x':.7, 'center_y':.8}
```

```
TextInput:
  id: prenom
  multiline: False
  size_hint: 0.3,0.035
  pos_hint: {'center_x':.7, 'center_y':.75}
```

```
Button:
  text: ("[color=e0e007]Confirmer[/color]")
  markup: True
  size_hint: 0.1,0.035
  pos_hint: {'center_x':.8, 'center_y':.65}
  on_press: root.btn_mail()
```

```
Button:
  text: ("[color=e0e007]Calculer[/color]")
  markup: True
  size_hint: 0.1,0.035
  pos_hint: {'center_x':.8, 'center_y':.45}
  on_press: root.btn_nuit()
```



Réservation Hotel R&T

Nom

Prénom

eMail

Confirmer l>Email Confirmer

Téléphone

Date d'arrivée

Date de départ Confirmer

Nombre de nuits

Nombre de chambres

Nombre de personnes 1 ☐ 2 ☐ 3 ☐ 4 ☐

Nombre de petits déjeuners

Options

TV ☐

Minibar ☐

Balcon vue mer ☐

Place de parking ☐

Vérifications :

Confirmer

Confirmer

Calculer

Veuillez choisir

Quitter
Envoyer
Suivant

Dans cette partie de la configuration nous allons nous pencher sur plusieurs points. Dans un premier temps, les cases vides représentent les `TextInput` qui sont affiliés aux variables "name", "prenom" par exemple. Donc quand nous allons écrire dans ces input, les valeurs seront stockées et définies comme des setters.

Comme cet exemple :

```
def info(self):
    ProgHotel.set_Nom(self.nom.text)
    ProgHotel.set_Prenom(self.prenom.text)
    ProgHotel.set_AdresseMail(self.eMail.text)
    ProgHotel.set_ConfirmAdresse(self.confirm.text)
    ProgHotel.set_Tel(self.tel.text)
    ProgHotel.set_Datearrive(self.date_arrive.text)
    ProgHotel.set_Datedepart(self.date_depart.text)
    ProgHotel.set_Nbrpetitdej(self.nbr_petitdej.text)
```


Le deuxième point est le bouton que nous avons créé pour confirmer l'adresse mail.

`on_press: root.btn_mail()` signifie que quand nous allons appuyer sur le bouton, la méthode `btn_mail()` va se lancer.

```
def btn_mail(self):

    ProgHotel.set_AdresseMail(self.eMail.text)
    #print(self.eMail.text)

    ProgHotel.set_ConfirmAdresse(self.confirm.text)
    #print(self.confirm.text)

    i = 3
    while i !=4:

        if ProgHotel.get_AdresseMail() != ProgHotel.get_ConfirmAdresse():

            self.erreur_email.text = "Email non valide"
            self.eMail.text = ""
            self.confirm.text = ""
            break

        email_decompo = ProgHotel.get_ConfirmAdresse().split("@")

        if len(email_decompo) == 1:
            self.erreur_email.text = "Email non valide"
            self.eMail.text = ""
            self.confirm.text = ""
            break

        else:
            i = 4
            self.erreur_email.text = "Email valide"

    if ProgHotel.get_AdresseMail() == "" or ProgHotel.get_ConfirmAdresse() == "":
        self.erreur_email.text = "Email manquant"
```

Nous pouvons constater que cette version est pratiquement la même que notre méthode que nous avons créé dans notre classe `Hotel` mais cette fois ci, elle est adaptée pour kivy et l'interface graphique

```
self.erreur_email.text = "Email non valide"
self.eMail.text = ""
self.confirm.text = ""
```

Le changement se fait sur ses parties là où nous allons écrire directement notre texte.

Le bouton du calcul de nuit s'effectue aussi avec

```
on_press: root.btn_nuit()
```

```
def btn_nuit(self):

    ProgHotel.set_Datearrive(self.date_arrive.text)
    #print(self.date_arrive.text)

    ProgHotel.set_Datedepart(self.date_depart.text)
    #print(self.date_depart.text)

    ProgHotel.Calcul_nuit()

    self.nbr_nuit.text = str(ProgHotel.get_Nbrnuit())
```

Ce qui va nous donner cette méthode ou le programme de notre classe Hotel peut être directement importé vu que nous avons pas besoin de faire d'adaptation

```
ProgHotel.Calcul_nuit()
```

```
self.nbr_nuit.text = str(ProgHotel.get_Nbrnuit())
```

Enfin, nous convertissons le get en chaîne de caractère vu que .text n'accepte que les chaînes de caractères.

Maintenant, nous allons voir cette partie qui est plus complexe que les autres car elle fait appel aux notions de checkbox et de menu déroulant.

Réservation Hotel R&T

Nom

Prénom

eMail

Confirmer l'Email

Téléphone

Date d'arrivée

Date de départ

Nombre de nuits

Nombre de chambres

Nombre de personnes 1 ☐ 2 ☐ 3 ☐ 4 ☐

Nombre de petits déjeuners

Options

TV ☐

Minibar ☐

Balcon vue mer ☐

Place de parking ☐

Vérifications :

```
Spinner:
  id: cham
  text: ("[color=e0e007]Veuillez choisir[/color]")
  markup: True
  values: ['1 chambre', '2 chambres', '3 chambres', '4 chambres']
  size_hint: 0.3,0.035
  pos_hint: {'center_x':.7, 'center_y':.40}
  color: (0.85, 0.85, 0.05, 1)
```

Nous commençons avec **Spinner** pour le menu déroulant.

