MonAvenir, vaste projet !

Objectif : créer un mini Parcoursup

application web destinée à recueillir et gérer les vœux d'affectation de lycéens du Rhône pour un établissement de l'enseignement supérieur dans le Rhône.

par Frédéric Junier, Brigitte Mougeot, Véronique Reynaud

Liste des fichiers créés dans l'ordre d'apparition/d'utilisation

creer_base_monavenir.sql
lycee-rhone-data.csv
superieur-rhone-data.csv
generer_base_csv.py
dossier csv contenant : lycee-rhone.csv, superieur-rhone.csv, admin.csv et 154 fichiers élèves selon les lycées…
peupler_base_bd.py
monavenir.bd
main.py
dossier templates contenant :
(pour l'étape 4)
accueil1.html accueil2.html rechercheFormation.html admin.html lycee.html superieur.html
(pour l'étape 6 – rôle élève)
accueil.html candidature.html compte.html eleve.html erreur.html rechercheFormation.html reponses.html resultatRecherche.html
(pour l'étape 6 – rôle administrateur)
accueil.html admin.html erreur.html rechercheFormation.html resultatRecherche.html resultatStatsEtabAvant.html resultatStatsGeneralAvant.html statsEtabAvant.html statsGeneralAvant.html
static / stylesheets / style_monavenir.css

FINI OU PAS ??? est-ce que ça a été vérifié….

Liste des fichiers donnés aux élèves dans l'archive materiel.zip

generer_base_csv_TODO.py
peupler_base_bd_TODO.py
main_exo1.py
main_exo2.py
dossier templates contenant : accueil1.html accueil2.html rechercheFormation.html
cadeau.py
dossier templates contenant : (rôle élève) accueil.html candidature.html compte.html eleve.html erreur.html rechercheFormation.html reponses.html resultatRecherche.html
dossier templates contenant: (rôle admin) accueil.html admin.html rechercheFormation.html resultatRecherche.html resultatStatsEtabAvant.html resultatStatsGeneralAvant.html statsEtabAvant.html erreur.html statsGeneralAvant.html
static / stylesheets / style_monavenir.css

Comment utiliser notre application

- Exécuter ***main.py***, ce qui met en route notre serveur web en attente sur le port 8000.
- Sur la même machine que le serveur, taper dans un navigateur l'URL *localhost :8000*.
- La page d'accueil de l'application est renvoyée.
- Un profil de test a été créé pour chaque type d'utilisateur :

* en tant qu'admin :	id	eleve	mdp	monavenir
* en tant qu'admin :	id	admin	mdp	monavenir
* en tant qu'établissement du supérieur :	id	superieur	mdp	monavenir
* en tant que lycée :	id	lycee	mdp	monavenir

Les élèves créeront un répertoire MonAvenir et commenceront par extraire tous les fichiers de l'archive materiel.zip dans ce répertoire.

Ou alors, ils peuvent aussi extraire au fur et à mesure des besoins…

MonAvenir, Sommaire

Pagination à finaliser et tps impartis

Atelier 1 : En utilisant la base suivante, écrire les requêtes SQL correspondant aux différents exercices.
Difficulté croissante (les dernières ne sont pas au programme officiellement). Peut servir d'évaluation.

admin(idAdmin, login, password, nom, prenom)
superieur(idSuperieur, login, password, nom, type, commune, lattitude, longitude, nbAdmis, nbAppel, coefNote1, coefNote2)
lycee(idLycee, login, password, nom, commune)
eleve(idEleve, #idLycee, login, password, nom, prenom, anneeNaissance, note1, note2)
candidature(#idEleve, #idSupérieur, statut)

Exercice1 : Donner tous les noms des lycées
```
SELECT nom FROM lycee ;                                    #On en obtient 154…
```

Exercice2 : Donner les noms des écoles du supérieurs dans l'ordre alphabétique décroissant
```
SELECT nom FROM superieur ORDER BY nom DESC ;               #On en obtient 181…
```

Exercice3 : Donner les 10 premiers noms, prénoms des élèves ayant obtenu au moins 19 en note1
```
SELECT nom,prenom FROM eleve WHERE note1 >= 19 LIMIT 10 ;   # Pichon Édith, Rocher Lorraine…
```

Exercice4 : Donner les prénoms et moyennes des élèves dont le prénom contient la lettre A et dont la moyenne des notes 1 et 2, nommée moyenne, est strictement supérieure à 16
```
SELECT prenom, (note1 + note2)/2 AS moyenne FROM eleve      #Astrid, Chantal, Antoinette, …
WHERE 16 <= moyenne AND prenom LIKE '%A%';                 #On en obtient 542…
```

Exercice5 : Donner le nombre d'élèves du lycee ' Lycée polyvalent Aiguerande'
```
SELECT COUNT(*) FROM eleve JOIN lycee USING(idLycee)
WHERE lycee.nom = 'Lycée polyvalent Aiguerande' ;          #On en obtient 60…
```

Exercice6 : Donner le nombre d'élèves du lycée ' Lycée général et technologique Jean-Paul Sartre' qui ont postulé pour l'IUT Lumière #On en obtient 4…
```
SELECT COUNT (*) FROM eleve JOIN lycee USING(IdLycee)
JOIN candidature USING(idEleve) JOIN superieur USING(Idsuperieur)
WHERE lycee.nom = 'Lycée général et technologique Jean-Paul Sartre' AND superieur.nom = 'IUT Lumière ';
```

Exercice7 : Donner les noms des filles prénommées 'Charlotte' qui sont dans le même lycée que 'Charlotte Turpin' (sauf elle-même !) *auto jointure…*
```
SELECT E2.nom                                              #On en obtient 2 : Renault, Traore
FROM eleve E1 JOIN eleve E2 ON E1.idLycee =  E2.idLycee
WHERE E1.nom = 'Turpin' AND E1.prenom = 'Charlotte' AND E2.nom <> 'Turpin' AND E2.prenom = 'Charlotte' ;
```

Exercice8 : Donner, pour chaque école du supérieur, son nom et le nombre de postulants *GROUP BY*
```
SELECT nom, COUNT(*)                                       #Institut des droits de l'ho, 583
FROM superieur join candidature using(idSuperieur) GROUP BY idSuperieur  ;  #Institut env. tecno, 538 …
```

Exercice9 : Donner, pour chaque école du supérieur, son nom et le nombre de postulants du lycée du Parc ayant plus de 16 de moyenne *GROUP BY + HAVING*
```
SELECT superieur.nom, COUNT(*)          #24 enregistrements  Institut des sciences de la famille 4
FROM candidature JOIN eleve USING(idEleve) JOIN lycee USING(idLycee) JOIN superieur USING(idSuperieur)
WHERE lycee.nom = 'Lycée général du Parc' GROUP BY idSuperieur HAVING ((note1+note2)/2 >= 16) ;
```

Exercice10 : Donner, pour chaque école du supérieur, son identifiant et le nombre de postulants du lycée d'identifiant 78 en écrivant 0 si aucun élève n'a postulé # Vérifier 97 : 0 *LEFT JOIN*
```
SELECT idSuperieur, COUNT(R. idEleve) AS 'Nombre Demandes' FROM candidature C LEFT JOIN (SELECT *
FROM eleve E WHERE idLycee = 78) R ON C.idEleve= R.idEleve GROUP BY idSuperieur ORDER BY idSuperieur ;
```

Atelier 2 : Requêtes SQL de création de la BD MonAvenir

Ecrire avec un éditeur de texte comme NotePad++ un fichier nommé **creer_base_monavenir.sql** permettant la construction de cette BD.
Enregistrer ce fichier dans le dossier MonAvenir.

Atelier 3 : Quelles fonctionnalités attendre de notre application ? papier crayon puis DB BROWSER

L'idée ici n'est pas seulement de savoir comment formuler une requête mais de savoir de quelles requêtes on va avoir besoin pour le projet...

