|  |  |  |
| --- | --- | --- |
| **Logo_Lycée** | **« Todo List »**  | logo%20ac%20orl%E9ans%20toursDescription : Description : Description : pemDescription : Description : Description : pem |





**SUJET DU TP**

**Mots-clés** : page Web dynamique, base de données, SGBD, SQLite,

Requêtes SQL, Python, micro Framework Bottle.

**PROBLÉMATIQUE**

Comment extraire des données à l’aide d’un SGBD, les mettre à jour et les présenter dans une page Web ?

**CONDITIONS DE DÉROULEMENT DE L’ACTIVITÉ**

|  |  |  |
| --- | --- | --- |
| **Phases de travail** | **Objectifs** | **Activités** |
| ***1) Mise en situation*** | ***Comprendre*** *le cahier des charges du site à réaliser.* | ***Lecture*** *de la mise en situation.* |
| ***2) Étude de*** ***la problématique*** | ***Présenter*** *des données issues d’une base SQLite dans une page Web et les* ***mettre à jour****.* | ***Interroger*** *et mettre à jour une base de données SQLite avec des requêtes SQL intégrées à du Python.****Créer*** *un serveur Web avec un micro Framework Python.****Créer*** *des modèles Web.* |
| ***3) Synthèse*** |

**SIGLES UTILISÉS**

|  |  |  |  |
| --- | --- | --- | --- |
| **Remarque** | Avertissement Important | Informations Information | DocumentÉditeur **VSCode** |

**PRÉPARATION DU DOSSIER DE TRAVAIL**

|  |  |
| --- | --- |
|  | Organisez votre dossier de travail en suivant les indications données en **annexe** à la fin de ce document. |

**RESSOURCES DOCUMENTAIRES, LOGICIELS ET MATÉRIELS**

PC, IDE VSCode avec les extensions SQLite et Python, micro Framework Python Bottle, répertoire du projet TODOLIST.

**WEBOGRAPHIE**

Présentation du TP <https://bit.ly/3fjM2HK>

Le module Python sqlite3 <https://bit.ly/3pwHouj>

Le micro Framework Bottle <https://bottlepy.org/>

Table des matières

[1 Mise en situation 3](#_Toc96709342)

[1.1 Site Web dynamique ? 3](#_Toc96709343)

[1.2 Cahier des charges du site « Ma liste des choses à faire » 3](#_Toc96709344)

[2 Étude de la problématique 4](#_Toc96709345)

[2.1 La base de données « Todo » 4](#_Toc96709346)

[2.2 Interagir en Python avec une base de données 4](#_Toc96709347)

[2.2.1 Création d’une connexion 4](#_Toc96709348)

[2.2.2 Exécution des requêtes 5](#_Toc96709349)

[2.2.3 Déconnexion 6](#_Toc96709350)

[2.3 Création d’un serveur Web avec le micro Framework Bottle 6](#_Toc96709351)

[2.3.1 Prise en main 6](#_Toc96709352)

[2.4 Affichage des données issues de la base « Todo » dans une page Web 7](#_Toc96709353)

[2.4.1 Solution 1 : pages Web dans le code Python 7](#_Toc96709354)

[2.4.2 Solution 2 : modèle Web (template) 7](#_Toc96709355)

[2.5 Mise à jour de la base « Todo » avec des formulaires 8](#_Toc96709356)

[2.5.1 Suppression d’éléments 8](#_Toc96709357)

[2.5.2 Ajout d’éléments 9](#_Toc96709358)

[2.6 Ajout d’une feuille de style 9](#_Toc96709359)

[3 Synthèse : idée de projet 10](#_Toc96709360)

**Table des illustrations**

[**Figure 1** : code source pour un site Web 3](#_Toc130831139)

[**Figure 2** : modèle client-serveur 3](#_Toc130831140)

[**Figure 3** : exemple de page "Ma liste des choses à faire" 3](file:///F%3A%5CPedago%5C2_Enseign_Gene_SI_NSI%5C20a_NSI%5CTNSI%5CC_Bases%20de%20donn%C3%A9es%5C2_TP%20_SQL%5CTP2%20SQL_Python_Web%5C1_Sujet%5CTNSI_TP_BDD_Langage_SQL_SQLite_Python_Bottle.docx#_Toc130831141)

[**Figure 4** : modèle client-serveur du projet 3](#_Toc130831142)