```
values: ['1 chambre', '2 chambres', '3 chambres', '4 chambres']
```

Cette ligne unique à **spinner** nous sert à créer les valeurs du menu pour ainsi les choisir:

Dans notre cas, nous avons un hôtel avec un maximum de 4 chambres.



Maintenant, nous allons parler de `CheckBox` et commencer avec le nombre de personnes. Sur notre fenêtre, nous pouvons voir que l'on peut sélectionner qu'une seule case arrondie.

Grâce à `group: "rond"` que nous avons appliqué sur les 4 `Checkbox`, une liaison s'est créée pour sélectionner qu'une seule checkbox ce qui est très utile dans ce cas.

```
CheckBox:
    id: pers
    group: "rond"
    size_hint:0.05,0.01
    pos_hint: {'center_x':.55, 'center_y':.35}
    on_active: root.checkbox_click(self, self.active, "1")
```

```
on_active: root.checkbox_click(self, self.active, "1")
```

Si je veux sélectionner la valeur 1, nous faisons appel à la méthode `checkbox_click()`.

```
def checkbox_click(self, instance, value, topping):

    if value == True:
        SousClasseHotel.checks.append(topping)
        tops = ''
        for x in SousClasseHotel.checks:
            tops = f'{tops} {x}'
        self.nbr_personnes.text = f'{tops}'

    else:
        SousClasseHotel.checks.remove(topping)
        tops = ''
        for x in SousClasseHotel.checks:
            tops = f'{tops} {x}'
        self.nbr_personnes.text = f'{tops}'
```

Avec cette méthode, dans un premier temps, si l'utilisateur clique sur la box, la valeur va devenir `True` :

C'est pour cela que si un utilisateur clique sur la box : `if value == True:`

`checks = []`

```
SousClasseHotel.checks.append(topping)
tops = ''
for x in SousClasseHotel.checks:
    tops = f'{tops} {x}'
self.nbr_personnes.text = f'{tops}'
```

La valeur va s'ajouter dans la liste vide préalablement créé avec la fonction `append()`.

```
else:
    SousClasseHotel.checks.remove(topping)
    tops = ''
    for x in SousClasseHotel.checks:
        tops = f'{tops} {x}'
    self.nbr_personnes.text = f'{tops}'
```

Si nous décochons l'option, la valeur se retire de la liste avec la fonction `remove()`.

Ensuite la valeur est stockée : `self.nbr_personnes.text = f'{tops}'`

Nous continuons de parler de **Checkbox** avec le choix des options. Le principe reste le même:

```
Checkbox:
    id: conf
    size_hint:0.01,0.01
    pos_hint: {'center_x':.85, 'center_y':.25}
    on_active: root.confort_click(self, self.active,"TV")
```

La construction pour le kv est la même mais aucun "group" n'est attribué pour laisser les cases carrées ainsi qu'un choix multiple. Quand nous cliquons sur la Checkbox "TV", nous faisons appel à la fonction **confort_click()**.

```
confo = []
```

```
def confort_click(self, instance, value1, topping)

    if value1 == True:
        SousClasseHotel.confo.append(topping)
        tops = ''
        for x in SousClasseHotel.confo:
            tops = f'{tops} {x}'
        self.confort.text = f'{tops}'

