Développement d'Applications Réticulaires – Projet Partie 2

À réaliser en binômes.
À rendre lors d'une soutenance le 5 avril 2016.

Dans ce projet, vous devrez implanter un cadriciel d'applications Web jouet de A à Z, sans utiliser aucun composant existant. Le but pédagogique est de démystifier les piles de développement Web existantes. Il ne s'agit bien sûr pas de les concurrencer.

Le projet est découpé en cinq semaines, mais libre à vous de prendre de l'avance. Il utilise principalement les connaissances acquises lors de la première partie du cours. De nombreux points de ce sujet sont laissés libres, n'hésitez pas à demander conseil et à vous entraider (sans, bien entendu, partager de code entre les binômes).

Le langage d'implantation est au laissé choix (à valider par l'enseignant). Vous préférerez cependant un langage statiquement typé. Dans le cas ou vous opteriez quand même pour un langage dynamique, vous devrez produire une documentation (au moins aussi bonne que des déclarations de types dans des langages typés) pour tous les objets manipulés par votre code.

1 Semaine 1 : un serveur HTTP

Dans un premier temps, vous devrez implanter un serveur HTTP, en vous servant uniquement des primitives de communication TCP du langage choisi.

1. On veut pouvoir manipuler les requêtes et les réponses non pas sous leur forme texte brute, mais sous forme d'objets structurés du langage de programmation choisi (objet, record, structure, etc.).

   Commencez donc par définir un type pour les requêtes et un pour les réponses, qui peuvent partager des types annexes, par exemple pour les codes, les URLs et les en-têtes. Idéalement, vous définirez les types les plus précis possibles pour les en-têtes dont la structure est connue (cookies, etc.).

   Veillez à implanter tous les verbes et un maximum de codes, mais il serait probablement trop long de décrire précisément tous les en-têtes. Vous pouvez traiter ceux qui vous semblent les plus utiles pour la suite et utiliser un cas générique pour les autres (qui sera de toutes façons indispensable puisque HTTP autorise les en-têtes personnalisés).
2. Écrivez un analyseur syntaxique de requêtes (qui lit la requête au format textuel HTTP et produit un objet requête) et un imprimeur de réponses (qui prend un objet réponse et produit la réponse au format textuel HTTP). On cherche seulement à écrire un serveur, il n'est donc pas strictement nécessaire d'écrire les traitements inverses. Cela pourrait cependant vous être utile pour écrire vos jeux de test.
3. Écrivez un programme principal de serveur générique (paramétré par du code spécifique à l'application) qui écoute sur un port donné, et effectue la boucle suivante :
   - lire la requête et l'analyser syntaxiquement ;
   - passer l'objet requête obtenu au code spécifique, qui doit retourner un objet réponse ;
   - imprimer cette réponse au format texte et l'envoyer au client.

   Cela pourra prendre la forme d'une fonction d'ordre supérieur en fonctionnel, ou d'une classe abstraite em objet.
4. Pour tester votre serveur, vous devrez écrire un serveur d'écho qui attend les requêtes sur un port qui lui est donné via la ligne de commande, et qui reproduit la requête dans le corps de la réponse.

   Ce serveur devra répondre correctement aux demandes de `Content-type` pour les types MIME `text/plain`, `text/html` et `application/json`. Dans le premier cas, il devra simplement analyser syntaxiquement la requête puis la réimprimer, dans le second on attend une page HTML expliquant la structure de la requête de façon lisible pour un humain (par exemple avec un tableau et des explications), dans le troisième cas, on veut obtenir la structure et le contenu de la requête dans un format JSON judicieusement choisi.

Les formats exacts sont laissés libres, il n'y aura pas de correction automatique.

2 Semaine 2 : un routeur de requêtes

Une application Web non triviale est décomposée en services associés à différentes URLs. Il faut donc un mécanisme qui analyse l'URL de la requête pour orienter le flot de contrôle vers le code spécifique à chaque service. C'est ce qu'on appelle souvent routage, aiguillage ou dispatch.

Nous allons donc écrire une couche de routage, qui s'intercalera entre le serveur générique développé la semaine précédente et celui des différents services spécifiques de l'application.

1. Si vous ne l'aviez pas fait la semaine précédente, définissez un type pour les URLs, permettant d'en décomposer les éléments (chemin, arguments GET, fragment).
2. Concevez votre propre mécanisme d'aiguillage. N'hésitez-pas pas à vous inspirer de frameworks que vous connaissez. Un bon aiguilleur devrait :
   - L'aiguilleur discrimine sur un maximum de paramètres de la requête (verbe, chemin, paramètre, fragment, contenu du corps, `Content-type`, etc.) et que le corps des service ne fasse que le minimum de branchements initiaux.
   - Le maximum d'erreurs soit détectées le plus tôt possible (colisions de chemins entre services, paramètres manquants lors de l'appel de service, etc.).