Consigne aux élèves :
- Travailler en groupes à partir de la base de données *MonAvenir*
- Chaque groupe d'élèves prend un rôle parmi : administrateur, lycéen, chef d'établissement du supérieur, proviseur d'un établissement du secondaire.
- Selon le rôle endossé, quelles requêtes voudriez-vous pouvoir effectuer sur la base de données une fois que les vœux des élèves ont été formulés ?
- Ecrire ces requêtes en langage SQL.
- Les tester et corriger si besoin en utilisant la BD *MonAvenir* et le logiciel DB BROWSER.
- Enregistrer vos requêtes dans un fichier **requêtes_administrateur.sql**, resp **requêtes_lycéen.sql**, resp **requêtes_superieur.sql**, resp resp **requêtes_ proviseur sql** selon les cas.

30 à 60 minutes

Atelier 1 : Travail sur des données open data et sur le format CSV
Génération des données sur les lycées et les établissements du supérieur

Lors de cet atelier, nous serons amenés à échanger avec les élèves sur :
- les données open-data
- les enjeux éthiques du stockage de données nominatives sur les élèves (d'où des données fictives)
- les normes d'encodage du texte (plus de problème d'encodage lorsqu'on ouvre avec NotePad…)

Les différents fichiers créés sont à enregistrer dans le dossier MonAvenir.

1/ Récupération des données lycées

Sur internet, saisir l'adresse : https://data.education.gouv.fr, puis cliquer sur l'onglet Données.
Effectuer des filtrages pour obtenir la liste des **155 lycées ouverts du Rhône**.
Exporter le jeu des 155 données au format CSV sous le nom **lycee-rhone-data.csv**
Ouvrir le fichier avec un tableur et garder uniquement les colonnes **Appellation officielle** et **Commune**.
Le Lycée en ligne 75 va ouvrir mais n'a pas encore de nom : supprimer cette ligne, donc 154 lycées en tout.

2/ Récupération des données supérieurs

Aller sur le site internet https://www.data.gouv.fr/en/datasets/etablissements-denseignement-superieur-2/# et exporter la liste des établissements d'enseignement supérieur au format csv et renommer sous **superieur-rhone-data.csv**
Ouvrir le fichier avec un tableur et supprimer les colonnes inutiles : garder uniquement **type d'établissement**, **nom**, **commune, département, latitude** et **longitude**. Avec des tris/filtres, garder uniquement les 181 écoles du Rhône, puis supprimer la colonne département et inverser l'ordre des deux premières colonnes en nommant : **nom**, **type**, **commune, latitude** et **longitude.**

MonAvenir
- lycee-rhone-data.csv
- superieur-rhone-data.csv
- generer_base_csv_TODO.py
- csv (dossier à créer)
  - lycee-rhone.csv
  - …

3/ Modification des données lycées et supérieurs, travail sur python

Ouvir le fichier **generer_base_csv_TODO.py .**
Dans ce fichier, on utilise le module Faker (qu'il faudra importer si nécessaire) afin de créer des pseudos ou mots de passes fictifs.
Ce fichier contient 3 fonctions : la première generer_login permet de générer un login en vérifiant que celui-ci n'est pas déjà pris ; les 2 fonctions suivantes permettent de compléter les fichiers CSV pour obtenir les fichiers **lycee-rhone.csv** et **superieur-rhone.csv** qui seront enregistrer dans une dossier csv à créer !
Compléter la fonction **generer_login** du fichier **generer_base_csv_TODO.py**,
puis en vous inspirant de la fonction **generer_csv_lycee**, écrire la fonction **generer_csv_superieur**.

Atelier 2 : Génération des données sur les élèves et leurs candidatures

1/ Générer les élèves

- Ecrire une fonction **anneeNaiss(a)** qui simule les années de naissances des élèves de terminales.

Elle prend en argument une année a et renvoie une année entre a et a +4 selon une certaine distribution correspondant aux pourcentages d'élèves ayant des années de retard ou d'avance…
Si a = 2000, la fonction doit renvoyer un nombre entier entre 2000 et 2004 compris, sachant qu'il y a environ 1% d'élèves qui ont 2 ans de retard (nés en 2000), 7% d'élèves qui ont 1 an de retard (nés en 2001), 8% d'élèves qui ont 1 an d'avance (nés en 2003) et 1% d'élèves qui ont 2 ans d'avance (nés en 2004).
En utilisant le fichier **generer_base_csv_TODO.py :**
- Compléter la fonction **generer_login** qui génère un login en vérifiant que celui-ci n'est pas déjà pris.
- Compléter la fonction **generer_csv_eleve** afin de créer des fichiers csv de 60 élèves pour chacun des lycées.

2/ Générer les candidatures

- Compléter la fonction **generer_csv_candidature**. Pour simplifier : toutes les écoles admettront 100 candidats et prendront 200 candidats en liste d'appel, mais on pourra modifier par la suite…

POUR FINIR : lancer le fichier python. Combien de fichiers contient le sous dossier csv. On trouve 156

On devrait trouver 157 !!!
Le problème vient du fait qu'il y a deux lycées professionnels privés Notre Dame, l'un à Givors, l'autre à Villefranche sur Saône ! Et que lorsque l'on crée les fichiers csv, on les nomme à partir de leur nom !

Pour y remédier : reprendre le fichier **lycee-rhone-data.csv**, et modifier les appellations suivantes :

Lycée professionnel privé Notre Dame G
Lycée professionnel privé Notre Dame V

Puis relancer le fichier python **generer_base_csv_TODO.py** et vérifier qu'il y a bien maintenant 157 fichiers dans le sous dossier csv.

Intérêt des bases de données propres….

MonAvenir Etape 3 : Création du formulaire inscription et du fichier main

30 minutes

Dans cette partie, il est intéressant de remarquer les différentes façons pour créer une base de données :

- directement à partir de DB Browser ou Sqlite
- à partir d'un fichier texte
- à partir d'un fichier CSV
- à la volée dans python…

MonAvenir
- creer_base_monavenir.sql
- peupler_base_bd_TODO.py
- dossier csv

Atelier 1 : Peuplement à partir d'un fichier .sql (NotePad++)

Compléter la fonction python **creer_base_monavenir** du fichier **peupler_base_bd_TODO.py**.
Elle prend en argument le fichier **monavenir.bd** à créer et le fichier **creer_base_monavenir.sql** à partir duquel on va créer la base de données. C'est celui qui a été réalisé dans l'atelier 2 de l'étape1.

Atelier 2 : Peuplement à partir d'un fichier .csv

ici on peut faire 3 groupes

En vous inspirant de la fonction **peupler_eleve**, compléter les fonctions python **peupler_lycee. peupler_superieur, peupler_admin,** du fichier **peupler_base_bd_TODO.py** en insérant les données lues, cette fois, dans les fichiers csv correspondants.
Attention les 3 fonctions sont plus simples et ne nécessitent pas l'utilisation d'un compteur…

Atelier 3 : Peuplement dynamique à partir d'un script de python

Compléter la fonction python **peupler_candidature.**
Puis regarder et commenter chaque ligne de la fonction **peupler_base_mon_avenir.**

En particulier, les élèves devront comprendre seul l'insertion d'un élève test...

POUR FINIR :
Lancer le fichier python **peupler_base_bd_TODO.py**.

Aller consulter la base de données créée avec le logiciel DB Browser.
Puis à la main, modifier les données de la base de sorte qu'il n'y ait aucun élève du Lycée polyvalent Charlie Chaplin qui demande EPITA Lyon.

```
SELECT E.nom, E.idEleve, L.nom, S.nom, S.idSuperieur FROM eleve AS E JOIN lycee AS L USING
(idLycee) JOIN candidature USING(idEleve) JOIN superieur AS S USING(idSuperieur)
WHERE S.nom = 'EPITA Lyon' AND L.nom = 'Lycée polyvalent Charlie Chaplin';
```

	nom	idEleve	nom	nom	idSuperieur
1	Perrot	5284	Lycée polyvalent Charlie Chaplin	EPITA Lyon	97

Développement Web côté serveur en Python 120 minutes

En raison du niveau très hétérogène de nos élèves et des conséquences liées à une fin d'année scolaire mouvementée, nous avons choisi de revoir une partie du programme de l'an dernier sur l'interaction client/serveur.
Le premier exercice pourrait être facultatif pour certains élèves.
Quelques liens hypertextes dans cette *couleur* et l'écriture rouge correspond aux trous élèves.

