

RAPPORT SUR LE SITE WEB Hôtel en Flask



Florent Escots

15/03/2021

Licence professionnelle ADSILLH

Table des matières

URL permettant de démarrer le projet flask hôtel.....	3
I – Module ProgWeb.....	3
1.1) Menu	3
1.2) Page “Réservé choix date ” :	4
1.3) Page “Choix chambre ”:	5
1.4) Page “Validation de la réservation” :	6
1.5) Onglet « Ajout d’une consommation » :	8
1.6) Onglet « Payer votre facture » :	10
1.7) Onglet « Sign Up » :	12
1.8) Onglet « Login » :	13
1.9) Onglet « Faire un commentaire » :	14
II – Les problèmes rencontrés	16
III – Amélioration de la base de données	17
III – Amélioration futur :	17

URL permettant de démarrer le projet flask hôtel.

Mon application web est hébergé au CREMI voici le lien permettant de tester l'application :

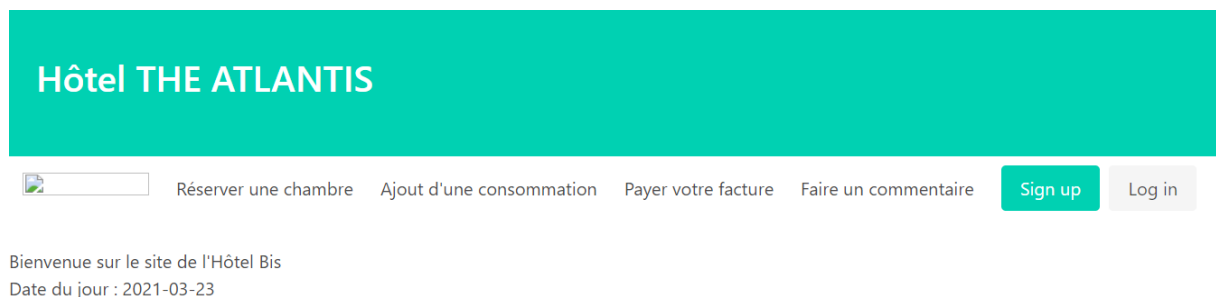
<https://florent-escots.emi.u-bordeaux.fr/flask/>

Dans les parties suivantes je vais détailler le fonctionnement de mon site web hôtel en flask. (ProgWeb et Nosql). Je vous détaillais par la suite les problèmes rencontrés. Je finirai mon rapport par vous expliquer les évolutions que j'ai effectuées de ma base de données et l'amélioration possible à effectuer sur mon projet hôtel.

I – Module ProgWeb

1.1) Menu

Pour commencer mon site web j'ai commencé par créer un menu dans lequel on y retrouve quelqu'une des pages qui sera utile pour l'utilisateur :



Extrait de code de hotel.html

```
<form method="POST" action="{url_for('menu_acceuil')}">
  <div id="navbarBasicExample" class="navbar-menu">
    <div class="navbar-start">
      <a class="navbar-item">
        <input type="submit" name="accueil" value="Réserver une chambre">
      </a>

      <a class="navbar-item">
        <input type="submit" name="accueil" value="Ajout d'une
consommation">
      </a>
    </div>
  </div>
</form>
```

Extrait de App.py

```
def menu accueil(error=None):
    session['accueil'] = request.form['accueil']
    if (session['accueil'] == "Réserver une chambre"):
        return render_template("choix_dates.html",
    hasError=error,connected=session['connexion'])
    if (session['accueil'] == "Ajout d'une consommation"):
        return render_template("ajout_consommation.html",
    hasError=error)
```

J'ai repris l'exemple du menu pour réserver une chambre et ajout d'une consommation ainsi que pour les autres champs. Dans un premier temps dans le fichier HTML j'ai ajouté 2 inputs qui ont comme nom « accueil ». Dans mon .py j'ai récupérer les valeurs avec : `session['accueil'] = request.form['accueil']`

Ce code permet de récupérer les valeurs des champs « accueil » et de les mettre dans une sessions.

Une fois les valeurs récupérées j'ai juste eu à faire une condition si la session accueil = réservé une chambre (valeur de l'input) alors il me retourne la page associé. J'ai effectué la même opération pour le reste des champs de mon menu.

