Interaction client-serveur - Plan de travail

1NSI – janvier 2023

Savoir-faire de la séquence

- Distinguer ce qui est exécuté sur le client ou sur le serveur et dans quel ordre. Distinguer ce qui est mémorisé dans le client et retransmis au serveur. Reconnaître quand et pourquoi la transmission est chiffrée.
- Analyser le fonctionnement d'un formulaire simple. Distinguer les transmissions de paramètres par les requêtes POST ou GET.

Exercice 1 — Requêtes et réponses

1. Ouvrir le navigateur, allez chercher les outils de développement (F 12) puis allez sur l'onglet Réseau (ou network).
   (a) Allez à l'adresse `raw.opytex.org`.. Faire apparaitre les colonnes `chemin` (ou path) et `url`.
   (b) En cliquant sur la requête, noter la méthode, le domaine (host), le fichier, l'adresse IP de la requête et l'état (ou status) de la réponse.
   (c) Faire la même chose pour l'adresse `https://raw.opytex.org/1NSI/05_Interaction_client-serveur/`.
   (d) La requête vers l'adresse `https://optez.org` engendre plusieurs requêtes noter les éléments (précisés à la question 1b) pour chacune d'entre elles.
2. Éléments théoriques - recherche documentaire
   (a) Expliquer le rôle que joue les méthodes vues dans les questions précédentes
   (b) Expliquer le rôle que joue les codes status vus dans les questions précédentes

Exercice 2 — Application web avec Bottle - requête GET

Dans cette partie, vous allez programmer une application web en utilisant la librairie python `bottle`. Vous trouverez la documentation (en anglais) à l'adresse suivante

`https://bottlepy.org/docs/dev/tutorial.html`

1. **Mise en place de Bottle et vérifications**
   (a) Copier-coller le code suivant (tiré de la documentation) pour initialiser l'application.

```python
from bottle import get, run

@get('/hello')
def hello():
    return "Hello World!"

run(host='localhost', port=8080, debug=True)
```

   (b) Exécuter votre programme puis rendez-vous avec le navigateur à l'adresse `http://localhost:8080/hello`
   Étudiez la requête (méthode, domaine, fichier et status).
   (c) Rendez-vous avec le navigateur à l'adresse `http://localhost:8080/plop`. Étudiez la requête (méthode, domaine, fichier et status).
   (d) Relier ce que vous avez noté aux questions précédentes avec le code exemple de Bottle.
   (e) Comment peut-on ajouter d'autres chemins à notre application ?

2. **Requêtes GET**

Pour toutes les questions suivantes, vous devrez tester votre travail en visitant la page associée et vous noterez l'URL.

(a) Supprimer le chemin `/hello`.

(b) Ajouter le chemin accessible à la racine de l'application (`/`) qui saluera les nouveaux venus.

(c) Ajouter le chemin `/about` qui vous présentera (en une phrase).

(d) Ajouter le chemin `/bottle` qui vous donnera le lien (cliquable) vers la documentation de Bottle.

(e) (*) On peut créer des chemins dynamiques en suivant la syntaxe suivante

```
@get("/page/<utilisateur>")
def page_utilisateur(utilisateur):
   return "Bonjour " + utilisateur + ", comment allez vous?"
```

Ajouter ce chemin à votre application. Proposez plusieurs URL qui l'utilisent.

(f) On peut aussi configurer des `query`

```
from botte import request, get

@get('/age')
def age_de():

   name = request.query.name
   age = request.query.age

   if age > 1:
      return name + " a " + age + " ans."
   elif age <= 0:
      return "Mouai..."
   else:
      return name + " a " + age + " an."
```

Ajouter ce chemin à votre application sans oublier d'ajouter les nouveaux imports. En reprenant votre cours sur les URL, trouver l'URL qui permet d'avoir sur votre navigateur le message

Bob a 5 ans.

(g) Pour le moment, le code renvoyé par notre application n'est pas du code HTML valide. Il est possible d'écrire des modèles (template en anglais) dans lesquels on va pouvoir injecter des variables python pour les rendre modulables.

À côté du fichier de votre application, créer un fichier `utilisateur.html` dans lequel vous mettrait le code HTML suivant

```html
<!DOCTYPE html>
<html lang=fr>
      <head>
            <meta charset="UTF-8">
            <meta name="Author" content="">
      <title>Page de {{ utilisateur }}</title>
      </head>

      <body>
      <h1>{{ utilisateur }}</h1>
      <p>
         Cette page change en fonction de l'utilisateur
         renseignée dans l'URL.
      </p>
      <p>Elle sait même faire des calculs 1+1 = {{ 1+1 }}</p>
      </body>
</html>
```

Ajoutez la fonction `template` dans les imports de bottle. Puis modifier le chemin `page_utilisateu`
pour avoir le contenu suivant

```python
@get('/page/<utilisateur>')
def page_utilisateur(utilisateur):
   return template("utilisateur.html", utilisateur=utilisateur)
```

Visitez plusieurs pages utilisateur et constatez les différences.
Comment fait-on pour utiliser une variable python dans modèle ?

Exercice 3 — Application web avec Bottle - requête POST

Les requêtes POST envoient des informations au serveur. Ces informations sont le plus souvent envoyé à partir d'une formulaire HTML.