Exercice 1 : découverte de *Flask*

Flask est un micro *Framework* permettant de développer des applications Web en *Python*. Il impose peu de choix prédéfinis au programmeur.
En appui ou en complément, on pourra utiliser les ressources en ligne suivantes :

- une activité de David Roche autour de *Flask* construite pour des élèves de première NSI
- la documentation officielle *https://flask.palletsprojects.com/en/1.1.x/*

MonAvenir
- main_exo1.py
- cadeau.py
- monavenir.db
- templates
  - accueil1.html
  - accueil2.html
  - rechercheFormation.html

1. Vérifier que vous avez bien tous les fichiers nécessaires ci-contre (certains étaient à extraire depuis l'archive **materiel.zip**).

2. Éditer le fichier **main_exo1.py** dans un environnement de programmation en *Python* tel que *Thonny* ou *Spyder*.

3. Exécuter ce script *Python*. Dans la console, on devrait obtenir un affichage d'une dizaine de lignes.

⋆ **Question a) Compléter :** L'application *Flask*, portant le nom du script, a lancé un serveur Web dont l'adresse [IP] est 127.0.0.1 (boucle locale) et le port *TCP* 8000.

⋆ **Question b)** Quel est le protocole de la couche application qui est utilisé lors d'un échange entre un client et un serveur Web ? http
Où sont situés le serveur et le client si on veut tester l'application ? sur la même machine

⋆ **Question c)** D'après le code de l'application, si on ouvre un navigateur Web comme Firefox et qu'on saisit l'*URL* http://127.0.0.1:8000/ dans la barre d'adresse, quel affichage devrait-on obtenir ? Vérifier.
Bonjour il est … heures … minutes et ... secondes. *Mis à jour avec les bonnes valeurs !*

⋆ **Question d)** Attendre quelques instants et rafraîchir la page. Que remarque-t-on ? mise à jour
Comment peut-on qualifier ce type de page Web ? page dynamique (vs statique)

4. La fonction accueil est préfixée par l'instruction @app.route('/'), qui est un *décorateur*.
Si l'*URL* se termine par /, la fonction accueil est appelée et retourne un code *HTML* après l'avoir formaté avec des paramètres de temps en heures, minutes et secondes. On parle de *contrôleur de route* (view dans la terminologie de flask). Néanmoins, il serait préférable de retourner une page Web complète en respectant la structure d'un document en *HTML*.
Flask propose une fonction *render_template* qui permet de retourner une page *HTML* complète.
Le terme template signifie que la page peut être paramétrée, comme nous le verrons plus tard.
Attention : tous les fichiers *HTML*, doivent se trouver dans le dossier **templates** du répertoire **materiel**.

⋆ **Question a)** Dans la console, arrêter le serveur avec la séquence clavier CTRL + C. Revenir sur la page html et rafraichir. Que se passe-t-il ? Firefox ne peut établir de connexion avec le serveur à l'adresse 127.0.0.1:8000

⋆ **Question b)** Dans le fichier **main_exo1.py**, remplacer "accueil1.html" par "acc.html" enregistrer le fichier python sans relancer le serveur. Rafraichir à nouveau la page html. Il ne se passe rien, ça fonctionne encore. Fermer puis relancer le serveur. Rafraichir.
Quel message d'erreur s'affiche ? Internal server error
Dans la barre d'outils de développement du navigateur (F12-réseau), récupérer le code d'erreur *HTTP*.
Quelle différence avec la célèbre erreur 404 ? 404 not found, coté client ici erreur 500 coté serveur.
Rectifier le nom du template puis, sans arrêter le serveur, rafraîchir la page html.
La modification est-elle prise en compte ? non , c'est tout l'intérêt du mode déboggage qui suit !!

⋆ **Question c)** Arrêter le serveur et activer le mode débogagge avec le paramètre d'exécution debug = True dans la ligne `app.run(debug=True, host='127.0.0.1', port=8000)`.
Lancer le serveur, et rafraichir encore la page web. Puis reprendre le début de la question b). Est-il désormais nécessaire de relancer le serveur pour que les modifications soient prises en compte ? non
Remarque : En phase de développement, nous activerons toujours le mode débogagge.

5. **Compléter :** render_template("accueil1.html", heure = h, minute = m, seconde = s) est un appel de fonction avec deux types de paramètres :
   - positionnel : le nom du fichier "accueil1.html"
   - nommés : heure = h, minute = m, seconde = s . Ainsi, render_template affiche une page web en page web en utilisant les valeurs des variables données en paramètre.

6. Consulter avec un éditeur de textes, comme *Notepad++*, le code du fichier **accueil1.html**.

⋆ **Question a)** Obtient-on le même affichage qu'avec l'application, si on ouvre **accueil1.html** directement avec un navigateur Web ? non ! On obtient un affichage avec de accolades : balises non remplacées !

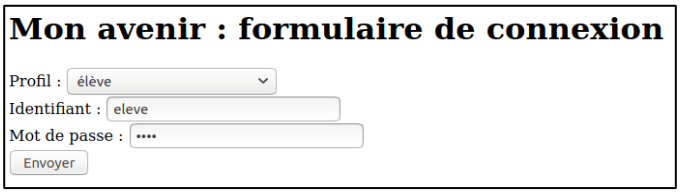
⋆ **Question b)** Quelle syntaxe particulière permet d'insérer les valeurs de ces paramètres dans le fichier **accueil1.html** ? les doubles accolades
Ce mécanisme est effectué par un moteur de template nommé *Jinja*, qui permet d'insérer des structures de contrôle comme des tests ou des boucles pour paramétrer plus finement l'affichage du *template*.

Exercice 2 : formulaire de connexion et base de données

L'objectif est de réaliser un formulaire de connexion dans la page d'accueil avec identifiant (`login`) et mot de passe (`password`). Le programme de traitement du formulaire doit vérifier l'existence du couple (`login`, `password`) dans la base de données **monavenir.db** fournie dans le dossier `materiel`.

Cahier des charges du formulaire html :

- Le formulaire aura trois champs :
  - un champ de sélection de profil :
    élève, lycée, admin, établissement du supérieur
  - un champ de saisie de `login`
  - un champ de saisie de `password`



- Le formulaire aura un bouton cliquable qui envoie les données au serveur. L'*URL* de la requête se termine par "/connexion", ce qui déclenche l'appel de la fonction contrôleur de route 'connexion'. Cette fonction devra interroger la base de données et selon le succès de la requête retourner une page web d'erreur ou une page `eleve.html`, `lycee.html`, `superieur.html`, `admin.html` selon le profil.

1. Ouvrir le fichier **accueil2.html** qui se trouve dans le dossier **templates**.

⋆ **Question a)** Quelle est la méthode d'envoi de ce formulaire ? POST Quelle autre méthode aurait pu être choisie ? GET Quelles sont les différences entre les deux ? affichage dans URL, limité en taille…

⋆ **Question b)** Quelle est l'*URL* complet d'envoi du formulaire ? 127.0.0.1/connexion

⋆ **Question c)** Quel est l'intérêt du type password ? cacher lettres entrées

⋆ **Question d)** Quels sont les différents types de widgets utilisés dans ce code ? Label, Select, Button

⋆ **Question e)** Les attributs id, name et for des éléments du formulaire ont toujours le même nom. Est-ce nécessaire ? A quoi servent ces attributs ?
https://developer.mozilla.org/fr/docs/Web/Guide/HTML/Formulaires/Les_blocs_de_formulaires_natifs
Id est un identifiant de balise pour servir de cible à du CSS, du JS ou lien hypertexte dans une page html
Name est le nom de paramètre qui sera transmise dans le formulaire.
Ils ne sont pas toujours égaux mais c'est pratique.
For permet de rattacher le label à une balise input à condition de porter le même nom que l'id et permet un meilleur confort d'utilisation.