Le menu est affiché sur toutes les pages du coup j'appel mon fichier hotel.html pour retrouver le menu à chaque page :

```
{% extends "hotel.html" %}
{% block retour %}

{% endblock %}
```

1.2) Page "Réservé choix date " :

```
<form method="POST" action="{{url_for('reservation_chambre')}}">
    <input type="date" required name="debut" placeholder="aaaa-mm-jj" /> <label> Date de
    début</label><br>
    <input type="date" required name="fin" placeholder="aaaa-mm-jj" /> <label> Date de
    fin</label><br>
    <br> <br>
    <div class="buttons">
        <button class="button is-success is-outlined"> <input type="submit"
    value="Valider"></button>

    </div>
</form>
```

Dans cette partie j'ai créé des inputs pour que le client puisse choisir la date à laquelle il veut réserver sa chambre.

```
# Condition à respecter réservation
if (session['debut'] < session['dateDuJour']):
    flash('Votre date d\'arrivée est déjà passer !! Veuillez saisir une
    date supérieure ou égal à la date du jour')
    return render_template("choix_dates.html",
    hasError=error,connected=session['connexion'])
elif (session['fin'] <= session['debut']):
    flash('Votre date de départ est antérieure à votre date d\'arrivée !')
    return render_template("choix_dates.html",
    hasError=error,connected=session['connexion'])
else:
    return render_template("choix_chambre.html", hasError=error,
    result=chambre_libre(),connected=session['connexion'])
```

J'ai ajouté toutes les conditions nécessaires pour éviter les erreurs de réservations :

- Si la date de fin est inférieure à la date du début
- Si la date de début est inférieure à la date du début

S'il y a ces 2 conditions alors un message est envoyé à l'utilisateur pour lui dire qu'il y a une erreur dans le choix des dates de la réservation.

1.3) Page "Choix chambre ":

Une fois le choix de la date sélectionné l'utilisateur peut choisir une chambre qui est disponible :

date d'arrivée : 2021-03-28

date de départ : 2021-03-29

Voici les chambres disponible à cette date :

J'ai créé une fonction dans lequel, j'ai écrit une requête SQL permettant de sélectionner les chambres libres sur ma base de données :

CHAMBRE PRIVILEGE ▼

➔

```
def chambre_libre():
    return pgsq1_select2('select
description from hotel.chambre where
disponibilite=true')
```

Valider

```
@app.route("/choix_chambre", methods=['POST'])
def choix_chambre(error=None):

    session['list_chambre'] = request.form['list_chambre']
    if (session['list_chambre'] == "CHAMBRE PRIVILEGE"):
        session['idChambre']=1
    elif (session['list_chambre'] == "CHAMBRE INFLUENCE"):
        session['idChambre']=2
```

Si le client choisi chambre privilège alors une variable session (session idchambre) est créée et mis à « 1 » sinon si le client choisi « chambre influence » alors session idchambre vaut 1. La variable

session idchambre va être utilisée par la suite pour faire l'insertion de la chambre choisie par l'utilisateur dans ma base de données.

Lorsque le client a sélectionné une date valide la fonction est appelé dans le render template :

```
return render_template("choix_chambre.html", hasError=error,  
result=chambre_libre(),connected=session['connexion'])
```

1.4) Page "Validation de la réservation" :

Descriptif de la chambre

Numéro de la chambre : 1

Prix par nuit : 80 €

Autre chambre

Valider la chambre

```
@app.route("/after_display_chambre", methods=['POST'])  
def after_display_chambre(error=None):  
    session['display_chambre'] = request.form['display_chambre']  
  
    if (session['display_chambre'] == "Valider la chambre"):  
        pgsqL_ajout_chambre(session['idClient'], session['idChambre'], session['debut'], session['fin'])  
        return render_template("choix_client.html", hasError=error, rows=pgsqL_liste_mail(),connected=session['connexion'])  
    elif (session['display_chambre'] == "Autre chambre"):  
        return render_template("choix_chambre.html", hasError=error, result=chambre_libre(),connected=session['connexion'])
```

Le choix de la chambre a été fait par l'utilisateur il a donc le descriptif de la chambre sélectionné, le numéro de la chambre ainsi que le prix.