[**Figure 5 :** dossier du projet "TodoList" 4](file:///F%3A%5CPedago%5C2_Enseign_Gene_SI_NSI%5C20a_NSI%5CTNSI%5CC_Bases%20de%20donn%C3%A9es%5C2_TP%20_SQL%5CTP2%20SQL_Python_Web%5C1_Sujet%5CTNSI_TP_BDD_Langage_SQL_SQLite_Python_Bottle.docx#_Toc130831143)

[**Figure 6 :** SQLite Explorer dans VSCode 4](#_Toc130831144)

[**Figure 7 :** table "todo" initiale 4](#_Toc130831145)

[**Figure 8** : résultat attendu dans la fenêtre interactive 4](file:///F%3A%5CPedago%5C2_Enseign_Gene_SI_NSI%5C20a_NSI%5CTNSI%5CC_Bases%20de%20donn%C3%A9es%5C2_TP%20_SQL%5CTP2%20SQL_Python_Web%5C1_Sujet%5CTNSI_TP_BDD_Langage_SQL_SQLite_Python_Bottle.docx#_Toc130831146)

[**Figure 9 :** liste de courses dans la fenêtre interactive 5](file:///F%3A%5CPedago%5C2_Enseign_Gene_SI_NSI%5C20a_NSI%5CTNSI%5CC_Bases%20de%20donn%C3%A9es%5C2_TP%20_SQL%5CTP2%20SQL_Python_Web%5C1_Sujet%5CTNSI_TP_BDD_Langage_SQL_SQLite_Python_Bottle.docx#_Toc130831147)

[**Figure 10 :** accueil 6](#_Toc130831148)

[**Figure 11 :** page 2 6](#_Toc130831149)

[**Figure 12 :** liste des courses dans le navigateur (partiel) 7](file:///F%3A%5CPedago%5C2_Enseign_Gene_SI_NSI%5C20a_NSI%5CTNSI%5CC_Bases%20de%20donn%C3%A9es%5C2_TP%20_SQL%5CTP2%20SQL_Python_Web%5C1_Sujet%5CTNSI_TP_BDD_Langage_SQL_SQLite_Python_Bottle.docx#_Toc130831150)

[**Figure 13** : table todo dans le navigateur (partiel) 8](file:///F%3A%5CPedago%5C2_Enseign_Gene_SI_NSI%5C20a_NSI%5CTNSI%5CC_Bases%20de%20donn%C3%A9es%5C2_TP%20_SQL%5CTP2%20SQL_Python_Web%5C1_Sujet%5CTNSI_TP_BDD_Langage_SQL_SQLite_Python_Bottle.docx#_Toc130831151)

[**Figure 14 :** suppression d’items dans la liste des choses à faire 8](file:///F%3A%5CPedago%5C2_Enseign_Gene_SI_NSI%5C20a_NSI%5CTNSI%5CC_Bases%20de%20donn%C3%A9es%5C2_TP%20_SQL%5CTP2%20SQL_Python_Web%5C1_Sujet%5CTNSI_TP_BDD_Langage_SQL_SQLite_Python_Bottle.docx#_Toc130831152)

[**Figure 15** : formulaire pour l'ajout d’items 9](#_Toc130831153)

[**Figure 16** : résultat final 9](#_Toc130831154)

[**Figure 17** : MCD de la base "Séjour" 10](#_Toc130831155)

[**Figure 18** : MLD de la base "Séjour" 10](#_Toc130831156)

# Mise en situation

## Site Web dynamique ?

Les sites Web que nous consultons sont la plupart du temps des sites **dynamiques**. Ces sites sont constitués de pages dont le contenu peut varier en fonction de **données** (heure, nom de l’utilisateur, formulaire rempli par l’utilisateur, etc.). Ces données sont stockées dans une **base de données** gérée par un système de gestion de bases de données (**SGBD**). Le **code** écrit en PHP, Ruby, **Python** etc. se charge de répondre aux requêtes envoyées par le client en renvoyant des pages HTML contenant des données situées dans la base de données.



**Figure 1** : code source pour un site Web

Le schéma ci-dessous représente l’organisation classique du modèle client-serveur sur le Web.



