Master 1 informatique – Université Claude Bernard Lyon 1 – lundi 15 janvier 2018

M1IF 03 – Conception d'applications Web – Examen

Durée : 1 heure 30 – Documents autorisés (4 pages max) – Ordinateurs, calculatrices, tablettes, téléphones portables… interdits

Questions de cours (barème : 12 points)

1. Décrivez les étapes du processus de négociation de contenus entre un client et un serveur en HTTP 1.1.
   Le client positionne les headers ACCEPT-xxx dans la requête pour indiquer un (ou plusieurs) type(s) de contenus (avec des valeurs de préférence) qu'il est capable de gérer. En fonction de cela, le serveur choisit le meilleur type de contenus à renvoyer et positionne les headers CONTENT-xxx.
2. Indiquez en quoi protocole AJP interne à Tomcat est similaire à l'interface côté serveur CGI. Précisez également leur principale différence de fonctionnement.
   AJP et CGI permettent de faire le lien entre un serveur HTTP (Apache) et un programme de traitement des requêtes. AJP encapsule les headers et les paramètres de la requête dans des objets alors que CGI utilise des variables d'environnement.
3. Que faut-il faire pour pouvoir utiliser le tag JSP `<c:out>`, et quel est son équivalent dans une servlet ?
   Il faut importer la bibliothèque JSTL Core ; il correspond à out.println() dans une servlet.
4. Décrivez (ou illustrez avec un diagramme UML dynamique) les étapes du traitement côté serveur d'une requête avec l'API Java, servlet (+ JSP + …), avec un pattern MVC pull-based de type 2 (contrôleurs délégués). Indiquez le type de chaque objet.
   Contrôleur principal (servlet) -> contrôleur de CU / délégué (servlet) -> vue (JSP) -> modèle (POJO) et chemin inverse.
5. En Spring, quel est l'intérêt de placer les composants de la vue dans un sous-dossier du dossier WEB-INF ? Citez deux types de composants que vous pouvez utiliser pour composer dynamiquement les vues côté serveur.
   Cela interdit l'accès direct à ces composants sans passer par le framework (et éventuellement une couche de contrôle d'accès).
   JSP + n'importe quel nom d'outil de templating côté serveur avec une implémentation en Java accepté.
6. En quoi le principe d'idempotence favorise-t-il la performance côté serveur ? En quoi a-t-il une influence sur l'API Navigation Timing côté client ?
   Il permet la mise en cache des réponses du serveur. Le client doit donc chercher, pour les méthodes idempotentes si la réponse est déjà dans son cache, et le temps passé à effectuer cette opération est indiqué dans les étapes du chargement d'une ressource, accessible avec l'API N.T.

Étude de cas (barème : 10 points)

Suite aux différents problèmes et dysfonctionnements du système informatique de gestion des TGVs, la SNCF vous a engagé(e) pour reconcevoir l'application Web de surveillance et de contrôle des aiguillages. Cette application devra permettre à ses utilisateurs :

- De connaître à tout moment l'état des aiguillages sur les voies ou celui d'un aiguillage donné (1)
- De modifier l'état d'un aiguillage (2)
- De programmer le changement d'état d'un aiguillage à une date et une heure donnée (3)
- De vérifier ou d'annuler un ordre de changement d'état (4)

Quelques indications supplémentaires :

- Un aiguillage possède un identifiant et deux positions (par ex : Voie_A et Voie_B) qui indique de quel côté il oriente les trains. Chaque aiguillage est équipé d'un capteur indiquant de quel côté il est positionné, et d'un actionneur pour modifier sa position.
- Ces capteurs et actionneurs sont tous reliés à un serveur HTTP central. Ce serveur utilise les technologies vues en cours : Spring Web MVC, REST, et AJAX (le serveur ne sert que des pages HTML statiques ou des données de capteurs en JSON).
- Côté client, on utilise un système de cartographie propriétaire pour visualiser les voies et les aiguillages, que l'on charge avec un script nommé `carteVoies.js` et accessible dans le répertoire `/scripts` du serveur. Ce script « connaît » les localisations des aiguillages et est capable de les représenter sur la carte, à l'aide d'une méthode `update(id, position)`. Les opérations autres que la visualisation des positions des aiguillages (positionnement, programmation…) sont réalisées à l'aide d'interfaces textuelles.