L'utilisateur à 2 choix :

- Il peut choisir une autre chambre qui permettra de retourner sur la page « choix chambre ».
- Il peut valider la chambre.

Si l'utilisateur choisi « valider la chambre » alors une requête va être effectuée pour faire un « insert » dans la base de données :

```
def pgsqL_ajout_chambre(idClient, numeroChambre, dateDebut, dateFin):  
    pgsqL_insert('insert into hotel.reservation values(DEFAULT, (%s),  
(%s),(%s), (%s), false, null )',[idClient, numeroChambre, dateDebut,  
dateFin])
```

Pour faire cette requête j'utilise les variables que j'ai auparavant récupérées dans des variables sessions :

- Id du client
- Numéro chambre choisi
- La date de début
- La date de fin

Cette fonction est donc appelée lorsque le client clique sur valider la chambre. J'ai utilisé la même opération que pour chambre libre pour appeler la fonction « pgsql_ajout_chambre ».

Ma base de données a bien enregistré la requête et il renvoie sur une page qui indique au client que la réservation a bien été effectuée.

Merci de votre commande
revenir à l'accueil

Le client peut cliquer sur « revenir à l'accueil » pour revenir à l'accueil.

Hôtel THE ATLANTIS

[Réserver une chambre](#)[Ajout d'une consommation](#)[Payer votre facture](#)[Faire un commentaire](#)[déconnecter](#)

connecté

Bienvenue sur le site de l'Hôtel Bis

Date du jour : 2021-03-27

bienvenu sur votre compte : Pinabel

1.5) Onglet « Ajout d'une consommation » :

Lorsque l'utilisateur clic sur ajout d'une consommation il atterrit sur cette page :

Choisissez votre consommation supplémentaire:

- ☐ Oasis au prix de : 2 euros
- ☐ Whisky au prix de : 4 euros
- ☐ Coca au prix de : 2 euros
- ☐ Orangina au prix de : 2 euros

Soumettre

Sur cette page est renseigné toutes les consommations supplémentaires que peut faire un client. Ils sont situés dans la base de données à la table « hotel.bar »

Lorsque l'utilisateur clic sur ajout d'une consommation une requête est effectuée dans ma base de données :

Voici la requête :

```
def pgsq1_choix_consommation():  
    return pgsq1_select2('select * from hotel.Bar')
```

Appel de la fonction:

```
if (session['accueil'] == "Ajout d'une consommation"):  
    return render_template("ajout_consommation.html", hasError=error,  
result=pgsq1_choix_consommation(), connected=session['connexion'])
```

Affichage du résultat de la requete en HTML :

```
{%if result %}  
    {% for leresultat in result %}  
        <div>  
            <input type="radio" value="{{leresultat[0]}}"  
name="boisson">{{leresultat[1]}} </input>  
            <label>au prix de : {{leresultat[2]}} euros</label>  
        </div>  
    {% endfor %}  
{%endif%}
```

Ici on récupère les éléments leresultat[0], leresultat[1], leresultat[2] de ma requête SQL qui correspond respectivement à idboisson, nomboisson et prix. Tant que l'affichage de result(la requête) est pas finis on fait une boucle jusqu'à ce que le tableau result soit finis.

Schéma explicatif :