**Figure 2** : modèle client-serveur

## Cahier des charges de la page « Ma liste des choses à faire »



**Figure 3** : exemple de page "Ma liste des choses à faire"

|  |
| --- |
| Cette page doit :- **présenter** la liste des « Choses à faire » dans des rubriques,- **proposer** la **suppression** de ce qui a été fait,- **proposer** l’**ajout** de nouvelles rubriques et de nouvelles « choses à faire » dans ces rubriques. |

Pour simplifier la conception et le test du site « Todo List » **sur le PC**, celui-ci sera codé en **Python** (côté serveur) et en **HTML**, **CSS** (côté client). Le serveur sera réalisé avec le micro Framework Web **Bottle**. Les données seront sauvegardées dans une base de données **SQLite**.

Le schéma précédent devient :



**Figure 4** : modèle client-serveur du projet

# Étude de la problématique



**Figure 5 :** dossier du projet "TodoList"

Pour répondre au cahier des charges, vous allez écrire du code en Python, SQL et HTML afin :

- d’interroger et de mettre à jour une base de données.

- d’afficher les données issues de cette base dans un navigateur.

**Ceci sera détaillé dans les prochains paragraphes.**

|  |
| --- |
| Le site contenu dans le dossier « **Todolist** » est organisé comme sur la figure ci-contre.* Le répertoire **bdd** contient la base de données et son code de création. Celui-ci permet de la reconstruire en cas de problème.
* Le répertoire **css** contient une feuille de style (utilisée à la fin du TP)
* Le répertoire **template** contient les modèles de page Web.
 |

**À l’aide d’un glisser-déplacer, chargez le dossier « TODOLIST » dans** **l’IDE VSCode**

## La base de données « Todo »

|  |  |
| --- | --- |
| La liste des « Choses à faire » est sauvegardée dans la base SQLite « Todo ». Cette base de données, contenue dans le fichier **todo.db,** est constituée de la table **todo** organisée comme ci-contre.**1.** **Affichez** l’explorateur SQLiteTouche **F1** → **SQLite : Open Database** et sélectionnez ***todo.db***. L’explorateur SQLite doit s’ouvrir en bas et à gauche de la fenêtre comme sur la copie d’écran ci-contre. | **Figure 6 :** SQLite Explorer dans VSCode |
| **2. Visualisez** le contenu de la table ***todo*** en cliquant sur le symbole Lirequi apparaît au survol de la souris dans **SQLITE EXPLORER**. Vous devez obtenir le résultat ci-dessous. |



**Figure 7 :** table "todo" initiale

## Interagir en Python avec une base de données



|  |  |
| --- | --- |
| Informations | Le **module** **sqlite3** permet d’interagir en Python avec un **SGBD SQLite**. On communique avec cette base de données en utilisant les **classes** et les **méthodes** définies dans ce module.Pour importer ce module dans un fichier, il suffit d’écrire ***import sqlite3.***Interagir avec une base de données nécessite de :**- Créer une connexion ;****- Créer un curseur pour exécuter des requêtes ;****- Se déconnecter** |

**Ceci sera détaillé dans les prochains paragraphes.**



**Figure 8** : résultat attendu dans la fenêtre interactive

### Création d’une connexion

|  |  |
| --- | --- |
| Informations | Pour utiliser le module **sqlite3**, on doit d’abord créer une [connexion](https://docs.python.org/fr/3/library/sqlite3.html#sqlite3.Connection) avec la méthode ***connect()*** et lui passer l’emplacement du fichier de stockage de la base de données (.db) en paramètre. Si celui-ci n’existe pas, il est créé.  |

Entrez la commande suivante sous ***# 2. Connexion à la base « todo »*** dans le fichier ***testreq.py***.

|  |  |
| --- | --- |
| Document | connexion = sqlite3.connect("bdd/todo.db") |

### Exécution des requêtes

|  |  |
| --- | --- |
| Informations | Pour exécuter les requêtes SQL avec du code Python, on se sert d’un **objet *cursor*,** récupéré en faisant appel à la méthode *cursor* de l’objet *connexion***.** |

* **Création d’un objet curseur**

Entrez la commande suivante sous ***# 3. Création d’un curseur*** dans le fichier *testreq.py*.

|  |  |
| --- | --- |
| Document | *curseur = connexion.cursor()* |

* **Modifications apportées à une table**

|  |  |
| --- | --- |
| Informations | Lorsque l’on effectue des modifications sur une table (**insertion**, **mise à jour** ou encore **suppression** d’éléments), celles-ci ne sont pas automatiquement validées. Ainsi, sans validation, les modifications ne sont pas effectuées dans la base et ne sont donc pas visibles par les autres connexions. Pour résoudre cela, il faut utiliser la méthode **commit** de l’objet de type *Connection*. |