    else:
        SousClasseHotel.confo.remove(topping)
        tops = ''
        for x in SousClasseHotel.confo:
            tops = f'{tops} {x}'
        self.confort.text = f'{tops}'
```

Le principe reste le même, pour stocker les valeurs pour **self.confort.text**

La dernière partie de la fenêtre se résume à la configuration de boutons pour naviguer entre les fenêtres, pour supprimer les données si jamais l'utilisateur veut recommencer sa réservation et les vérifications nécessaires pour la valider.

Réservation Hotel R&T

| | | | |
|--|--|--|---|
| Nom | <input style="width: 100%;" type="text"/> | | |
| Prénom | <input style="width: 100%;" type="text"/> | | |
| eMail | <input style="width: 100%;" type="text"/> | | |
| Confirmer l'Email | <input style="width: 80%;" type="text"/> | <input style="width: 20%;" type="button" value="Confirmer"/> | |
| Téléphone | <input style="width: 100%;" type="text"/> | | |
| Date d'arrivée | <input style="width: 100%;" type="text" value="jj/mm/aaaa"/> | | |
| Date de départ | <input style="width: 80%;" type="text" value="jj/mm/aaaa"/> | <input style="width: 20%;" type="button" value="Confirmer"/> | |
| Nombre de nuits | <input style="width: 80%;" type="text"/> | | <input style="width: 20%;" type="button" value="Calculer"/> |
| Nombre de chambres | <input style="width: 100%;" type="button" value="Veuillez choisir"/> | | |
| Nombre de personnes | 1 | <input type="radio"/> | 2 <input type="radio"/> |
| | | | 3 <input type="radio"/> |
| | | | 4 <input type="radio"/> |
| Nombre de petits déjeuners | <input style="width: 100%;" type="text"/> | | |
| Options | | | |
| | TV | <input type="checkbox"/> | |
| | Minibar | <input type="checkbox"/> | |
| | Balcon vue mer | <input type="checkbox"/> | |
| | Place de parking | <input type="checkbox"/> | |
| Vérifications : | | | |
| | | | |
| <input style="width: 50%;" type="button" value="Suppr"/> | | <input style="width: 50%;" type="button" value="Quitter"/> | <input style="width: 50%;" type="button" value="Envoyer"/> |
| | | <input style="width: 50%;" type="button" value="Suivant"/> | |

Tout d'abord, nous avons nous vérifications qui assurent que l'utilisateurs à bien remplie tous les champs et correctement :

Vérifications :

Email valide
dates valides

Vérifications :

Infos manquantes

Vérifications :

Email valide
dates valides
Informations envoyées

Vérifications :

Confirmation manquante
dates valides
numéro non valide

Nous avons ensuite le bouton qui sert à supprimer tous les champs pour recommencer sa réservation:

```
def suppr(self):  
    self.nom.text = ""  
    self.prenom.text=""  
    self.eMail.text = ""  
    self.confirm.text=""  
    self.tel.text=""  
    self.date_arrive.text=""  
    self.date_depart.text=""  
    self.nbr_nuit.text = ""  
    self.nbr_petitdej.text = ""  
    self.erreur_date.text=""  
    self.erreur_email.text=""  
    self.erreur_info.text = ""
```

Il suffit juste de remettre les valeurs avec des chaînes de caractères vides.

Le bouton envoyer nous sert à envoyer nos informations entrées dans la base de données.

```
Button:  
    text: ("[color=e0e007]Envoyer[/color]")  
    markup: True  
    size_hint: 0.1,0.05  
    pos_hint: {'center_x':.7, 'center_y':.05}  
    on_press:  
        root.envoié()
```

Ce bouton fait appel à la méthode `envoié()`.

```
def envoié(self):  
  
    if self.nom.text == "":  
        self.erreur_info.text = "Infos manquantes"  
  
    elif self.prenom.text == "":  
        self.erreur_info.text = "Infos manquantes"  
  
    elif self.eMail.text == "":  
        self.erreur_info.text = "Infos manquantes"
```



```

else:
    self.erreur_info.text = "Informations envoyées"
    SousClasseHotel.info(self)

```

Les conditions sont testées dans cette fonction pour savoir s'il manque des informations avant d'envoyer les informations.

Si aucune information manque à l'appel, nous exécutons la méthode `info()`.

```

def info(self):
    ProgHotel.set_Nom(self.nom.text)
    ProgHotel.set_Prenom(self.prenom.text)
    ProgHotel.set_AdresseMail(self.eMail.text)
    ProgHotel.set_ConfirmAdresse(self.confirm.text)
    ProgHotel.set_Tel(self.tel.text)
    ProgHotel.set_Datearrive(self.date_arrive.text)
    ProgHotel.set_Datedepart(self.date_depart.text)
    ProgHotel.set_Nbrpetitdej(self.nbr_petitdej.text)

```

Elle permettra de stocker dans les variables les données que nous rentrons.

Nous avons ensuite le bouton quitter qui fait appel à la méthode `btn_exit()`

```

Button:
    text: ("[color=e0e007]Quitter[/color]")
    markup: True
    size_hint: 0.1,0.05
    pos_hint: {'center_x':.6, 'center_y':.05}
    on_press: root.btn_exit()

```

```

def btn_exit(self):
    sys.exit(0)

```

Cette méthode très simple permet de quitter l'application.

Nous avons maintenant le bouton suivant qui permet de naviguer sur le second onglet ci-dessous :

Réservation Hotel R&T

Pour voir votre facture, cliquez sur ce bouton : [Voir](#)

Facture pour M/Me :

Base :

Nombre de nuits :

Nombre de petits déjeuners :

Options :

Prix total :

Confirmez votre facture : [Confirmer](#)

Identifiez vous : [Ici](#)

Quittez : [Ici](#)

[Retour](#)

Pour se faire, j'ai du utiliser la classe :