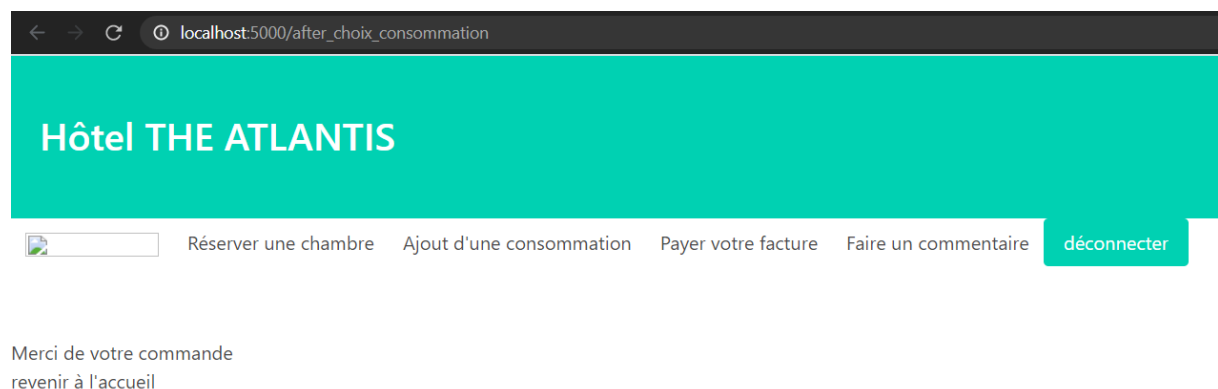


Le client peut alors choisir une consommation supplémentaire et cliquer sur soumettre. Lorsque le bouton est appuyé, une requête insert va alors se faire pour ajouter la consommation choisie du client. La requête utilise l’ID du client, le jour et la boisson choisi qui sont tous les 3 récupérer avec des variables en sessions.

```
def pgsq1_ajout_consommation(idClient, jour, boisson):  
    pgsq1_insert('insert into hotel.consommation  
values(Default,(%s),(%s),(%s), false);', [idClient, jour, boisson])
```

Lorsqu’il y a un ajout d’une consommation comme la réservation, le statu est « A réglé » (False) il faudra se rendre dans l’onglet « Payer votre facture » pour la régler.

Un message indique au client que sa commande a bien été effectuée :



1.6) Onglet « Payer votre facture » :

Produit	Prix	Réglé
Whisky	4	False
Whisky	4	True
Orangina	2	True

le total des consommations déjà payées s'élève à : 6 €

le total des consommations non payées s'élève à : 4 €

Numéro de la chambre	Prix	nuitée	prix du séjour	Réglé
1	80	1	80	False
1	80	1	80	True
1	80	2	160	True
1	80	1	80	True
1	80	5	400	True

le total des chambres déjà payées s'élève à : 720 €

le total des chambres non payées s'élève à : 80 €

On y retrouve un tableau concernant les consommations supplémentaires qu'a effectuées le client. On sait s'il a réglé ou pas ces consommations. En dessous il y a 2 phrases qui expliquent à l'utilisateur du compte ce qu'il a payé et ce qui lui reste à payer.

Ces 2 éléments sont aussi des requêtes effectuées à la base de données :

Pour afficher le tableau des consommations :

```
def pgsqldb_facture_bar(idClient):  
    return pgsqldb_select('select hotel.Bar.prix_bar, hotel.Bar.produit,  
hotel.consommation.regler from hotel.Bar , hotel.consommation where  
hotel.Bar.id_boisson = hotel.consommation.id_boisson and  
hotel.consommation.idClient = (%s) ;', [idClient])
```

Pour afficher les consommations qui lui reste à payer :

```
def pgsqldb_facture_bar_false(idClient):  
    return pgsqldb_select('select sum(hotel.Bar.prix_bar) as tt_non_payé from  
hotel.Bar , hotel.consommation where hotel.Bar.id_boisson =  
hotel.consommation.id_boisson and hotel.consommation.idClient = (%s) and  
hotel.consommation.regler = false', [idClient])
```

Pour afficher les consommations qu'il a payé :

```
def pgsql_facture_bar_true(idClient):  
    return pgsql_select('select sum(hotel.Bar.prix_bar) as tt_non_payé from  
hotel.Bar , hotel.consomption where hotel.Bar.id_boisson =  
hotel.consomption.id_boisson and hotel.consomption.idClient = (%s) and  
hotel.consomption.regler = true',[idClient])
```

Chaque une des fonctions est appelée lorsque l'onglet « payer votre facture » est cliqué :

```
elif (session['accueil'] == "Payer votre facture"):  
    return render_template("paiement_facture.html", rowsbar=pgsql_facture_bar(session['idClient']),connected=session['connexion'],  
rowschambre=pgsql_facture_chambre(session['idClient']),  
rowsbartrue=pgsql_facture_bar_true(session['idClient']),  
rowsbarfalse=pgsql_facture_bar_false(session['idClient']),  
rowschambretrue=pgsql_facture_chambre_true(session['idClient']),  
rowschambrefalse=pgsql_facture_chambre_false(session['idClient']))
```

J'ai effectué la même opération pour afficher le tableau récapitulatif des nuits d'hôtel réservé. (voir code en pièce jointe).