2. Reprendre le programme **main1.py** et l'enregistrer sous un nouveau nom : **main2.py** dans le même répertoire. Puis modifier la fonction accueil pour qu'elle retourne **accueil2.html** avec le formulaire.

3. Compléter et ajouter la fonction connexion ci-dessous pour traiter les données du formulaire.

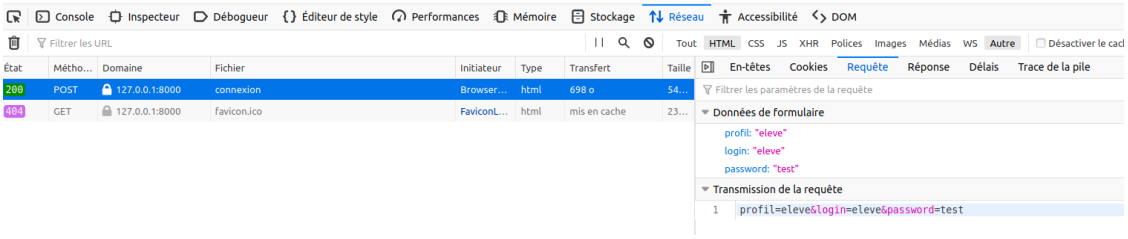
```
#dispatcheur de route / URL
@app.route('/connexion',methods = ['POST'])
def connexion():
  "Controleur de la route '/connexion' "
  if request.method == 'POST':
    #les valeurs des paramètres sont dans le dictionnaire request.form
    result = request.form
    profil = result['profil']                    #récupération de la valeur du paramètre profil
    login = result['login']                      #récupération de la valeur du paramètre login
    password = result['password']                #récupération de la valeur du paramètre password
    conn = sqlite3.connect('monavenir.db')       #connexion à la base de données
    conn.row_factory =  sqlite3.Row              #pour récupérer les lignes sous forme de dictionnaire
    cur = conn.cursor()                          #création d'un curseur pour parcourir la base
    #soumission d'une requête SQL avec paramètres pour... ici ils doivent completer le but
    cur.execute("SELECT * FROM {profil} WHERE login=? and password=? ;".format(profil),(login, password))
    user = cur.fetchone()      #récupération de la ligne de resultat
    cur.close()                #fermeture du curseur
    conn.close()               #fermeture de la connexion
    if user:
        return render_template("{}.html".format(profil))
# on ouvre un serveur en local sur le port 8000
app.run(debug = True, host='127.0.0.1', port=8000)
```

Remarque : L'accès à la base de données s'effectue en trois temps.

- Connexion et création d'un curseur qui est l'objet permettant de lire ou d'écrire dans la base.
- Interrogation ou modification de la base avec une requête en *SQL* formatée à l'aide de paramètres :
  `cur.execute("SELECT * FROM {profil} WHERE login=? and password=? ;".format(profil),(login, password))`
- Fermeture du curseur puis de la base.

Le nom de la table est inséré dans la requête avec un formatage de chaîne de caractères en Python.
L'insertion de valeurs dans la condition du WHERE suit une syntaxe particulière : les ? seront remplacés dans l'ordre par les valeurs du tuple de paramètres (login, password). Il s'agit d'un mécanisme de sécurité conte l'injection de code *SQL* malveillant à la place des valeurs attendues. Voir le site *https://bobby-tables.com/*.

⋆ **Question a)** Tester une connexion élève dont le couple (login,password) = (eleve,test).
Quelle est la page *HTML* retournée par le contrôleur de route `connexion` ? page d'accueil de l'élève
Où sont stockées les données du formulaire dans le code Python ci-dessus ?
dans request.form stocké dans la variable resul (dictionnaire) on accède aux données avec result['profil']...
Faire apparaître ces données avec les outils de développement du navigateur Web : F12
Reseau / requête / Données de formulaire..., transmission...



⋆ **Question b)** Créer dans le dossier **templates**, des fichiers **admin.html**, **lycee.html** et **superieur.html** qui seront retournés par la fonction connexion pour les autres profils possibles. Tester que tout fonctionne avec des utilisateurs sélectionnés directement dans la base avec *sqlitebrowser*.

Exercice 3 : formulaire de recherche

Une fois la connexion réalisée, on souhaite qu'un utilisateur de profil élève puisse interroger la base. Par exemple pour rechercher les établissements du supérieur par type (10 types différents) et/ou par commune …

1. Ajouter au programme **main2.py**, le contenu du fichier **cadeau.py**.

2. La fonction rechercheFormation retourne un formulaire de recherche de formation selon deux critères pour la route "/rechercheFormation" saisie après l'*URL* du serveur. Consulter son code.

   ⋆ **Question a)** Pourquoi n'a-t-on pas écrit @app.route('/ rechercheFormation ',methods = ['POST']) ?
   c'est pas un formulaire…

   ⋆ **Question b)** Traduire en français les deux requêtes *SQL* exécutées par la fonction.
   Selection des différents types d'établissements du supérieur
   Selection des différentes communes d'établissements du supérieur

   ⋆ **Question c)** Ouvrir le fichier **rechercheFormation.html** du dossier **templates** avec un éditeur de texte. Tester le formulaire avec l'élève test.
   Comment sont générées les listes d'option par le moteur de template *Jinja* ? avec la boucle {% for ….%}

3. Compléter la fonction resultatRecherche qui doit traiter les données du formulaire envoyé depuis rechercheFormation.html. Certains blocs commentés en #TO DO, doivent être complétés avec l'exécution par le curseur des requêtes *SQL* appropriées.

4. En s'inspirant de rechercheFormation.html, compléter les TO DO / à compléter dans le *template* resultatRecherche.html dans le dossier templates qui pourra être rempli avec les valeurs des variables liste_sup et result calculées par la fonction resultatRecherche.

5. On veut désormais créer une fonction **contrôleur de route** interface qui redirige vers la page d'accueil du profil eleve.html, lycee.html etc. depuis n'importe quelle page du site.

   Remarque : Il faut donc utiliser un mécanisme de mémorisation du profil lors de la navigation dans le site. *A l'origine, le protocole HTTP imaginé par Tim Berners-Lee était « sans état » : chaque requête était indépendante, sans possibilité pour le serveur de lier deux requêtes successives venant du même système et donc de garder en mémoire des informations sur un utilisateur. En 1994, pour favoriser le développement du commerce en ligne, des ingénieurs de Netscape proposent un mécanisme d'échange d'information au format texte, un **cookie** stocké chez le client ou le serveur. L'article https://linc.cnil.fr/fr/une-petite-histoire-du-cookie raconte l'histoire des **cookies**. Dans notre cas, nous allons utiliser un **cookie de session** stocké sur le serveur, la **session** étant l'interaction client/serveur entre la connexion et la déconnexion. Lors de la déconnexion, il faut penser à effacer le **cookie de session**.*

   Répondre aux questions et compléter le code du fichier **main_exo2.py** au fur et à mesure :

   ⋆ **Question a)** Après la définition de l'application, il faut définir une clef secrète de session pour chiffrer le **cookie de session**, normalement il faut utiliser une clef aléatoire.
   Quelle est la clé choisie dans le code ci-dessous ? "clef secrète"

```
#création d'une instance de l'application
app = Flask(__name__)
#clef de session
app.secret_key = "clef secrète"
```