```
class WindowManager(ScreenManager):  
    pass
```

Dans mon script kv, la création se fait comme ceci :

```
WindowManager:  
    SousClasseHotel:  
        Facture:
```

```

Button:
    text: ("[color=e0e007]Suivant[/color]")
    markup: True
    size_hint: 0.06,0.04
    pos_hint:{'center_x':.95, 'center_y':.03}
    on_press:
        app.root.current = "second"
        root.manager.transition.direction = "left"

```

La particularité de ce bouton est le changement d'onglet. En effet avec :

```

on_press:
    app.root.current = "second"
    root.manager.transition.direction = "left"

```

Nous avons la transition qui se fait dans la direction gauche se dirigeant vers la fenêtre nommée : **"second"**

```

<Facture>:
    canvas.before:
        Color:
            rgba:(.6,.07,.07,.6)
        Rectangle:
            pos: self.pos
            size: self.size

    title:'Hotel2'
    auto_dismiss: False
    name: "second"

```

La particularité de cette page se compose en deux parties :

Réservation Hotel R&T

Pour voir votre facture, cliquez sur ce bouton :

Facture pour M/Me :

Base :

Nombre de nuits :

Nombre de petits déjeuners :

Options :

Prix total :

Confirmez votre facture :

Identifiez vous :

Quittez :

Cette première partie est la récupération des données pour le récapitulatif de la réservation du client :

Pour voir votre facture, cliquez sur ce bouton :

Facture pour M/Me :

Base :

Nombre de nuits :

Nombre de petits déjeuners :

Options :

Prix total :

Confirmez votre facture :

Nous avons un bouton “voir” qui permet d’afficher la facture en elle même :

```
Button:
    text: ("[color=e0e007]Voir[/color]")
    markup: True
    size_hint: 0.1,0.05
    pos_hint: {'center_x':.8, 'center_y':.75}
    on_press:
        root.calculer()
        root.reception()
```

Il fait appel à deux méthodes : `calculer()` et `reception()`

```
def calculer(self):

    self.base.int = 60
    ops = 20

    self.prix.text = str(int(ProgHotel.get_Nbrnuit()) * (self.base.int + (self.base.int * 0.2)) + int(ProgHotel.get_Nbrpetitdej()) * 10 + (ops + (ops * 0.2))) + " €"
    self.base.text = str(self.base.int) + " €"

    self.result_nuit.text = str(ProgHotel.get_Nbrnuit())
    self.dej.text = str(ProgHotel.get_Nbrpetitdej())

    self.opt.text = ', '.join(SousClasseHotel.confo)
```

Avec cette méthode, nous récupérerons le prix à travers un calcul, nous le convertissons la base en str pour l'afficher ainsi que le nombre de nuitées et le nombre de petits déjeuners.

La dernière ligne de cette méthode avec la fonction `.join()` permet à l'affichage, de séparer les résultats de la liste confo.

Le bouton confirmer nous permet de récupérer le nom et l'adresse mail de l'utilisateur pour ensuite se connecter pour voir sa réservation.

```
Button:
    text: ("[color=e0e007]Confirmer[/color]")
    markup: True
    size_hint: 0.1,0.05
    pos_hint:{'center_x':.72, 'center_y':.27}
    on_press:
        root.confirm()
```

Le bouton, une fois appuyé, fait appel à la méthode `confirm()`.

```
Facture.l_iden.clear()

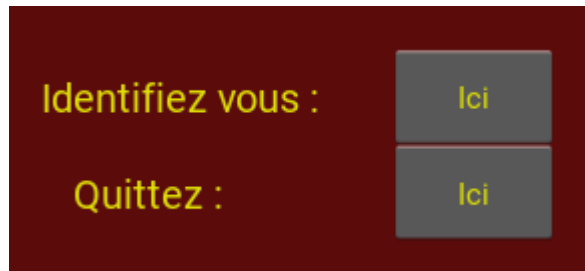
print(self.prix.text)

Facture.l_iden.append(ProgHotel.get_Nom())
Facture.l_iden.append(ProgHotel.get_AdresseMail())

print(Facture.l_iden)
```

Elle va permettre de mettre dans une liste les informations précédentes énumérées comme le nom et l'adresse mail

La deuxième partie de cet onglet traite deux boutons, le premier le bouton pour accéder à un autre onglet pour s'identifier pour sa réservation et l'autre, un bouton pour quitter.



```
Button:
    text: ("[color=e0e007]Ici[/color]")
    markup: True
    size_hint: 0.1,0.05
    pos_hint:{'center_x':.3, 'center_y':.1}
    on_press:
        app.root.current = "third"
        root.manager.transition.direction = "left"
```

Le bouton pour s'identifier utilise le même système comme pour notre 1ère et 2nd page.

