TD Web / XML

ENSEIRB

Programmation Web / XML

TD : Client Serveur, XML, XSD, XPath, XSLT

Version 6.00 du 1er octobre 2014

Etat : Travail

SOPRA GROUP

Historique :

Version	Date	Origine de la mise à jour	Rédigée par	Validée par
1.0	07/09/2009	Création	Mathieu Lombard	
2.0	30/11/2010	Ajustements pour introduire le XSLT (2ème séance) et le XPATH (3ème séance) ainsi que l'introduction des schémas en fin de chaque séance	Mathieu Lombard	
3.0	02/11/2011	Élargissement de la partie client-serveur pour couvrir XML, XSD, XPath et XSLT	Mathieu Lombard	
4.0	05/11/2012	Corrections mineures	Mathieu Lombard	
5.0	11/10/2013	Ajout d'informations sur les objets JAVA manipulés	Mathieu Lombard	
5.1	08/11/2013	Ajout d'un schéma global d'architecture et d'explications pour différencier client et serveur	Mathieu Lombard	
6.0	01/10/2014	Ajustements divers	Mathieu Lombard	

Préambule :

Le but de ce TD est de développer un serveur Web capable de répondre aux demandes de page Web d'un navigateur et gérant les requêtes AJAX.

En interne, le serveur s'appuiera sur XML, XSD, XPath et XSLT pour pouvoir afficher les informations demandées

L'exemple fonctionnel sur lequel s'appuiera le TD est la gestion de tâches que l'on peut décrire et affecter à des utilisateurs.

Ce TD suivra les grandes étapes suivantes :

- Mise en place d'une application web fonctionnant sur le serveur Jetty
  - Le développement s'effectuera en JAVA
  - Dans un premier temps, le serveur ne renverra que du texte et du HTML
- Extension du serveur vers le XML, le serveur sera alors capable de renvoyer du XML
- Écriture de schéma XSD pour indiquer les interfaces client-serveur
- Ajout de « requêtage » XPath en interne du serveur pour récupérer des valeurs depuis le XML
- Utilisation de XSLT pour afficher des pages selon des gabarits prédéfinis et le XML interne au serveur
- Évaluation 1
- Évaluation 2

Ce TD s'étendra sur 7 sessions qui jalonneront les étapes permettant d'aboutir à l'application Web "monde réel" ; les 2 dernières sessions seront consacrées à l'évaluation.

À la fin de chaque session, une « prise de recul » sera effectuée afin de s'assurer de la bonne compréhension de « ce qui vient d'être fait » ; cela durera en général 20 minutes et s'appuiera essentiellement sur des schémas explicitant les mécanismes utilisés durant la séance.

Sommaire :

0. INTRODUCTION

0.1. Le serveur et la WebApp

Le serveur sur lequel s'appuiera le TD est Jetty (c'est un serveur HTTP de Servlets léger et suffisant pour les besoins du TD).

Tous les développements liés aux TDs s'effectueront dans une « Web Application » (WebApp) que Jetty hébergera ; le langage utilisé sera Java.