⋆ **Question b)** Dans la fonction connexion, enrichir le bloc final de if user avec le peuplement du **cookie de session** qui dans *Flask* est un objet avec la même interface qu'un dictionnaire. On peut remarquer qu'il n'est plus forcément nécessaire de transmettre au *template* le dictionnaire user : il faut penser à remplacer user par session['user'] dans le *template* eleve.html :

```
if user:
  #dictionnaire de session
  session['user'] = dict(user)  #les objets de type ROW retournés ne sont pas sérialisables et stockables dans le dictionnaire du cookie de session
  session['profil'] = profil    #on stocke le profil dans le cookie de session
  return render_template("{}.html".format(profil))
```

⋆ **Question c)** Ajouter la fonction **contrôleur de route** interface en complétant le code proposé :

```
#dispatcheur de route / URL
@app.route('/interface')
def interface():
  "Controleur de la route '/interface' "
  if 'profil' in session and session['profil']:
    "à compléter avec un return render_template(...)" #TO DO
```

⋆ **Question d)** Enfin, compléter avec la fonction **contrôleur de route** deconnexion :

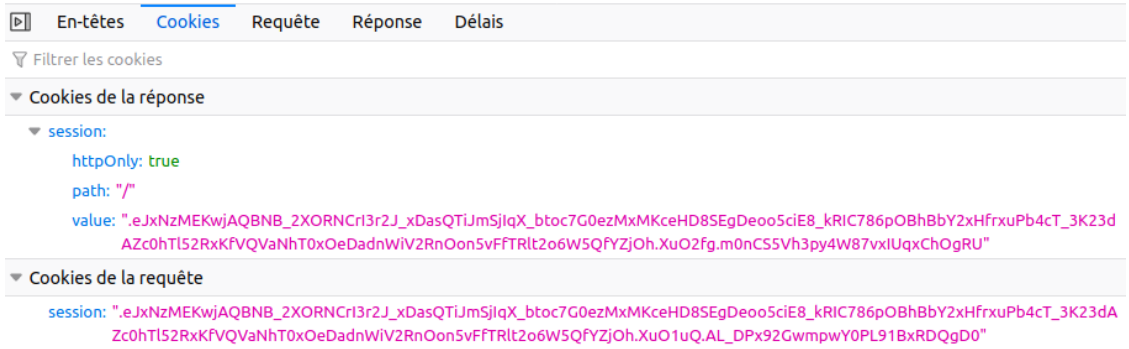
```
#dispatcheur de route / URL
@app.route('/deconnexion')
def deconnexion():
  #on vide le dictionnaire de session
  print(session)    #debug
  session.clear()   #on vide le dictionnaire de session
  print(session)    #debug
  #redirection vers la route controlée par la fonction accueil
  #return render_template('/')
  return redirect(url_for('accueil'))
```

6. ⋆ **Question a)** Ouvrir un navigateur et la fenêtre des outils de développement.

⋆ **Question b)** Réaliser une session complète avec l'élève test : connexion, recherche de formation, retour à l'interface du profil, déconnexion.

⋆ **Question c)** Afficher lors de chaque chargement de page, les **cookies** contenus dans les requêtes et réponses *HTTP* : pour quelles pages a-t-on un **cookie** dans la requête et dans la réponse ? juste dans la requête ? Observer l'évolution de la valeur du **cookie** de requête.

⋆ **Question d)** Interpréter les valeurs affichés par les deux instructions print lors de l'exécution de la fonction. permet de vérifier que le cookie de cession a bien été vidé.



Non développé. En prolongement. Le code est fourni.

MonAvenir Etape 6: Création des formulaires de consultations

30 à 60 minutes

TP noté par groupes : Le travail préliminaire a donné tous les outils nécessaires aux élèves pour réaliser cette tache finale et donc le travail sera rendu pour **évaluation**.

Travail sur formulaire consultation HTML et fichier main
Chaque groupe se voit attribuer un rôle parmi : élève, lycée, établissement du supérieur, administrateur.
Nous détaillons ci-dessous les consignes données pour les rôles élève et administrateur.

Avec le rôle élève

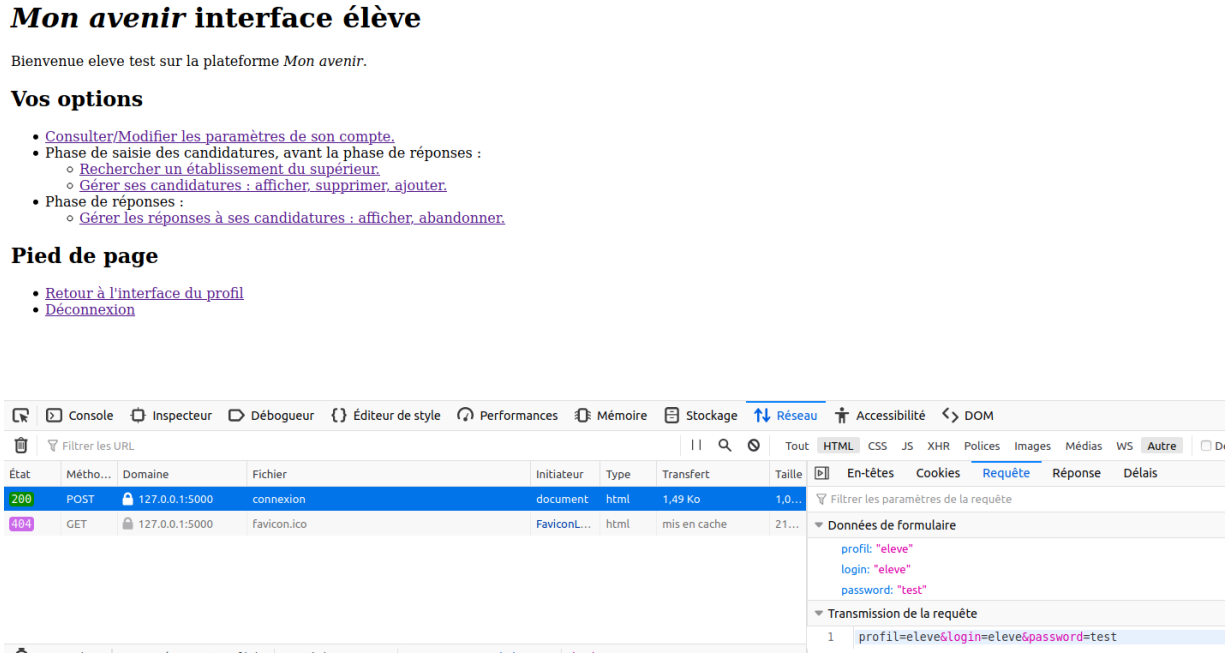
Vérifier que vous avez bien tous les fichiers nécessaires ci-contre (certains étaient à extraire depuis l'archive **materiel.zip**).

```
MonAvenir
  main.py
  monavenir.db
  static
    stylesheets
      style_monavenir.css
  templates
    accueil.html
    candidature.html
    compte.html
    eleve.html
    erreur.html
    rechercheFormation.html
    reponses.html
    resultatRecherche.html
```

Pour chaque fonctionnalité demandée dans les ateliers, il faudra écrire :

- un formulaire *HTML* dans le dossier *templates*
- une fonction **contrôleur de route** dans le script Python `main.py` en vous aidant des activités réalisées à l'étape 4
- une page *HTML* retournée par la fonction **contrôleur de route** et placée dans le répertoire `templates`.