```
Button:
    text: ("[color=e0e007]Ici[/color]")
    markup: True
    size_hint: 0.1,0.05
    pos_hint:{'center_x':.3, 'center_y':.05}
    on_press: root.quitter()
```

Le bouton pour quitter utilise le même système comme pour notre 1ère page.

Résultat :

Réservation Hotel R&T

Pour voir votre facture, cliquez sur ce bouton : [Voir](#)

Facture pour M/Me : Soragna Quentin

| | |
|---|---------|
| Base : | 60 € |
| Nombre de nuits : | 1 |
| Nombre de petits déjeuners : | 4 |
| Options : TV, Minibar, Balcon vue mer, Place de Parking | |
| Prix total : | 136.0 € |

Confirmez votre facture : [Confirmer](#)

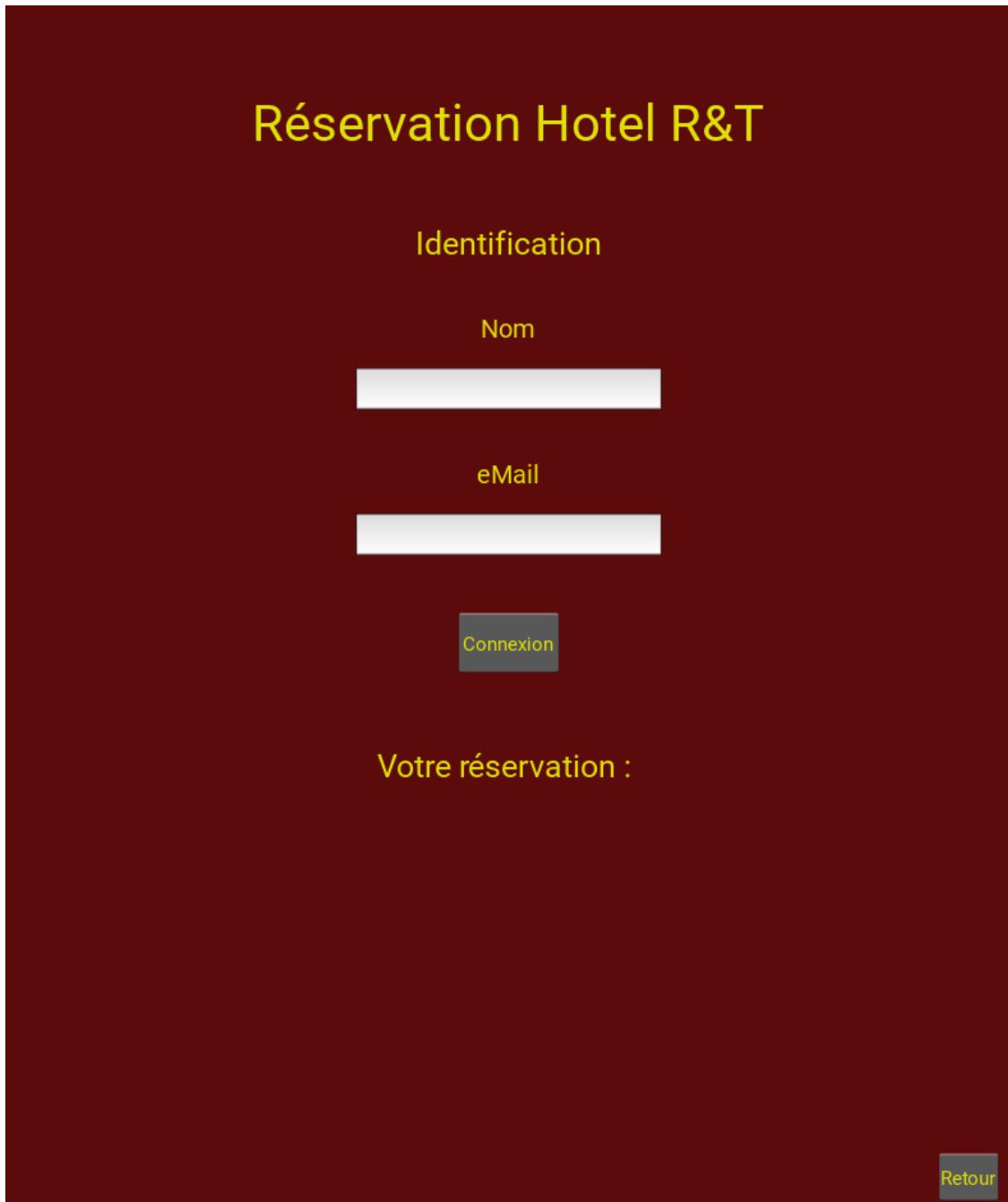
Information enregistrée

Identifiez vous : [Ici](#)

Quittez : [Ici](#)

[Retour](#)

Une fois que l'utilisateur souhaite s'authentifier, nous tombons sur cette page :



Réservation Hotel R&T

Identification

Nom

eMail

Connexion

Votre réservation :

Retour

C'est à partir de ce moment-là que nous commençons à traiter la partie base de données.

Tout d'abord, avant de commencer de faire des commandes sql via python nous devons créer la database. Pour cela on se connecte à la VM où l'on va stocker la base de données puis on démarre MariaDB pour créer la database, ici Hotel avec la commande CREATE DATABASE Hotel :

Ensuite on crée un utilisateur qui aura les droits sur la base de données avec les commandes

CREATE USER 'user'@'localhost' IDENTIFIED BY 'mot_de_passe':

GRANT ALL PRIVILEGES ON * . * TO 'nouveau_utilisateur'@'localhost':

FLUSH PRIVILEGES:

Maintenant qu'on a fait cela, on retourne sur le programme python et on va se connecter à la base de données située sur une machine virtuelle ayant comme adresse 10.10.10.41. Pour cela on utilise le module python mysql.connector avec comme paramètre l'adresse ip, le nom d'utilisateur et le mot de passe pour se connecter à la base de données ainsi que la database, ici 'Hotel'. Après cela, on exécute des commandes sql qui vont créer les tables nécessaires si elles ne sont pas déjà créées.