Pour que le client paye sa facture il doit simplement cliquer sur le bouton en bas « payer ». Les règlements de chaque tableau passent false à true donc le client n'a plus rien à payer.

Produit	Prix	Réglé
Whisky	4	True
Whisky	4	True
Orangina	2	True

le total des consommations déjà payées s'élève à : 10 €

le total des consommations non payées s'élève à : None €

Numéro de la chambre	Prix	nuitée	prix du séjour	Réglé
1	80	1	80	True
1	80	1	80	True
1	80	2	160	True
1	80	1	80	True
1	80	5	400	True

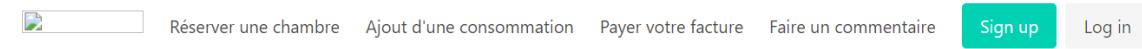
le total des chambres déjà payées s'élève à : 800 €

le total des chambres non payées s'élève à : None €

paye

1.7) Onglet « Sign Up » :

Pour faire une réservation le client doit être authentifié pour associer la réservation et les consommations au client associé. Si le client n'a pas de compte en créer un en cliquant sur Sign Up :



Bienvenue sur le site de l'Hôtel Bis
Date du jour : 2021-03-27

Page Sign_up :

Votre Nom

Votre prénom
Votre Adresse
Ville
Code postal
E-mail
Mot de passe

[S'inscrire!](#)

[Vous avez déjà un compte? Connectez-vous maintenant!](#)

L'utilisateur devra renseigner ces informations. L'adresse mail ainsi que le mot de passe seront utilisés pour authentifier le client. Chaque champ est enregistré avec des inputs puis récupérer dans des variables en « sessions » ce qui m'a permis de les réutiliser facilement dans mes requêtes SQL.

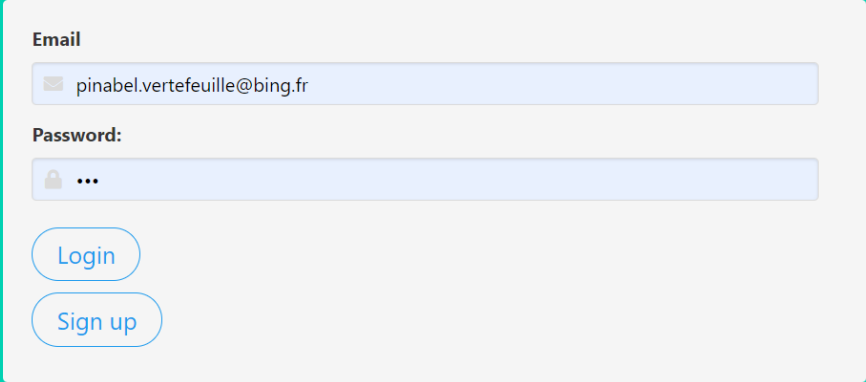
```
@app.route("/after_choix_client", methods=['POST'])
def after_choix_client(error=None):
    session['newnom'] = request.form['newnom'].strip().capitalize()
    session['newprenom'] = request.form['newprenom'].strip().capitalize()
    session['newemail'] = request.form['newmail'].strip().lower()
    session['password'] = request.form['password'].strip()
```

La requête ci-dessous utilise chaque une des variables sessions créées ci-dessus. Elle me permet d'inscrire un nouvel utilisateur sur ma base de données. Cette requête est appelée lorsque le client appuie sur le bouton s'enregistrer.

```
def pgsq1_ajout_client(newnom, newprenom, newadresse, newville,
newCode_postal,newmail,password):
    pgsq1_insert('insert into Hotel.client
values(DEFAULT, (%s), (%s), (%s), (%s), (%s), (%s), (%s));', [newnom, newprenom,
newadresse, newville, newCode_postal,newmail,password])
```

1.8) Onglet « Login » :

L'utilisateur a aussi la possibilité de se login en se connectant soit par l'intermédiaire du bouton « Vous avez déjà un compte ? Connectez-vous maintenant ! » ou avec le menu en cliquant sur login.