* **Exécution d’une requête**

La méthode ***execute()*** appliquée au curseur exécute **UNE** requête. On souhaite ajouter du chocolat à la liste de courses. Entrez les commandes suivantes sous ***# 4a) UNE requête d’INSERTION*** dans le fichier *testreq.py*.

|  |  |
| --- | --- |
| Document | *donnees = ("Courses", "chocolat")**curseur.execute("INSERT INTO todo (categorie, sujet) VALUES (?, ?)", donnees)**connexion.commit()* |

|  |  |
| --- | --- |
| Avertissement | A l’aide de , **testez** **ligne par ligne** le code entré jusqu’à présent en exécutant dans l’ordre les cellules **1 → 4a**.Pour voir les effets de la requête, il faut mettre à jour la base *todo* dans **SQLITE EXPLORER** (VSCode) avec l’icône  et recharger la table *todo* avec . |

* **Exécution de plusieurs requêtes**

La méthode ***executemany()*** appliquée au curseur exécute **PLUSIEURS** requêtes. Entrez les commandes suivantes sous

***# 4b) Plusieurs requêtes d’INSERTION*** dans le fichier *testreq.py et testez-les (4b seul)*.

|  |  |
| --- | --- |
| Document | *donnees = [("Loisirs", "voile"),("Courses", "sucre"),("Projets", "voyage")]**curseur.executemany("INSERT INTO todo (categorie, sujet) VALUES (?, ?)", donnees)**connexion.commit()* |

* **Parcours des enregistrements**

|  |  |
| --- | --- |
| Informations | Pour obtenir des éléments contenus dans la base de données, on **exécute une requête** puis on récupère le résultat de la requête à l’aide de **l’objet de type *cursor***. |

Les méthodes *fetchone*(), f*etchmany*() ou *fetchall*() permettent de **récupérer le résultat** de la requête. Ce résultat est fourni sous la forme d’un **tuple**, d’une **liste de tuples** ou **None** s’il n’y en a pas.

* ***fetchone()*** récupère **un seul** résultat.
* ***fetchmany(n)*** récupère **n résultats** en une opération.

Remarque : Le nombre de résultats prend par défaut la valeur de l’attribut ***arraysize*** du curseur.

* ***fetchall()*** récupère **tous les résultats** de la requête.

*Pour voir l’effet de ces trois commandes*, entrez le code suivant dans le fichier *testreq.py* sous ***# 4c) Parcourir des enregistrements.***

|  |  |
| --- | --- |
| Document | *curseur.execute("SELECT \* FROM todo")**resultat = curseur.fetchone()**print(f"Cas 1. Le seul résultat récupéré est {resultat}")****Résultat attendu****n = 3**curseur.execute("SELECT \* FROM todo")**resultat = curseur.fetchmany(n)**print(f"Cas 2. Les {n} résultats récupérés sont {resultat}")**curseur.execute("SELECT \* FROM todo")**resultat = curseur.fetchall()**print(f"Cas 3. Tous les résultats : {resultat}")****Résultat attendu*****Figure 9 :** liste de courses dans la fenêtre interactive |

**Activité 1** : on souhaite récupérer la liste des courses à faire dans une variable *lstCourses* et l’afficher dans la fenêtre interactive comme ci-contre.

**Complétez** le code Python sous *# Activité 1* dans le fichier *testreq.py.* ***Testez-le*** et complétez le document réponse 1 (**DR1)**.

### Déconnexion

|  |  |
| --- | --- |
| Informations | On applique la méthode **close()** à la connexion pour la fermer. |

## Création d’un serveur Web avec le micro Framework Bottle



Pour afficher les données issues de la base et saisir les modifications à réaliser dans un navigateur, il faut créer un serveur HTTP. Ceci sera réalisé avec le micro Framework Bottle.

|  |  |
| --- | --- |
| Informations | Bottle est un micro Framework Web [**WSGI**](http://www.wsgi.org/) (**W**eb **S**erver **G**ateway **I**nterface) rapide, simple et léger pour Python. Il est distribué sous la forme d'un module de fichier unique et n'a pas de dépendances autres que la bibliothèque standard Python. |

### Prise en main

Pour comprendre la philosophie de Bottle nous allons mettre en œuvre un exemple. Entrez le code ci-dessous dans le fichier *hello.py.*

|  |  |
| --- | --- |
| Document | import bottle# Définition d'une route et d'un gestionnaire @bottle.route('/hello')def hello():    return "Hello World !"# run() démarre le micro serveur Web Bottle sur: http://127.0.0.1:port/bottle.run(bottle.app(), host='127.0.0.1', port=5000, debug=True, reloader=True) |

**1.** Exécutez ce script avec .

Un texte similaire à celui reproduit ci-dessous doit apparaître dans le TERMINAL de VSCode pour indiquer que le serveur est démarré et attend des requêtes HTTP.