```
conn = mysql.connector.connect(host="10.10.10.41",
                                user="user",
                                password="user",
                                database='Hotel')
cursor = conn.cursor()

cursor.execute("""
CREATE TABLE IF NOT EXISTS chambres(
    num_chambre int NOT NULL AUTO_INCREMENT,
    type_chambre varchar(30) DEFAULT NULL,
    nb_personne int(10) DEFAULT NULL,
    nb_nuit int(10) DEFAULT NULL,
    option varchar(60) DEFAULT NULL,
    prix varchar(34) DEFAULT NULL,
    PRIMARY KEY(num_chambre)
);
""")
conn.commit()

print("oeoeoeo")

cursor.execute("""
CREATE TABLE IF NOT EXISTS connexion(
    num_client int NOT NULL AUTO_INCREMENT,
    nom varchar(30) DEFAULT NULL,
    email varchar(50) DEFAULT NULL,
    PRIMARY KEY(num_client)
);
""")
conn.commit()
```

```

cursor.execute("""
    CREATE TABLE IF NOT EXISTS clients(
        num_client int NOT NULL AUTO_INCREMENT,
        nom varchar(30) DEFAULT NULL,
        prenom varchar(30) DEFAULT NULL,
        telephone varchar(30) DEFAULT NULL,
        email varchar(50) DEFAULT NULL,
        PRIMARY KEY (num_client)
    );
""")

conn.commit()

cursor.execute("""
    CREATE TABLE IF NOT EXISTS reservation(
        num_client int NOT NULL AUTO_INCREMENT,
        num_chambre integer,
        date_arrive varchar(30) DEFAULT NULL,
        date_depart varchar(30) DEFAULT NULL,
        nb_petit_dej int(10) DEFAULT NULL,
        PRIMARY KEY (num_client)
    );
""")

conn.commit()

```

Par la suite, nous allons ajouter toutes les données entrées dans l'interface kv dans la base de données. Pour cela, nous allons en premier faire des dictionnaires pour toutes les données qu'on veut entrer dans chaque table. Les self.x.text représentent les données récupérées dans l'interface kv.

```

data_chambre= {
    "type_chambre": self.typechambres.text,
    "nb_personne": self.nbr_personnes.text,
    "nb_nuit":self.nbr_nuit.text,
    "option":self.confort.text,
    "prix":prix,
}

data_client= {
    'nom':self.nom.text,
    'prenom':self.prenom.text,
    'tel':self.tel.text,
    'email':self.eMail.text,
}

data_reserv= {
    'num_chambre':random.randint(1,100),
    'date_arrive':self.date_arrive.text,
    'date_depart':self.date_depart.text,
    'nb_petit_dej':self.nbr_petitdej.text
}

data_connexion= {
    'nom':self.nom.text,
    'email':self.eMail.text,
}

```

Ces dictionnaires servent ensuite à ajouter les valeurs dans les tables grâce à un insert. Pour entrer les valeurs du dictionnaire, on stocke la commande dans une variable puis dans VALUES on met entre parenthèses le nom de la valeurs que l'on veut entourer d'un '%' et d'un 's' pour que cela prenne la valeur du dictionnaire. Enfin, on exécute la commande avec un exécute et entre parenthèses la requête et le dictionnaire.

A chaque fois que l'on modifie, crée ou supprime des tables, on doit entrer la commande conn.commit() pour effectuer les changements sur la base de données.