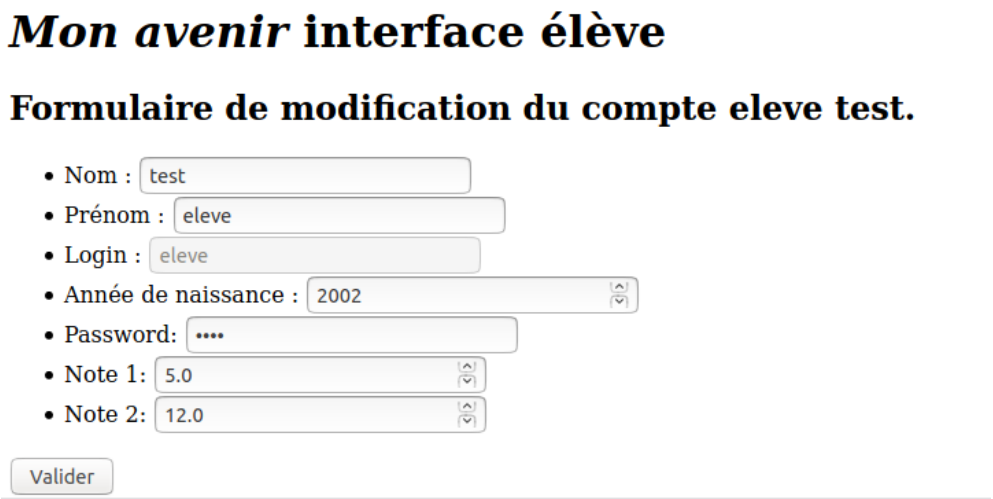
Lancer l'application web en exécutant le fichier `main.py` et se connecter avec le profil `élève` et le compte d'identifiants `(login, password) = (eleve, test)`. On arrive alors sur l'interface d'accueil du profil.



Rôle élève - Atelier 1 :

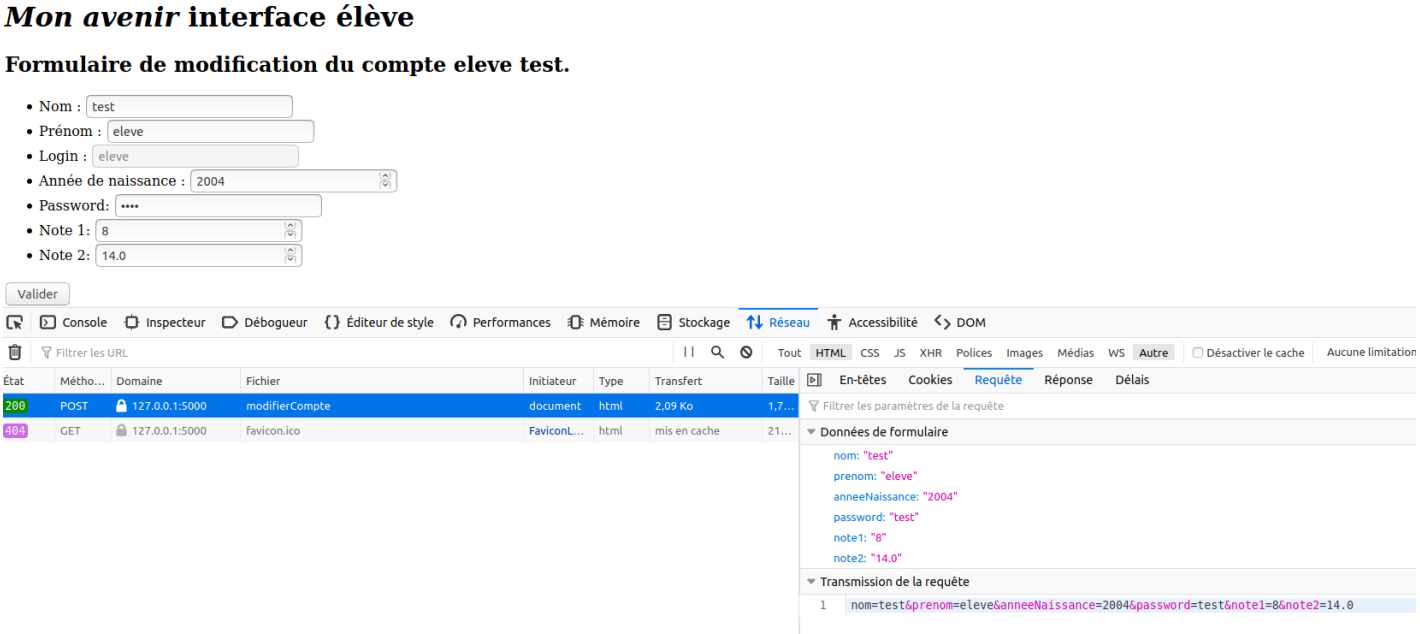
1. Ecrire un compte-rendu du travail effectué dans un fichier Libre-Office `compte-rendu-atelier1.odt` à la racine du répertoire `materiel`.

2. Compléter le formulaire dans le fichier `compte.html` du répertoire `templates` afin qu'il s'affiche comme ci-dessous lorsqu'on clique sur le lien `Consulter/Modifier les paramètres de son compte` de l'interface élève. Les paramètres du formulaire sont les attributs récupérés lors de la connexion dans `session['user']` : `login`, `anneeNaissance`, `note1`, `note2`.



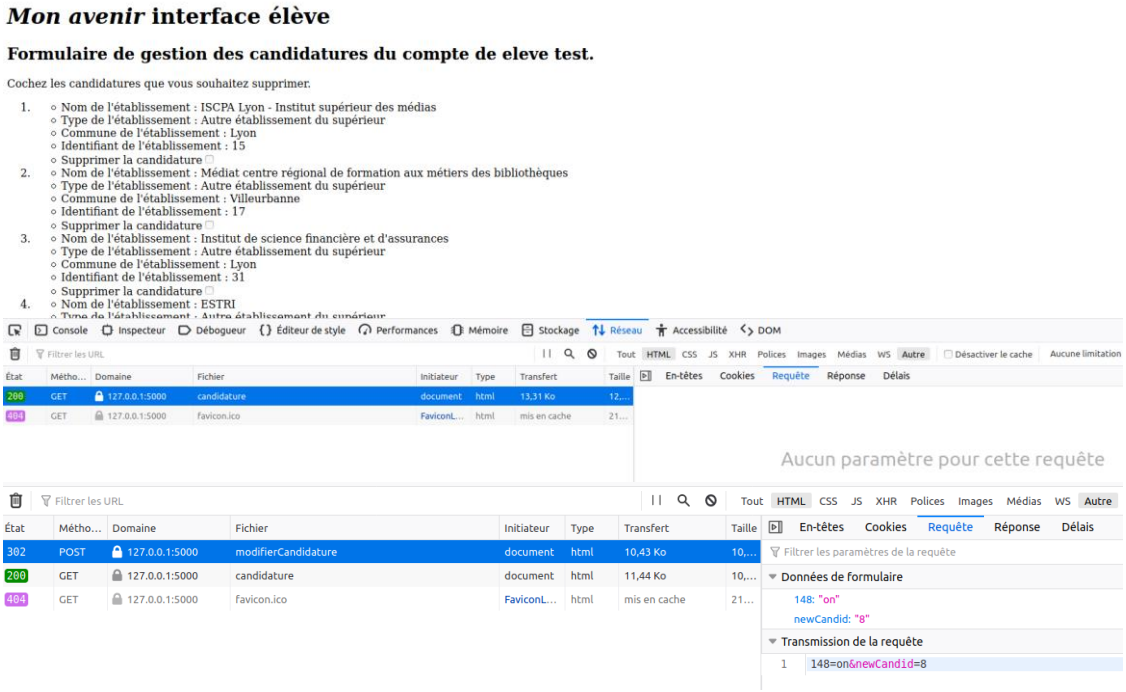

```
<form action="/modifierCompte" method="post">
            <ul>
                <li><label for='nom'>Nom : </label> <input type='text' id='nom' n
ame='nom' value="{{session['user']['nom']}}"></li>
                <li><label for='prenom'>Prénom : </label> <input type='text' id='
prenom' name='prenom' value="{{session['user']['prenom']}}"></li>
                <!--   TO DO  : à compléter                    -->
            </ul>

            <input type="submit" value="Valider"/>
</form>
```

3. Compléter la fonction `requeteMajCompte` dans le **contrôleur de route** `modifierCompte` du script `main.py` afin que le clic sur le bouton `Envoyer` du formulaire dans `compte.html` permette la mise à jour des paramètres du compte.



