Master 1 informatique – Université Claude Bernard Lyon 1 – mardi 16 décembre 2014

MIF 13 – Programmation Web – Examen

Durée : 1 heure 30 – Documents autorisés – Ordinateurs, calculatrices et téléphones portables interdits

Questions de cours (barème : 12 points)

1. Quel peut être l'inconvénient d'un système de load-balancing sur les requêtes idempotentes et comment y faire face ?
   Les requêtes idempotentes sont par nature cachables. Problèmes de performance si la même requête est envoyée à plusieurs serveurs →centraliser la gestion du cache. Problème de gestion des utilisateurs identifiés →algo de balancing en fonction de l'adresse IP (cf. TP1).
2. Indiquez les actions que doit effectuer un client pour dupliquer une ressource RESTful côté serveur.
   GET à l'URI de la ressource,
   POST à l'URI de la nouvelle ressource.
3. En quoi la négociation de contenus impacte-t-elle la structure d'une fonction de callback en JavaScript ?
   Il faut réagir en fonction du type MIME renvoyé pour représenter la ressource par le serveur.
4. Comment Spring s'y prend-il pour instancier automatiquement des beans dont les constructeurs comportent des paramètres ?
   Pattern IoC : le container Spring utilise 1) le fichier XML de config, 2) les annotations (« @Autowired ») et 3) un mécanisme d'introspection pour résoudre le référentiel de dépendances et instancier les classes dans l'ordre nécessaire pour disposer des instances à passer en paramètre du bean.
5. Quel type d'outil utiliseriez-vous pour refactorer une application Java fenêtrée en application Web ? Donnez un exemple d'outil de ce type.
   Bibliothèque logicielle indirecte, ex. Google Web Toolkit.
6. Citez deux raisons pour lesquelles on peut dire qu'une application Web utilisant REST et AJAX passe plus facilement à l'échelle qu'une application Web « classique ».
   REST : pas de stockage de l'état du client côté serveur →pas de gestion des sessions.
   AJAX : déport de la mise en forme et du métier côté client.
   Dans les 2 cas, moins de ressources consommées côté serveur →les serveurs peuvent servir plus de clients.

Étude de cas (barème : 10 points)

Suite à l'explosion des lettres de demandes de cadeaux, le Père Noël a décidé de moderniser son infrastructure de traitement du courrier. Il utilise actuellement une application Java qu'il a développée lui-même pendant l'été. Cette application permet de gérer :

- Les produits (cadeaux de tous types)
- Les prospects (enfants ayant envoyé une lettre de demande de cadeau)
- Les demandes (cadeaux demandés, ordre des souhaits)
- Le suivi du stock de cadeaux (choisir les cadeaux à livrer en fonction des demandes et de la disponibilité)
- L'envoi de réponses aux demandes, accompagnées de publicité ciblée en fonction des préférences des prospects

Comme il a suivi des cours de génie logiciel, le Père Noël a conçu cette application en MVC. Cette application comporte donc déjà des briques (POJOs) contenant le modèle nécessaire à son fonctionnement. Il vous a choisi(e) pour moderniser cette application et la rendre plus efficace dans la gestion des demandes de cadeaux et plus attractive pour les clients potentiels (sociétés de fabrication de jouets). Pour cela, vous devez mettre en place un site Web permettant (en plus de ce qui existe déjà) :

- De rechercher et d'accéder aux visualisations des cadeaux (pages des fabricants),
- D'enrichir les descriptions des cadeaux en utilisant d'autres sources de données disponibles sur le Web sous forme de données liées (Linked Data), de photos, etc.
- De saisir directement sa demande de cadeau dans l'application,
- D'accéder à la réponse à sa demande.

Comme il connaît les technologies à la mode, le Père Noël vous impose les contraintes suivantes :

- L'application doit être RESTful.
- L'application doit utiliser Spring Web MVC.
- Le formulaire de recherche doit offrir un système de suggestion automatique des noms de cadeaux.

Conception (barème : 5 points)

7. Listez les différents éléments nécessaires au fonctionnement de votre application, en indiquant leur nature : classes Java, pages statiques, scripts, servlets, JSP, beans (précisez le scope)…
   CECI EST UN EXEMPLE DE REPONSE – TOUT AUTRE STRUCTURE SENSEE EST TOUT AUSSI VALIDE.
   Côté serveur :
   Contrôleur principal Spring
   Contrôleurs de CU (servlets ou classes annotées acceptées) : produit, prospect, demande, réponse…
   Modèle (beans) : produit, prospect, demande, réponse… + suggestion / recherche de cadeau
   Vue (JSP pour HTML et XML, éventuellement classes de sérialisation pour JSON) : produit, prospect, demande, réponse… + pages statiques et CSS
   Autres : DAOs →BD, filtres pour l'authentification & autres aspects (logs…)
   Côté client :
   Affichage des pages transmises par le serveur
   Scripts : suggestion automatique, enrichissement des données en Linked Data…
   Bibliothèques JS directes : métier (jQuery), graphique (jQuery UI / Bootstrap / Foundation…)
8. Décrivez, à l'aide de diagrammes UML appropriés, leurs communications lorsqu'un utilisateur interagit avec le formulaire de recherche de cadeaux. Les éléments du SI externes à votre application seront modélisés comme des boîtes noires.
   Faire un diagramme de séquence ou de communication. Dans tous les cas, un diagramme dynamique, représentant :
   Loop :
   Action initiale de l'utilisateur : tape un caractère
   Côté client : Interception par un script →requête asynchrone d'autocomplétion
   Côté serveur : contrôleur Spring →contrôleur délégué →interrogation du modèle (cadeaux) →renvoie une liste d'id / noms de cadeaux →retour au contrôleur délégué →composition de la vue côté serveur (JSP / sérialisation JSON) →
   Côté client : callback JS →Affichage d'une popup contenant les noms et cadeaux et les liens en fonction des ids
   Fin Loop.
   Côté client : L'utilisateur clique sur un lien →requête synchrone
   Côté serveur : contrôleur Spring →contrôleur délégué →interrogation du modèle (cadeau) →renvoie les infos (nom, description…) du cadeau →retour au contrôleur délégué →composition de la vue côté serveur (JSP)
   Côté client : Affichage de la vue → onload : script d'enrichissement LD →requête(s) asynchrone(s) →callback : affichage des infos récupérées

Programmation (barème : 5 points)

9. Écrivez la fonction JavaScript de callBack qui affiche le pop-up contenant les informations reçues lors de la suggestion automatique (précisez les noms des ressources accédées côté serveur).

10. Écrivez la JSP qui représente la vue correspondant à un cadeau donné.