```
sql_chambre="INSERT INTO chambres(type_chambre, nb_personne, nb_nuit, option,prix) VALUES (%(type_chambre)s, %(nb_per
cursor.execute(sql_chambre, data_chambre)
conn.commit()

sql_client="INSERT INTO clients(nom, prenom, telephone, email) VALUES (%(nom)s, %(prenom)s, %(tel)s,%(email)s)"
cursor.execute(sql_client, data_client)
conn.commit()

sql_reserv="INSERT INTO reservation(num_chambre, date_arrive, date_depart, nb_petit_dej) VALUES (%(num_chambre)s,%(da
cursor.execute(sql_reserv, data_reserv)
conn.commit()

sql_connexion="INSERT INTO connexion (nom, email) SELECT %(nom)s, %(email)s WHERE NOT EXISTS(SELECT 1 FROM connexion
cursor.execute(sql_connexion,data_connexion)
-----+

nb_nuit, option,prix) VALUES (%(type_chambre)s, %(nb_personne)s, %(nb_nuit)s,%(option)s,%(prix)s)"

l) VALUES (%(nom)s, %(prenom)s, %(tel)s,%(email)s)"

, date_depart, nb_petit_dej) VALUES (%(num_chambre)s,%(date_arrive)s, %(date_depart)s, %(nb_petit_dej)s)"

om)s, %(email)s WHERE NOT EXISTS(SELECT 1 FROM connexion WHERE email = %(email)s)"
```

Maintenant qu'on a créé et rempli les tables on peut vérifier notre travail en allant sur la VM et interroger la base de données avec des requêtes simple pour voir les tables et leur contenu

```
MariaDB [Hotel]> show tables;
+-----+
| Tables_in_Hotel |
+-----+
| chambres        |
| clients          |
| connexion        |
| reservation      |
+-----+
4 rows in set (0.001 sec)
```

```
MariaDB [Hotel]> select * from clients;
+-----+-----+-----+-----+-----+
| num_client | nom      | prenom | telephone | email                                     |
+-----+-----+-----+-----+-----+
| 1          | Ocana    | Kevin  | 0695292472 | kevin.ocana@etu.univ-amu.fr             |
| 2          | Soragna  | Quentin| 0783414040 | quentin.soragna@etu.univ-amu.fr         |
+-----+-----+-----+-----+-----+
2 rows in set (0.000 sec)
```

```
MariaDB [Hotel]> select * from chambres;
+-----+-----+-----+-----+-----+-----+
| num_chambre | type_chambre | nb_personne | nb_nuit | option | prix |
+-----+-----+-----+-----+-----+-----+
| 1 | 1 chambre | 2 | 4 | Minibar TV Balcon vue mer Place de Parking | 342.0 |
| 2 | 4 chambres | 4 | 1 | TV Minibar Balcon vue mer Place de Parking | 136.0 |
+-----+-----+-----+-----+-----+-----+
2 rows in set (0.002 sec)
```

```
MariaDB [Hotel]> select * from connexion;
+-----+-----+-----+
| num_client | nom | email |
+-----+-----+-----+
| 1 | Ocana | kevin.ocana@etu.univ-amu.fr |
| 2 | Soragna | quentin.soragna@etu.univ-amu.fr |
+-----+-----+-----+
2 rows in set (0.001 sec)
```