4. Compléter la fonction `requeteListeCandidature` dans le **contrôleur de route** `candidature` de `main.py` afin qu'elle détermine la liste des candidatures de l'élève.
Cette liste est nécessaire à l'affichage du formulaire de gestion des candidatures avant la phase de réponses lorsqu'on clique sur `Gérer ses candidatures : afficher, supprimer, ajouter.` dans l'interface élève.

5. Compléter ensuite le fichier `candidature.html` du répertoire `templates` afin que le formulaire s'affiche comme ci-dessous lorsqu'on clique sur le lien `Gérer ses candidatures : afficher, supprimer, ajouter.` de l'interface élève. La première image montre le haut du formulaire avec la liste des candidatures. On supprime une candidature en la cochant. La seconde image montre le bas avec la possibilité de saisir une nouvelle candidature à l'aide de son identifiant récupéré par le formulaire de recherche de formation `Rechercher un établissement du supérieur`.




```
<form action="/modifierCandidature" method="post">
            <ol>
                {% for candidature in session['liste_candidature'] %}
                <li>
                    <ul>
                    <li> Nom de l'établissement : {{candidature['nom']}} </li>
                    <li> Type de l'établissement : {{candidature['type']}}  </li>
                    <!---  TO DO à compléter  -->
                </ul>
                </li>
                {% endfor %}
                <li><label for="newCandid">Nouvelle candidature</label>  <input type="num
ber" name="newCandid" id="newCandid"></li>
            </ol>
            <input type="submit" value="Valider"/>
</form>
```

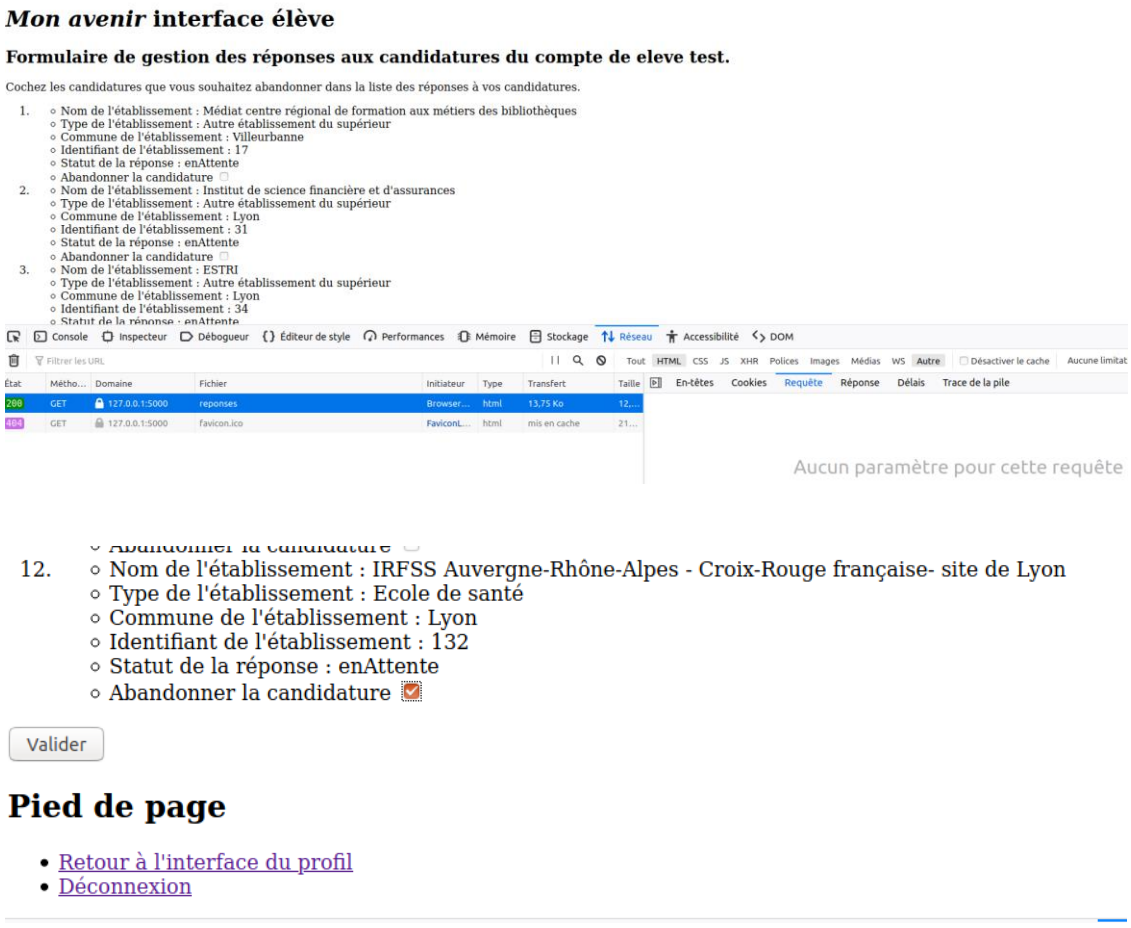
6. Compléter enfin les fonctions `requeteListeCandidatureApresSuppression` et `requeteTraitementNouvelleCandidature` dans le **contrôleur de route** `modifierCandidature` du script `main.py` afin que le clic sur le bouton `Envoyer` du formulaire dans `candidature.html` permette la mise à jour des candidatures. Ecrire les requêtes SQL pour mettre à jour la BD ?

Rôle élève - Atelier 2 :

1. Ecrire un compte-rendu du travail effectué dans un fichier Libre-Office `compte-rendu-atelier2.odt` à la racine du répertoire `materiel`.

2. Compléter la fonction `requeteListeReponsesCandidatures` dans le **contrôleur de route** `reponses` de `main.py` afin qu'elle détermine la liste des candidatures de l'élève. Cette liste est nécessaire à l'affichage du formulaire de gestion des candidatures avant la phase de réponses lorsqu'on clique sur `Gérer les réponses à ses candidatures : afficher, abandonner.` dans l'interface élève.

3. Compléter ensuite le fichier `reponses.html` du répertoire `templates` afin que le formulaire s'affiche comme ci-dessous lorsqu'on clique sur le lien `Gérer les réponses à ses candidatures : afficher, abandonner.` de l'interface élève. La première image montre le haut du formulaire avec la liste des candidatures. On supprime une candidature en la cochant. La seconde image montre le bas avec le bouton d'envoi du formulaire.



4. Compléter enfin les fonctions `requeteAbandonCandidature` et `requeteMajListeAppel` dans le **contrôleur de route** `modifierReponses` du script `main.py` afin que le clic sur le bouton `Envoyer` du formulaire dans `reponses.html` permette la mise à jour des réponses. Préciser dans `compte-rendu.odt` comment les valeurs `idSuperieur`, `idEleve` et `statut` sont transmises par le formulaire.

Avec le rôle administrateur

Vérifier que vous avez bien tous les fichiers nécessaires ci-contre (certains étaient à extraire depuis l'archive **materiel.zip**).

```
MonAvenir
  main.py
  monavenir.db
  static
    stylesheets
      style_monavenir.css
  templates
    accueil.html
    admin.html
    erreur.html
    rechercheFormation.html
    resultatRecherche.html
    resultatStatsEtabAvant.html
    resultatStatsGeneralAvant.html
    statsEtabAvant.html
    statsGeneralAvant.html
```

Pour chaque fonctionnalité demandée dans les ateliers, il faudra écrire :

- un formulaire *HTML* dans le dossier *templates*
- une fonction **contrôleur de route** dans le script Python `main.py` en vous aidant des activités réalisées à l'étape 4
- une page *HTML* retournée par la fonction **contrôleur de route** et placée dans le répertoire `templates`.