Conception (barème : 6 points)

7. Listez les différents éléments dont vous aurez besoin pour réaliser cette application. Indiquez leurs types (composants Spring (précisez le type), classes Java, pages statiques, scripts…) et à quoi ils servent.
   Contrôleurs : Voie, Aiguillage, Ordre
   Beans : Voie, Aiguillage, Ordre

Vues (-> XML, JSON…) : Voie, Aiguillage, Ordre
Pages statiques (HTML) : formulaires simples, remplissage en AJAX, ou dynamiques (JSP) : formulaires pré-remplis
Scripts métier : visualisation / modification des aiguillages pour effectuer les tâches 1 à 4
Autres ressources : CSS, jQuery, carteVoies…
+ filtres, config Spring (bonus)

8. Décrivez l'API du serveur permettant de mettre en œuvre les opérations listées dans les lignes (1) à (4) ci-dessus.

(1) : renvoie l'état de tous les aiguillages d'une voie
Requête : GET …/voie/{id}
Paramètres : id : id de la voie
Contenu retourné : tableau d'aiguillages avec leurs ids et positions
Code de retour : 200 OK / 404 Not Found (si id erroné)

(1 bis) : renvoie l'état d'un aiguillage
Requête : GET …/augiuillage/{id}
Paramètres : id : id de l'aiguillage
Contenu retourné : id et position de l'aiguillage
Code de retour : 200 OK / 404 Not Found

(2) : modifie l'état d'un aiguillage (actionneur)
Requête : PUT …/voie/{id}
Paramètres : id : id de l'aiguillage, position souhaitée
Contenu retourné : aucun
Code de retour : 204 No Content / 400 Bad Request (si manque un paramètre) / 404 Not Found

(3) : ajout d'un nouvel ordre
Requête : POST …/ordre
Paramètres : 1 ordre (id aiguillage, position, date, heure)
Contenu retourné : aucun, mais un header LINK vers la nouvelle ressource créée (facultatif)
Corde de retour : 201 Created / 404 Not Found

(4) : vérification de l'état d'un ordre
Requête : GET …/ordre/{id}
Paramètres : id : id de l'ordre
Contenu retourné : représentation de l'ordre (id aiguillage, position, date, heure)
Code de retour : 200 OK / 404 Not Found (si id erroné)

(4 bis) : suppression d'un ordre
Requête : DELETE …/ordre/{id}
Paramètres : id : id de l'ordre
Contenu retourné : aucun
Code de retour : 204 No Content / 404 Not Found (si id erroné)

Programmation (barème : 4 points)

9. Pour des raisons de performance, la mise à jour côté client se fait voie par voie (plutôt qu'aiguillage par aiguillage). Écrivez la fonction JavaScript qui requête le serveur de façon asynchrone (toutes les 10 secondes) pour une voie, et met à jour la carte pour tous les aiguillages de la voie.
   - Fonction JS avec l'id de la voie en paramètre
   - Requête AJAX à (1)
   - Callback -> boucle sur les aiguillages de la voie
   - Call update(id aiguillage, position)
   - Timeout

10. Pour des raisons de sécurité, chaque modification de la position d'un aiguillage est accompagnée d'une requête au capteur correspondant (pour vérifier que la requête a bien été prise en compte et qu'il n'y a pas de défaillance matérielle). Écrivez la fonction JavaScript qui réalise ces deux opérations.
    - Fonction JS avec l'id de l'aiguillage et la position en paramètre
    - Requête AJAX à (2)
    - Callback -> Requête AJAX à (1 bis)
    - Test de la position / paramètre initial
    - Remontée d'erreur en fonction du résultat du test