A login form titled "Login" is displayed on a teal background. The form is a light gray rectangle containing two input fields. The first field is labeled "Email" and contains the text "pinabel.vertefeuille@bing.fr". The second field is labeled "Password:" and contains three dots, indicating a password. Below the password field are two buttons: "Login" and "Sign up", both with rounded corners and blue outlines.

Comme dit auparavant l'utilisateur doit se connecter avec son adresse mail et son mot de passe.

Requête qui permet de vérifier si l'utilisateur a saisi un bon mail et mots de passe.

```
def login(mail,password):
    return pgsq1_select("select * from hotel.client where mail = (%s) and
password = (%s) ;", [mail,password])
```

Si l'utilisateur a saisi les bonnes valeurs pour s'authentifier (`if(str(login(session['mail'], session['password'])) != "[]")` alors il est envoyé dans index.html sinon il retombe sur login.html.

```

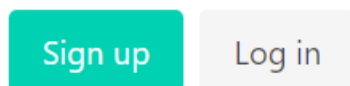
        if(str(login(session['mail'], session['password'])) != "[]" ):
            session['connexion'] =1
            return render_template("index.html", hasError=error,
rows=login(session['mail'],
session['password']),connected=session['connexion'])
        else:
            return render_template("login.html", hasError=error,
rows=login(session['mail'],
session['password']),connected=session['connexion'])
            return render_template("login.html", hasError=error,

```

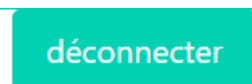
La variable session['connexion'] permet de dire que l'utilisateur est bien authentifié. Pour réserver une chambre l'utilisateur doit forcément être authentifié alors j'ai ajouté une condition dans mon menu : Si session['connexion'] = 1 alors l'utilisateur peut accéder cliquer sur le menu et être redirigé sur les bonnes pages sinon si l'utilisateur clique sur un des boutons du menu d'en être authentifié donc session['connexion'] = 0 alors l'utilisateur est redirigé vers la page de login.

Cela permet à l'utilisateur d'être forcément authentifié pour réserver une chambre ou bien ajouter une consommation.

Lorsque l'utilisateur n'est pas connecté il y a les boutons :

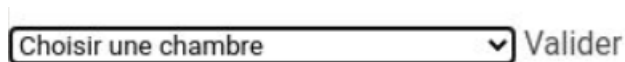


Et lorsque l'utilisateur est connecté donc session['connexion'] = 1 il y a un bouton se déconnecter et les boutons sign up et login sont retirés :



1.9) Onglet « Faire un commentaire » :

Pour faire un commentaire le client doit choisir dans un premier temps la chambre à laquelle il veut faire un commentaire et appuyer sur valider :



Une fois valider le client peut écrire un commentaire puis cliqué sur envoyer le commentaire :



Pour ce faire je me connect à la base « fescots » de mongodb :

```

def get_mg_db():
    db = MongoClient("mongodb://mongodb.emi.u-bordeaux.fr:27017").fescots
    return db

```

Puis j'insère le commentaire avec ma fonction :

```
def set_commentaire(com):  
    mgdb = get_mg_db()  
    result = mgdb.hotelcollection.insert(  
        {  
            "commentaire": com,  
            "numchambre": session['list_chambre'],  
            "nomclient": login(session['mail'], session['password'])[0][1]  
        }  
    )  
    return result
```

Lorsque le client clique sur ajouter un commentaire un fichier JSON est créé qui est appelé hotelcollection dans lequel on va retrouver les champs sélectionnés/ inscrits par le client :

Le commentaire, le numéro de la chambre et le nom client que je récupère avec la fonction login écrite auparavant.

Les commentaires des chambres seront affichés lorsque le client réserve une chambre :

Descriptif de la chambre

Numéro de la chambre : 1

Prix par nuit : 80 €

Autre chambre

Valider la chambre

Chambre : CHAMBRE PRIVILEGE

test

Par Gauthier

Chambre : CHAMBRE PRIVILEGE

Mon premier commentaire! ;)

Par Gauthier

Les commentaires sont affichés avec la fonction get_commentaire() qui permet de lister tous les commentaires selon la chambre choisie par l'utilisateur.

```
def get_commentaire():  
    mgdb = get_mg_db()  
    return mgdb.hotelcollection.find( { "numchambre": session['list_chambre']
```

Chambre : 1
Mon premier commentaire! ;)
Par Gauthier

Pour effectuer le CSS de mon site flask j'ai utilisé les Templates de « BULMA » et je les ai adapté pour mon site hôtel.