*Bottle v0.12.18 server starting up (using WSGIRefServer())...*

*Listening on http://127.0.0.1:5000/*

*Hit* ***Ctrl-C to quit****.*

**2.** Ouvrez un navigateur et entrez : **127.0.0.1:5000/hello**. Celui-ci doit afficher :



****

|  |  |
| --- | --- |
| Avertissement | **Arrétez le serveur** en entrant **Ctrl-C** **dans la console** pour pouvoir le redémarrer dans le prochain fichier Python.. |

|  |  |
| --- | --- |
| Informations | **Fonctionnement****@bottle.**[**route()**](https://bottlepy.org/docs/dev/api.html#bottle.route) est un décorateur qui lie un **morceau de code** à un **appel d'URL**. Dans le cas présent, nous lions **/hello** à la fonction hello(). C'est ce qu'on appelle une **route** (d'où le nom du décorateur). On peut définir autant de routes qu’on le souhaite. Chaque fois qu'un navigateur demande une URL, la fonction associée est appelée et **la valeur de retour est envoyée au navigateur**.  |

**Activité 2** : on souhaite créer les deux pages Web ci-dessous. Dans chacune des pages, un clic sur le lien « **ici** » doit afficher l’autre page.

**Complétez** le code du fichier ***pages.py***. Les titres sont à réaliser avec des balises <h1>, les lignes de texte avec des balises <p> et les liens avec des balises <a>. Notez votre code sur le **DR2**.

 *Exemple de résultat attendu*

|  |  |  |
| --- | --- | --- |
| **Figure 10 :** accueil |  | **Figure 11 :** page 2 |

## Affichage des données issues de la base « Todo » dans une page Web

Vous allez maintenant intégrer des données extraites de la base Todo dans une page Web.

### Solution 1 : pages Web dans le code Python

Le fichier ***todolist1.py*** contient le code ci-dessous. **Testez-le** pour voir le résultat dans le navigateur.

|  |  |
| --- | --- |
| Document | import sqlite3import bottle# Définition d'une route et d'un gestionnaire pour la page d'accueil@bottle.route('/')def accueil():    # 1. Connexion à la base    connexion = sqlite3.connect('bdd/todo.db')    # 2. Création d'un curseur pour exécuter les requêtes    curseur = connexion.cursor()    # 3. Exécution des requêtes    curseur.execute("SELECT \* FROM todo ORDER BY categorie,sujet")    # Le résultat renvoyé par la requête est une liste de tuples    resultat = curseur.fetchall()    # 4. Déconnexion    curseur.close()    # 5. Construction de la page et transmission au navigateur    return '<h2>Ma liste des "choses à faire"</h2>' + str(resultat)# run() démarre le micro serveur Web Bottle sur: http://127.0.0.1:port/bottle.run(bottle.app(), host='127.0.0.1',           port=5000, debug=True, reloader=True) |



**Figure 12 :** liste des courses dans le navigateur (partielle)

|  |  |
| --- | --- |
|  | La table todo est renvoyée triée par ORDER BY. On notera que faire un **prétraitement** **des données** avec le langage SQL simplifie souvent le post-traitement à réaliser avec le langage du serveur (Python, PHP, etc.). |

**Activité 3** : on souhaite obtenir l’affichage de la liste des courses dans le navigateur comme sur la copie d’écran du document réponse 3 (**DR3)**.

**Modifiez** le code du fichier ***todolist2.py*** pour que la page Web s’affiche comme sur la copie d’écran du **DR3**. Les sous-titres sont à réaliser avec des balises <h3> et les éléments de liste avec des balises <li>. Un retour à la ligne après </li> nécessite \n. Notez votre code sur le **DR3**.