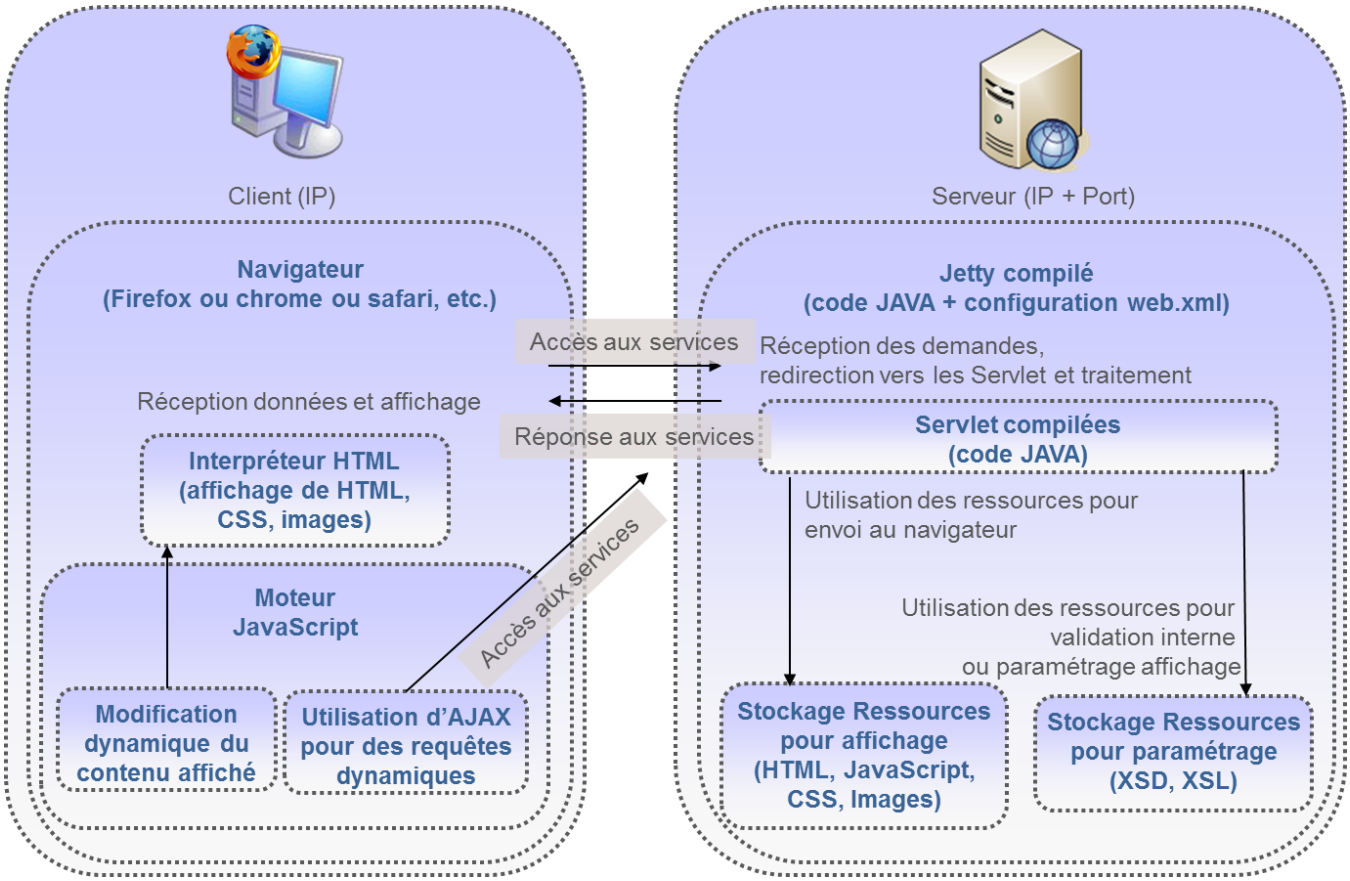
Pour que Jetty déploie une WebApp (ce qui revient à l'exécuter dans une certaine mesure, du moins permettre à un client d'y accéder via ses Servlets), il faut que celle-ci soit contenue dans le répertoire webapp de Jetty.

Ainsi, l'arborescence fournie en entrée de ce TD contient à la fois :

- L'environnement d'exécution de Jetty
- L'arborescence correspondant à la WebApp qui fait l'objet du TD (contenue sous webapp/EnseirbWebXMLWebApp)

Le code de base de l'application se trouve sur une page indiquée en TD. Il faut le dé-zipper à un emplacement ne comportant pas d'espace dans le chemin.

0.2. La vision d'ensemble – schéma d'architecture »



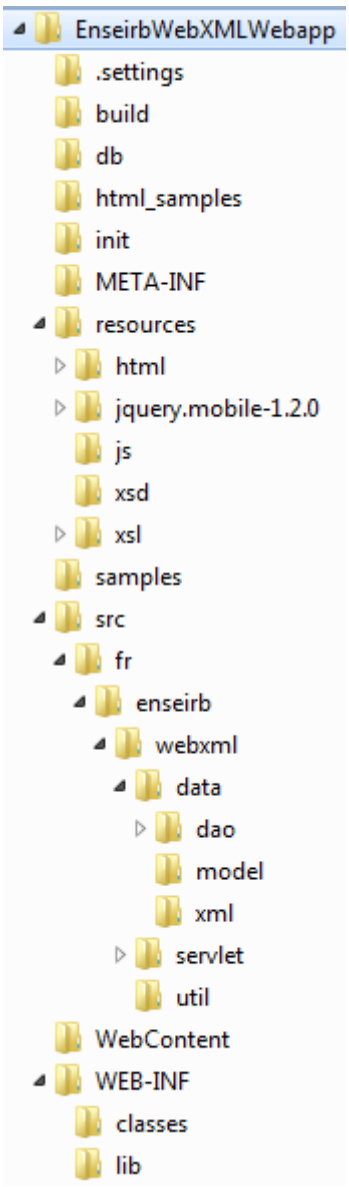
Tous les développements concernés par ce TD seront effectués côté serveur (classes JAVA, HTML, JavaScript, XSD, XML, XSL). Le serveur s'appuiera sur ces ressources pour rendre les services demandés par le client.