Lancer l'application web en exécutant le fichier `main.py` et se connecter avec le profil `admin` et le compte d'identifiants `(login, password) = (admin, monavenir)`. On arrive sur l'interface d'accueil du profil.

Rôle administrateur - Atelier 1 :

1. Ecrire un compte-rendu du travail effectué dans un fichier Libre-Office `compte-rendu-atelier1.odt` à la racine du répertoire `materiel`.

2. Compléter le formulaire dans le fichier `statsEtabAvant.html` du répertoire `templates` afin qu'il s'affiche comme ci-dessous lorsqu'on clique sur le lien `Formulaire pour générer des statistiques par établissement du supérieur.` de l'interface `admin`.

```
<form action="/resultatStatsEtabAvant" method="post">
        <ul>
             <li> <a href="/rechercheFormation">Recherche d'un établissement</a></li>
             <li><label for='idSuperieur'>Identifiant de l'établissement : </label>
                 <input type='text' id='idSuperieur' name='idSuperieur' required></li>
             <li><label for="total">Nombre total de candidatures pour l'établissement
                 </label><input type="checkbox" id="total" name="total" unchecked></li>
                  <!--  TO DO , à compléter       -->
        </ul>
        <input type="submit" value="Valider"/>
</form>
```

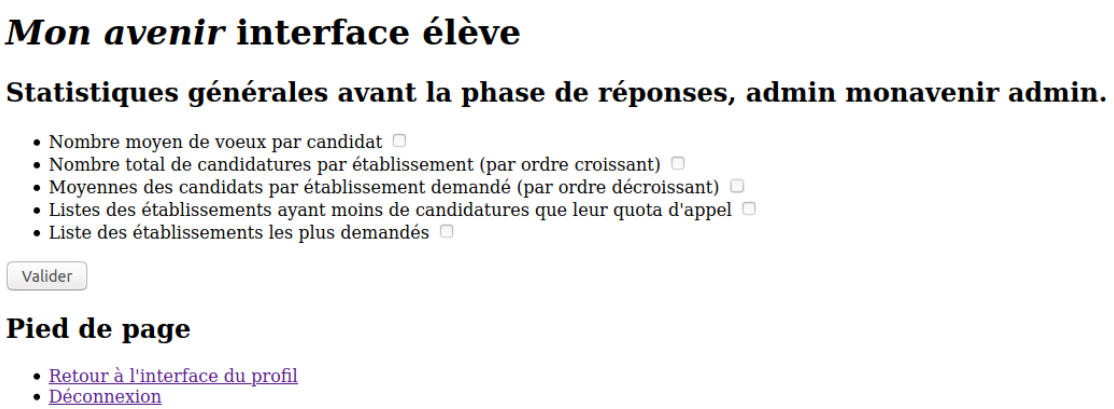
3. Compléter les cinq fonctions de requêtes dans le **contrôleur de route** `resultatStatsEtabAvant` du script `main.py` afin que le clic sur le bouton `Envoyer` du formulaire dans `statsEtabAvant.html` permette l'affichage des statistiques sélectionnées dans le formulaire.

4. Que se passe-t-il si on clique sur les liens `Classer les candidatures pour chaque établissement et démarrer la phase de réponses.`, `Formulaire pour générer des statistiques par établissement du supérieur.` ou `Formulaire pour générer des statistiques globales.` ?
   Inspecter le fichier `main.py` pour expliquer ces comportements.
   Que pourriez-vous faire pour compléter cette application Web ?
   Répondre dans le fichier `compte-rendu.odt`.

Rôle administrateur - Atelier 2 :

1. Ecrire un compte-rendu du travail effectué dans un fichier Libre-Office `compte-rendu-atelier2.odt` à la racine du répertoire `materiel`.

2. Compléter le formulaire dans le fichier `statsGeneralAvant.html` du répertoire `templates` afin qu'il s'affiche comme ci-dessous lorsqu'on clique sur le lien `Formulaire pour générer des statistiques globales.` de l'interface `admin`.

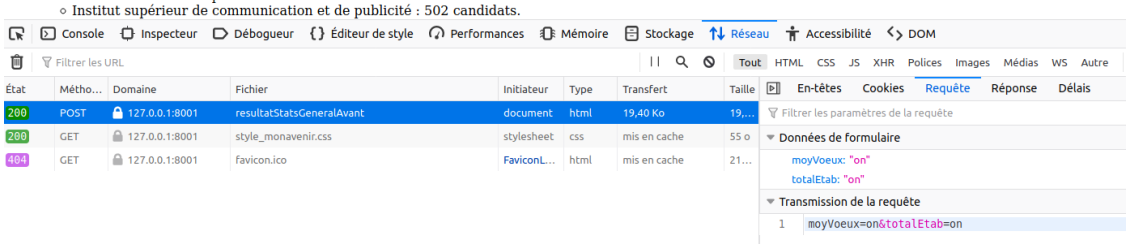



```
<form action="/resultatStatsGeneralAvant" method="post">
       <ul>
          <li><label for="moyVoeux">Nombre moyen de voeux par candidat </label>
              <input type="checkbox" id="moyVoeux" name="moyVoeux"  unchecked>
          </li>  <!--  TO DO , à compléter
          <li><label for="maxCandid">Liste des établissements les plus demandés </label>
              <input type="checkbox" id="maxCandid" name="maxCandid"  unchecked>
          </li>
       </ul>
          <input type="submit" value="Valider"/>
</form>
```

***Mon avenir* interface admin**

Statistiques générales avant la phase de réponses, admin monavenir admin.

- **Nombre moyen de voeux par candidat : 10**
- **Nombre de total de voeux par établissement du supérieur :**
  - Institut formation et recherche sur les organisations sanitaires : 457 candidats.
  - ISEG - Digital Marketing and Communication School : 484 candidats.
  - Institut de formation d'aides-soignant de l'hôpital Nord-Ouest Tarare : 487 candidats.
  - Institut de formation d'aides-soignants de l'Argentière : 488 candidats.
  - Faculté des sciences et technologies - Département informatique : 488 candidats.
  - Faculté d'odontologie : 494 candidats.
  - Faculté de droit : 497 candidats.
  - Ecole de psychologues praticiens : 498 candidats.
  - Ecole du Personnel Paramédical du service de santé des Armées : 499 candidats.
  - Ecole Emile Cohl : 499 candidats.
  - Ecole supérieure de biologie-biochimie-biotechnologies : 499 candidats.
  - Institut supérieur de gestion : 500 candidats.
  - SUP'de Com école supérieure de communication : 501 candidats.
  - Institut supérieur de communication et de publicité : 502 candidats.



3. Compléter les cinq fonctions de requêtes dans le **contrôleur de route** `resultatStatsEtabAvant` du script `main.py` afin que le clic sur le bouton `Envoyer` du formulaire dans `statsEtabAvant.html` permette l'affichage des statistiques sélectionnées dans le formulaire.

4. Que se passe-t-il si on clique sur les liens `Classer les candidatures pour chaque établissement et démarrer la phase de réponses.`, `Formulaire pour générer des statistiques par établissement du supérieur.` ou `Formulaire pour générer des statistiques globales.` ?
   Inspecter le fichier `main.py` pour expliquer ces comportements.
   Que pourriez-vous faire pour compléter cette application Web ?
   Répondre dans le fichier `compte-rendu.odt`.

Avec le rôle de chef d'établissement du supérieur

Dans cet atelier, les élèves vont travailler sur une base dont les candidatures ont été classées.
Une proposition d'énoncé, le matériel élève et les corrigés sont accessibles depuis *https://gitlab.com/frederic-junier/foad-bloc4-mougeot-reynaud-junier/-/tree/master/docs_eleves/etape6/ateliers_superieur*

*MonAvenir*Etape 7: : CSS…

En D.M. On demande aux élèves de réfléchir à l'UX Design… et de prévoir un CSS pour le projet.
Ce travail peut encore être mené par petit groupe et fera l'objet d'un bonus.