*Indication : la variable html doit contenir la concaténation des données extraites de la base et des balises HTML.*

|  |  |
| --- | --- |
| Avertissement | On peut mettre du code HTML dans le code Python pour les petites applications, mais **pour les projets Web plus volumineux**, il est recommandé de séparer le HTML de la base de données ou du code backend avec des **modèles Web**. |

### Solution 2 : modèle Web (template)

|  |  |
| --- | --- |
| Informations | Les modèles Web (templates) séparent la base de données et la logique, de la présentation Web. **Ces modèles permettent de construire des pages HTML.** Pour les distinguer, on leur donne l’**extension tpl**. Bottle a un moteur de modèle intégré appelé *Simple Template Engine*. |

Cette nouvelle solution est construite avec deux fichiers : ***todolist3.py*** et ***pagev1.tpl*** *(située dans le répertoire template).*

* ***pagev1.tpl*** contient le code HTML de la page et du code python **formaté**.
* ***todolist3.py*** gère la base de données et identifie le modèle Web ***pagev1.tpl*** auquel il transmet les données***.***

*Exemple*

html = bottle.template('template/pagev1.tpl', **table**=resultat) # Code extrait de *todolist3.py*

Pour construire la variable html, on passe le chemin vers le modèle web *page1.tpl* et le « resultat » de la requête SQL à la méthode « template » de Bottle. Le **traitement à réaliser** pour présenter les données dans la page est placé dans *pagev1.tpl*. Le lien entre les deux fichiers se fait avec la variable **table**.

**Extrait** de *pagev1.tpl*

 %categorie = ''

 %for ligne in **table**:

 %if categorie != ligne[0]:

 %categorie = ligne[0]

 <h3>{{ligne[0]}}</h3>

 %end

 <li>{{ligne[1]}}</li>

 %end

La syntaxe du code ci-dessus nécessite **quelques explications**.

|  |  |
| --- | --- |
| Informations | **Les modèles** sont des **pages HTML** avec du **code Python**. Le code Python peut être soit dans des **blocs** avec un **<%** pour démarrer le bloc et un **%>** pour terminer le bloc, soit chaque ligne peut commencer par **%**. **{{…}}** permet d’intégrer une expression Python à condition qu’elle corresponde à quelque chose ayant une représentation sous la forme d’une chaîne de caractères.Deux des principales différences du code Python de modèle en ligne avec le Python sont :* l'**indentation** de la ligne de Python n'est **ni requise ni utilisée** (mais recommandé pour faciliter la relecture)
* les **instructions de contrôle** comme: if,  for, etc. **ont besoin d'une instruction de fin (%end)**
 |

**Testez** le code de la page *todolist3.py* dans un navigateur. Vous devez obtenir un résultat équivalent à celui de *todolist2.py.*



**Figure 13** : table todo dans le navigateur (partielle)

**Activité 4** : on souhaite obtenir l’affichage de la table todo dans le navigateur comme sur la copie d’écran du document réponse 3 (**DR3**).

**Ajoutez** le code nécessaire dans le fichier ***pagev1.tpl*** pour que le tableau de la copie d’écran du **DR3** s’affiche à la suite de la liste des courses.

En HTML, un tableau est réalisé avec les balises : <*table>*, <*tr>* et <*td>*. **Notez** votre code sur le **DR3**.

Vous avez vu comment intégrer les données de la base todo à un modèle. Vous allez maintenant donner à l’internaute la possibilité de **mettre à jour la base** en proposant les fonctionnalités **SUPPRIMER** un élément et **AJOUTER** un élément dans la page Web.

## Mise à jour de la base « Todo » avec des formulaires

### Suppression d’éléments

Le modèle Web ***pagev2.tpl*** reprend le code de *pagev1.tpl*. Comme la page Web doit transmettre des informations, un **formulaire** lui est ajouté.

**Extrait** de *pagev2.tpl*

 <form action="/retrait" method="POST">

 %categorie = ''

 %for ligne in table:

 %if categorie != ligne[0]:

 %categorie = ligne[0]

 <h3>{{ligne[0]}}</h3>

 %end

 % # Ajout d'un bouton de suppression après chaque item

 <li>{{ligne[1]}}

 <button type="submit" name="**supprimeItem**" value="{{ligne[2]}}" class= "bouton">Supprimer</button>

 </li>

 %end

 </form>

La fonction **SUPPRIMER** est réalisée en plaçant un bouton « Supprimer » à côté de chaque item. Un clic sur le bouton transmet son id unique (ligne2) à la fonction **retirer()** chargée de le supprimer. Le formulaire a une requête POST avec l’action ciblant « /retrait » dans ***todolist4.py***.