- L'enregistrement de services soit facile et bien intégré au langage de programmation choisi (par exemple en utilisant la réflexion pour faire correspondre les noms et types des paramètres de la requête et ceux de la méthode implantant le service).
- L'aiguillage soit rapide même avec un grand nombre de services.

Il faut trouver un bon équilibre et ne pas être trop ambitieux pour réussir à obtenir quelque chose de fonctionnel dans le temps imparti.

Prenez le temps de réfléchir et de faire valider vos choix.

3. Implantez une application Web jouet de stockage de points en 2D (une table globale en mémoire partagée dans le serveur) répondant aux requêtes suivantes :

`GET /list`	Renvoie la liste des `id` des points existants
`GET /p/<id>/x`	Renvoie la coordonnée *x* du point `id`
`GET /p/<id>/y`	Renvoie la coordonnée *y* du point `id`
`PUT /p/<id>?x=<value>&y=<value>`	Modifie 0, 1 ou 2 *x*,*y* passés en GET
`POST /p`	Nouveau point, *x*,*y* passés dans le corps, retourne l'`id`
`DELETE /p/<id>`	Supprime un point

N'hésitez pas à en ajouter pour tester plus votre aiguilleur. Vous pouvez ajouter des URLs, des variantes selon de `Content-type`, etc.

3 Semaine 3 : un gestionnaire de sessions

Pour écrire des applications Web, il faut pouvoir reconnaître que deux requêtes proviennent du même client. Cela se fait classiquement avec un mécanisme de sessions.

1. Le mécanisme de base est simple : lors de la réception d'une requête, si le client envoie un cookie de session, on le réutilise. Sinon, on en crée un frais et on le lui envoie. Usuellement, ce cookie est le hachage d'un condensat de paramètres constants pour un utilisateur (IP, User Agent, etc.), permettant lors de la réception d'un cookie de session par le serveur de vérifier qu'il ne s'agit pas d'une usurpation. Vous pouvez commencer par une version plus basique, non sécurisée.
2. L'intérêt d'un tel mécanisme est de partager un état entre les différents appels aux différents services. Améliorez votre aiguilleur de la semaine précédente pour que le serveur stocke un objet état pour chaque session (dont le type est commun à toute l'application mais personnalisable pour chaque application). Le code de chaque service devra recevoir cet objet en paramètre supplémentaire. La table de sessions est usuellement stockée sur disque, mais vous pourrez la stocker en mémoire pour simplifier.
3. Ajoutez une gestion de la durée de vie des sessions.
4. Écrivez une micro application pour tester cette nouvelle fonctionnalité. Par exemple, une gestion d'inscription et d'authentification d'utilisateurs.

4 Semaine 4 : une systèmes de patrons

Les patrons (templates) sont un mécanisme classique pour fabriquer des pages par morceaux. Un bon mécanisme de patrons doit être à la fois simple à utiliser pour le concepteur des patrons et celui qui les utilise. La syntaxe doit donc être simple, et l'intégration avec le langage de programmation choisi maximale.

1. Concevez votre langage de patrons et écrivez l'analyseur syntaxique. N'hésitez-pas à vous inspirer de langages existants, et restez sur des choses simples. Il vaut mieux concevoir un système très simple mais très bien intégré au langage qu'un système puissant mais difficile à utiliser. Faites valider vos choix avant de commencer à implanter.
2. Trouvez une façon d'intégrer au mieux ces patrons au langage. Dans un langage à objets, ou pourra par exemple utiliser la réflexion pour instancier un patron avec les champs d'un objet. Dans un langage fonctionnel, on pourra par exemple transformer simplement un patron en fonction dont les paramètres sont les trous du patron. Vous êtes libres de choisir si les patrons doivent être lus à la compilation (ce qui permet par exemple de générer du code ad-hoc) ou l'exécution (ce qui permet de changer les patrons sans recompiler).

5 Semaine 5 : un petit exemple

Utilisez les briques développées précédemment pour implanter une micro application Web de votre choix. Points bonus si certaines fonctionnalités de votre application sont utilisables sous forme de services JSON / JSONP, si votre application utilise des XHR, et/ou si vous utilisez des fonctionnalités HTML5 avancées.

Un exemple pour donner la difficulté attendue serait une application de notification de présence, ou les utilisateurs peuvent s'identifier pour indiquer s'ils sont disponibles ou non, avec une page publique indiquant la disponibilité de tout le monde. On peut imaginer de nombreuses petites améliorations comme l'obtention des états en JSON, ou la mise à jour de la page en temps réel par XHR.