On reprend le code de la question 1a de l'exercice précédent. Puis on ajoute à côté du script de notre application, le modèle suivant :

```html
<!DOCTYPE html>
<html lang=fr>
      <head>
            <meta charset="UTF-8">
            <meta name="Author" content="">
      <title></title>
      </head>

      <body>
      {{ body }}
      </body>
</html>
```

Ce modèle permettra d'avoir une structure HTML propre. Il n'y aura plus qu'à passer le contenu du body.

1. Supprimer la route `hello` et ajouter la route suivante

```python
@get('/formulaire')
def get_formulaire():
    formulaire = """
    <form method='post' action='bonjour'>
    <input type='text' name='nom' placeholder='Votre nom ?'/>
    <input type='submit' value='Bonjour bottle !'/>
    </form>
    """
    return template("modele_base.html", boby=formulaire)
```

2. Quelle URL doit-on entrer dans le navigateur pour accéder à ce formulaire ? Envoyez un nom et étudier la requête.
3. On ajoute maintenant la route suivante du type POST pour réceptionner cette requête. Mais avant cela, vous devez ajouter la fonction `post` dans les imports.

```python
@post('/formulaire')
def bonjour() :
    nom = bottle.request.forms.get('nom')
    body = "<p>Bonjour mon(a) che(è)r(e) " + nom + "</p>"

    return template("modele_base.py", body=body)
```

4. Remplissez le formulaire, envoyez le et étudier les requêtes.
5. **Bilan sur les formulaires et les requêtes post :**
   (a) Identifier les nouvelles balises HTML utiles pour créer un formulaire.
   (b) Chercher sur internet d'autres type possible pour la balise `input`.
   (c) Comment `bottle` fait pour accéder au information du formulaire lors une requête POST ?
   (d) Quelles sont les différences entre transmettre une information avec une requête POST et l'utilisation des query ?

Exercice 4 — Application web avec Bottle - Accès authentifié

Vous devez écrire une application web qui a les routes suivantes :

- **la racine (/)** : contient un formulaire de connexion avec deux champs (utilisateur et mot de passe). L'envoie du formulaire se fait vers la route `/prive`
- **/privé** : route avec deux comportements :
  — Affiche "accès autorisé" sur le couple utilisateur et mot de passe correspond à l'utilisateur et au mot de passe enregistré dans le code l'application.
  — Affiche "accès non autorisé" sinon.

Exercice 5 — Application web avec Bottle - Accès authentifié

Vous devez écrire une application web qui a les routes suivantes :

- **/ (la racine)** : contient plusieurs formulaires. Un qui permet de savoir si une année est bissextile et un autre qui compte le nombre de jours depuis le début de l'année.
- **/bissextile** : qui affiche l'année et qui dit si l'année est bissextile ou non.
- **/jours** : qui affiche la date et le nombre de jour depuis le début de l'année.

Exercice 6 — QCM

1. Parmi GET et POST, quelle méthode d'envoi de formulaire crypte les informations envoyées au serveurs ?

   a) les deux : GET et POST
   b) GET seulement
   c) POST seulement
   d) aucune des deux

2. Parmi les réponses suivantes, que permet d'effectuer la méthode POST du protocole HTTP ?

   a) Définir le style d'une page
   b) Pirater des données bancaire
   c) Envoyer une page web vers le client
   d) Envoyer les données saisies dans un formulaire HTML vers un serveur

3. Charles veut accéder à son forum favori. Il sais son adresse (URL) sur son navigateur Web, qui lui affiche une erreur 404.
   Quelle cas de figure **n'explique pas** sa situation ?

   a) une mise à jour du serveur qui héberge le forum
   b) une erreur de saisie de sa part
   c) une panne de sa connexion internet
   d) un changement de titre du forum qu'il veut consulter

4. Parmi les éléments suivants, lequel est un protocole ?

   a) GET
   b) POST
   c) HTTP
   d) HTML

5. Dans une page HTML, que pet la balise `<form action="http://monsite.fr" method="POST">` ?

   a) d'envoyer des données à l'URL `http://monsite.fr` sans les ajouter au corps de la requête HTTP
   b) d'envoyer des données à l'URL `http://monsite.fr` et de les ajouter au corps de la requête HTTP mais pas à l'URL
   c) de télécharger une formulaire depuis l'URL `http://monsite.fr`
   d) de récupérer des données depuis l'URL `http://monsite.fr`

6. Dans le contexte du Web, qu'est-ce qu'une transmission chiffrée ?

   a) une transmission optimisée pour les grands nombres
   b) une transmission sous forme binaire
   c) une transmission d'information cryptée
   d) une transmission facturée proportionnellement à la taille du message

7. Mehdi a écrit une page HTML contenant des éléments `input` de formulaire. Il place ces éléments de formulaire :

   a) entre la balise `<form>` et la balise `</form>`
   b) entre la balise `<formulary>` et la balise `</formulary>`
   c) entre la balise `<code>` et la balise `</code>`
   d) entre la balise `<script>` et la balise `</script>`

8. Quelle utilisation faut-il avoir pour garantir qu'une transmission entre un client et un serveur sera-t-elle chiffrée ?

   a) lorsqu'on utilise le navigateur web Firefox
   b) lorsqu'on utilise la méthode POST
   c) lorsqu'on utilise le protocole HTTPS
   d) lorsqu'on utilise HTML et CSS