**Figure 14 :** suppression d’items dans la liste des choses à faire

**Extrait** de *todolist4.py*

@bottle.route('/retrait', method='POST')

def **retirer**():  # Retrait d'un élément dans la table todo

 # Récupération de l’id de l’élément à supprimer

idelem = bottle.request.forms.get("**supprimeItem**").strip()

    # b1. Connexion à la base de données à compléter

  # b2. Création d'un curseur pour exécuter les requêtes à compléter

  # b3. Exécution de la requête

  sqlstr = f"DELETE FROM todo WHERE id={str(idelem)}"

  curseur.execute(sqlstr)

  # b4. Mise à jour de la table à compléter

  # b5. Déconnexion

  curseur.close()

 # b6. Retour à la page principale pour afficher la liste modifiée

  bottle.redirect("/")

**Activité 5** : **complétez** le code de la page *todolist4.py* et **testez-le** dans un navigateur. Vous devez obtenir un résultat comme sur la copie d’écran ci-contre. Il doit être possible de supprimer les items. Complétez le **DR4**.

### Ajout d’éléments

L’ajout d’éléments dans la base se fera avec le **formulaire** ci-dessous.



**Figure 15** : formulaire pour l'ajout d’items

On donne le code suivant.

|  |  |
| --- | --- |
| Document |         <hr>        <div id="rem2">Compléter le formulaire pour ajouter une nouvelle tâche.</div>        <form action="/ajout" method="POST">            <select name="nouvelleCategorie">                <option value="Courses" selected>Courses</option> <!-- A compléter -->            </select>            <input type="text" size="25" name="sujet">            <input type="submit" name="save" value="Sauvegarder">        </form> |

**Activité 6** : **ajoutez** et **complétez** le code ci-dessus dans le fichier ***pagev2.tpl***. Testez son fonctionnement dans un navigateur. Notez votre code sur le **DR4**.

Il reste à concevoir le code permettant de lier le formulaire à la base de données.

**Activité 7** : En vous inspirant du code de la fonction retirer(), concevez le code de la fonction **ajouter()** dans le fichier ***todolist4.py***. Imprimez votre code (voir prof) et joignez-le au document réponse.

*Remarque :* ajouter ***accept-charset="ISO-8859-1"*** *après method="POST"* dans la balise *form* pour **encoder** correctement les caractères accentués*.*

## Ajout d’une feuille de style

Une feuille de style est disponible dans le répertoire css. Son intégration se fait en ajoutant le code ci-dessous dans le fichier *todolist4.py* …

|  |  |
| --- | --- |
| Document | # Chargement du fichier style.css@bottle.route('/css/<filename:path>')def server\_static(filename):    " Fichiers statiques du site placés dans le rép /css/ "    return bottle.static\_file(filename, root='css') |

… et une balise link dans l’en-tête (**head**) du fichier *pagev2.tpl*.

|  |  |
| --- | --- |
| Document | <link rel="stylesheet" href="/css/style.css"> |

Le résultat final doit ressembler à la copie d’écran ci-dessous.



**Figure 16** : résultat final

# Synthèse : idée de mini projet

On rappelle le **modèle conceptuel** de données (**MCD**) de la base « **Séjour** » utilisée dans le TP1 BDD.



**Figure 17** : MCD de la base "Séjour"

On rappelle également le schéma de son **modèle relationnel** (**MLD**)



**Figure 18** : MLD de la base "Séjour"

**Activité 8**: en vous inspirant du travail fait dans ce TP, concevez une application Web destinée à alimenter et à consulter la base de données « Séjour ».

# Annexe : connexion à votre compte et préparation du dossier de travail sur le serveur NAS\_SIN

1. **Connexion au serveur NAS\_SIN pour accéder au compte du cours.**



*  et entrez **\\192.168.231.187** dans l’invite de commande.
* Complétez la fenêtre de sécurité Windows ci-dessous avec l’**identifiant** et le **mot de passe** de votre compte sur le serveur **NAS\_SIN. Vous accédez ainsi aux dossiers partagés sur le serveur.**



**(1)**



**Dossiers partagés sur le serveur NAS\_SIN**

***(1) Votre dossier personnel***

* Double-cliquez sur le dossier **tnsi.**



1. **Dossier à utiliser dans le TP**

Copiez le dossier **Todolist** situé dans ***tnsi/3\_Bases\_de\_donnees/*** et collez-le dans ***home/TP/***