Master 1 informatique – Université Claude Bernard Lyon 1 – mercredi 27 février 2019

M1IF 03 – Conception d'applications Web – Examen

Durée : 2 heures – Documents autorisés (4 pages max) – Ordinateurs, calculatrices, tablettes, téléphones portables… interdits

Questions de cours (barème : 12 points)

Maximum : 1 phrase ET 3 lignes (en caractères lisibles). Les réponses plus longues ne seront pas lues. Il est inutile de recopier les slides du cours.

1. Pourquoi une requête GET ne doit-elle pas posséder d'en-tête `Content-Type` ?
   Il n'y a pas de contenu dans une requête GET ; les paramètres sont passés dans l'URL.
2. Donnez une raison pour laquelle certains langages « au-dessus » de HTTP (par exemple SOAP ou SPARQ) envoient toutes leurs requêtes en POST.
   - contenus des requêtes non limités en taille
   - les contenus n'apparaissent pas en clair dans l'URL
3. Expliquez en quoi les serveurs répondant aux requêtes décrites à la question précédente perdent en scalabilité.
   POST n'est pas cachable.
4. En quoi l'utilisation de `workers` par nginx améliore-t-elle la sécurité lors de l'exécution de scripts CGI ?
   Les workers ne sont pas exécutés en tant que root -> moins de risque de dommages côté serveur en cas d'utilisation malveillante ou de script buggué.
5. Quel type d'outil faut-il utiliser pour que l'accès à différents serveurs (hôtes / numéros de ports différents) soit transparent pour les clients (même URL de base) ?
   Reverse proxy.
6. Dans un serveur en Java, quelle est la différence entre un attribut et un paramètre de requête ?
   Un paramètre est positionné par le client et un attribut par le serveur.
7. On dit que Spring possède un conteneur « léger ». Mais qu'est-ce qu'un conteneur « lourd » ?
   Un conteneur dont l'API contraint les composants à un fonctionnement complexe (héritage de classes, implémentation d'interfaces). Ex. : public class NewServlet extends HttpServlet {
   protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException{
8. Pourquoi place-t-on les JSP de Spring dans le répertoire WEB-INF d'un projet ?
   Pour ne pas qu'elles puissent être requêtées directement par le client sans passer par un contrôleur.
9. Quel est le mécanisme fourni nativement par l'API Servlet pour partager des objets entre différents éléments (servlets, JSP…) d'une application Web ?
   Le contexte applicatif.
10. Que doit faire un serveur Web à la réception d'une requête asynchrone de type AJAX ? Pourquoi ?
    Positionner le header Content-Type car par défaut, un serveur renvoie du text/html.
11. En termes de performance, donnez un avantage et un inconvénient au fait de réaliser une application Web sous forme de Single-Page Application.
    Avantage : pendant le fonctionnement, transfert de données uniquement (sans la présentation), et asynchrone -> plus fluide (car contenu de la réponse plus petit) et ne bloque pas l'interface ;
    Inconvénient : plus lourde à charger au démarrage -> CRP plus long.
12. En quoi l'utilisation de Content Delivery Networks peut-elle améliorer le chemin critique de rendu d'une application ?
    Elle permet d'aller chercher des ressources dans le cache du navigateur, économisant ainsi des requêtes HTTP.

Programmation (barème : 10 points)

13. Écrivez la méthode de service d'une `HttpServlet` qui enregistre le paramètre de requête `"pseudo"` dans la session utilisateur et renvoie un code de réponse vide (`No Content`). Ne vous préoccupez pas de la gestion des erreurs.
    protected void doPost(HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException {
    req.getSession(true).setAttribute("pseudo", req.getParameter("pseudo"));
    res.setStatus(204); } // ou SC_NO_CONTENT
14. Écrivez le contenu de l'élément `<body>` de la page JSP qui récupère ce pseudo depuis la session et l'affiche dans la page, précédé de « Bonjour ».
    bonjour<%= ${ sessionScope.pseudo } %>
    Accepté aussi : toute autre syntaxe correcte.

15. Écrivez la méthode de service d'un filtre Java qui vérifie que la session contient bien un pseudo ; si oui, elle applique le pattern « chaîne de responsabilité » ; si non, elle renvoie une erreur d'authentification (« Forbidden »).

```
public void doFilter(ServletRequest req, ServletResponse res, FilterChain chain) throws IOException,  ServletException {
   if( req.getSession(false) != null && req.getSession().getAttribute("pseudo") != null ) { chain.doFilter(req, res); }
   else { res.setStatus(403); } }
```

16. Écrivez la méthode d'un contrôleur Spring d'une application de chat, qui renvoie au client la liste des messages de l'utilisateur identifié (en XML ou en JSON), en interrogeant la méthode `getMessagesFromUser()` d'un bean `gestionMessages` (déjà injecté). On ne s'occupe pas ici du mapping des données.

```
@GetMapping(value = "/my-messages", produces= MediaType.APPLICATION_JSON_VALUE)
public @ResponseBody List getMyMessages(@SessionAttribute(name = "pseudo") String pseudo) {
      return gestionMessages.getMessagesFromUser(pseudo);    }
```

17. Écrivez une fonction JavaScript qui requête le contrôleur précédent (à l'adresse "`/my-messages`") en utilisant jQuery ; si le serveur renvoie un objet, elle passe cet objet à une fonction de callback ; sinon, elle change le hash de l'URL pour "`login`".

```
function getMyMessages(callback) {
   $.getJSON('/my-messages').done(callback).fail(() => { location.hash= 'login'; });
}
```

18. Écrivez un template de l'objet précédent (contenant une propriété `messages` qui a pour valeur un tableau de strings) qui produit une liste HTML avec Mustache.

```
<ul>{{ #messages }}
   <li>{{ . }} </li>
{{ /messages }}</ul>
```