```
MariaDB [Hotel]> select * from reservation;
+-----+-----+-----+-----+-----+-----+
| num_client | num_chambre | date_arrive | date_depart | nb_petit_dej |
+-----+-----+-----+-----+-----+-----+
| 1 | 72 | 14/02/2023 | 18/02/2023 | 3 |
| 2 | 67 | 02/04/2023 | 03/04/2023 | 4 |
+-----+-----+-----+-----+-----+-----+
2 rows in set (0.001 sec)
```

Réservation Hotel R&T

Nom

Prénom

eMail

Confirmer l'Email

Téléphone

Date d'arrivée

Date de départ

Nombre de nuits

Nombre de chambres

Nombre de personnes

Nombre de petits déjeuners

Options

Vérifications :

☐ 1
 ☒ 2
 ☐ 3
 ☐ 4

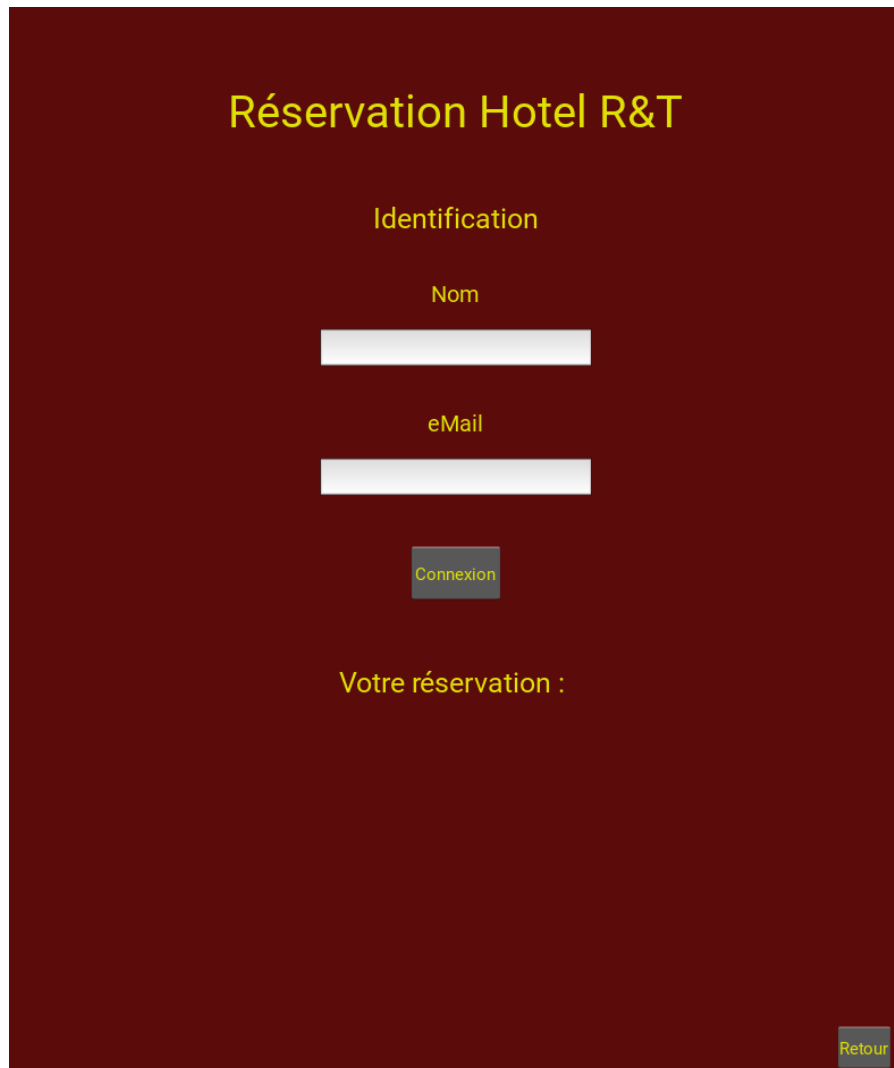
☒ TV
 ☒ Minibar
 ☒ Balcon vue mer
 ☒ Place de parking

Email valide

dates valides

Informations envoyées

Dans notre interface kv, nous avons une page qui sert à se connecter et récupérer les informations sur votre réservation avec votre nom et votre email.



Pour pouvoir faire cela, nous devons récupérer les infos dans la table réservation grâce à l'ID du client.

Pour cela on doit d'abord se reconnecter à la base avec la commande suivante car on a changé de fonction et surtout de class

```
conn = mysql.connector.connect(host="10.10.10.41",  
                                user="user",  
                                password="user",  
                                database='Hotel')  
cursor = conn.cursor()
```

Ensuite, on effectue une requête sql pour récupérer l'id du client grâce à son nom et son email dans la table connexion.

Si le client n'a pas encore fait de réservation le `cursor.fetchall()` va renvoyer une liste vide donc si c'est le cas on affiche Utilisateur Inconnu, si ce n'est pas le cas, on récupère uniquement le chiffre de l'ID avec un `client[0][0]`

```
cursor.execute("select num_client from connexion where nom = '"+str(self.nom.text)+"' and email = '"+str(self.mail.text)+"'")
client=cursor.fetchall()
if client==[]:
    self.erreur.text = "Utilisateur inconnu"
else:
    client=client[0][0]
    print(client)

    self.affi_client.text = str(client)
```

Maintenant qu'on a l'ID du client, nous devons récupérer les informations de la réservation du client. Pour cela, on effectue une requête select dans la table réservation avec comme condition l'ID du client.

Enfin, on stocke dans des variables chaque valeur.

```
cursor.execute("select num_chambre,nb_petit_dej,date_arrive,date_depart from reservation where num_client = '"+str(client)+"'")
resultat = cursor.fetchall()
n_chambre = str(resultat[0][0])
n_petit_dej = str(resultat[0][1])
d_arrive = str(resultat[0][2])
d_depart = str(resultat[0][3])

self.affi_chambre.text= str(n_chambre)
self.affi_petit_dej.text= str(n_petit_dej)
self.affi_arrive.text= str(d_arrive)
self.affi_depart.text= str(d_depart)
```

On obtient donc cela lorsqu'on appuie sur le bouton connexion.

Réservation Hotel R&T

Identification

Nom

Soragna

eMail

quentin.soragna@etu.univ-amu.fr

Connexion

Votre réservation :

N° Client : 2

N° Chambre : 67

Nombre de déjeuners : 4

Date d'arrivée : 02/04/2023

Date de départ : 03/04/2023

Retour