II – Les problèmes rencontrés

Le plus gros problème que j'ai rencontré durant ce projet est le fonctionnement de Django. N'ayant pas énormément de connaissance en web j'ai eu du mal à afficher les variables de retour de mes requête SQL. Cela dit une fois le mécanisme compris j'ai pu avancer mon projet assez vite car le mécanisme d'affichage des données est assez similaire entre les fonctions.

Autre problème que j'ai rencontré, c'est la mise en place de mon projet flask sur l'hébergement de l'université. Mon projet ne voulait pas s'afficher avec l'URL :

<https://florent-escots.emi.u-bordeaux.fr/flask/>

Après re vérification de tous mes fichiers je me suis rendu compte que le problème venait d'un import de librairie que j'avais oublié. J'ai fini par réussir à afficher le projet correctement.

III – Amélioration de la base de données

Dans ma base de données créé initialement au premier semestre je n'ai pas ajouté beaucoup de champs :

Dans la table hotel.réserveation j'ai ajouté un champ réglé en Boolean :

	numero_reservation [PK] entier	idclient entier	numero_chambre entier	date_debut Date	date_fin Date	regler booléen
1	3	3	1	10/12/2017	15/12/2017	faux
2	4	3	1	21/03/2021	26/03/2021	faux
3	6	3	1	2021-03-28	03/04/2021	faux

Et dans la table hotel.consommation le champs régler :

	id_consommation [PK] integer	idclient integer	jour_consommation date	id_boisson integer	regler boolean
1	1	1	2021-03-20	1	true
2	2	1	2021-03-20	2	true
3	3	1	2021-03-20	1	true

Les 2 champs que j'ai ajouté mon permis de différencier le règlement entre une chambre et une consommation. Dans mon site ces champs mon servi pour effectuer un détail de la facture.

III – Amélioration futur :

Une des améliorations que je pourrai effectuer est de faire en sorte que l'utilisateur puisse retirer sa réservation faite. Pour se faire il suffira simplement d'ajouter une requête SQL qui indique de supprimer la chambre réservée par l'utilisateur, cette requête utilisera l'id du l'utilisateur ainsi que la chambre choisie.

Une autre amélioration serait s'ajouter sur ma base de données une section « administrateur » en Boolean ce qui me permettrait d'ajouter des fonctionnalités supplémentaires qui seront accessibles uniquement par un administrateur par exemple le fait d'initialiser la base de données ou de la supprimer. J'ai créé une fonction init_db et drop db que je n'ai pas eu le temps d'implanté sur mon site web :

```
def pgsql_drop_db():
    db = pgsql_connect()
    cursor = db.cursor()
    try:
        with app.open_resource('hotel_destruction.sql') as f:
```

```

        cursor.execute(f.read().decode('utf8'))
        db.commit()
    except Exception as e :
        flash('Désolé, service indisponible actuellement.')
        flash(str(e))
        return redirect(url_for('hello', error=str(e)))

def pgsqldb_init_db():
    db = pgsqldb_connect()
    cursor = db.cursor()
    try:
        with app.open_resource('hotel_creation.sql') as f:
            cursor.execute(f.read().decode('utf8'))
        db.commit()
        with app.open_resource('hotel_function.sql') as f:
            cursor.execute(f.read().decode('utf8'))
        db.commit()
        with app.open_resource('hotel_contrainte.sql') as f:
            cursor.execute(f.read().decode('utf8'))
        db.commit()
        with app.open_resource('hotel_view.sql') as f:
            cursor.execute(f.read().decode('utf8'))
        db.commit()
        with app.open_resource('hotel_insertion.sql') as f:
            cursor.execute(f.read().decode('utf8'))
        db.commit()
    except Exception as e :
        flash('Désolé, service indisponible actuellement.')
        flash(str(e))
        return redirect(url_for('hello', error=str(e)))

```