Le client (ici le navigateur) servira pour vérifier que les services rendent bien ce qui est attendu. Le navigateur recevra alors des données et pourra les interpréter (cas du HTML) ou les afficher (cas du XML).

Note : dans le cas du HTML, le navigateur affiche les données et exécute le JavaScript. Ainsi, le JavaScript n'est que stocké côté serveur et n'est vraiment interprété que côté client.

0.3. L'arborescence de la WebApp

L'arborescence du projet (de la WebApp) est la suivante :



Elle se décline comme suit :

- /src, contient les sources JAVA du serveur

- le chemin *fr.enseirb.webxml* correspond au package de base
- .data correspond à la gestion des données (stockage dans la base de données et manipulation XML de ces données)
- .servelt contient toutes les Servlets de la WebApp
- .util contient des classes JAVA facilitant les développements liés aux TDs ; **ce répertoire sera très utile tout au long des TDs**

- /db
  - Contient la base de données que stocke le serveur (ce répertoire est vide en début de fonctionnement du serveur et se remplit au fur et à mesure des actions utilisateurs)
- /init, contient une base de données permettant d'initialiser les données du projet
  - Cela permet d'avoir des données « réelles » afin que le serveur puisse renvoyer « quelque chose » quand il est interrogé
- /resources, contient les fichiers autres que JAVA
  - ce répertoire contiendra les fichiers .xml, .xsl, .xsd, .html et .js utilisés par le serveur
- /WEB-INF, contient le fichier web.xml qui décrit les points d'entrée du serveur (les Servlets)
- /WEB-INF/classes, contient les classes compilées pouvant être exécutées
- /WEB-INF/lib, contient les librairies JAVA dont dépend la WebApp

Les ressources à versionner sous SVN sont le répertoire /src/*, /resources/*, /WEB-INF/web.xml

0.4. La compilation et l'exécution

Le serveur Jetty n'a pas besoin d'être compilé (il l'est déjà). En revanche, la WebApp doit l'être à chaque fois que des sources Java changent.

Pour compiler **la WebApp, il faut se placer à sa racine** et lancer l'instruction suivante :

```
javac -sourcepath src -encoding cp1252 -d WEB-INF\classes -cp "WEB-INF\lib\*;..\..\lib\*" src\fr\enseirb\webxml\servlet\*.java
```

Dans un environnement Unix/Linux (et non Windows), il faut remplacer les « \ » par des « / » et le « ; » par « : ».

Pour exécuter **le serveur, il faut se placer à sa racine** et lancer l'instruction suivante :

```
java -jar start.jar
```

0.5. L'application de test

Une application permettra de tester automatiquement la réalisation des élèves.

Hypothèses de l'application de test :

- Les URLs auxquelles réagit cette WebApp sont strictement identiques à ce qui était demandé dans les TDs (une note récapitulative est fournie en annexe).

Pour utiliser l'application de test, il faut tout d'abord la décompresser (le .jar et le répertoire data ; tous les 2 dans un chemin ne comportant pas d'espace) puis lancer, à la racine, la commande suivante :

```
java -jar EnseirbWebXMLEval1.jar <webAppName> <serverPort> <resultDirPath>
```

- <webAppName> correspond au nom de la WebApp, à priori « EnseirbWebXMLWebApp »

- <serverPort> correspond au port du serveur, par exemple « 8080 »
- <resultDirPath> correspond à un chemin vers un répertoire dans lequel un fichier résultat sera écrit (ce répertoire doit exister au préalable).

Il est à noter que l'URL utilisée pour accéder au serveur sera « localhost ».

Par ailleurs, les noms des élèves d'un binôme sont récupérés automatiquement via l'URL */about*, il faut donc bien s'assurer que celle-ci est opérationnelle (sinon, le fichier de résultat ne permettra pas d'identifier les élèves et cela sera considéré comme une absence de fichier).

1. PREMIÈRES SERVLETS

1.1. Prise en main

La WebApp fournie contient déjà une Servlet fonctionnelle : la page d'accueil.

Pour pouvoir y accéder, chaque groupe doit choisir un numéro de port pour son propre serveur et l'indiquer dans le fichier ./etc/jetty.xml avec la ligne suivante (cette ligne existe déjà, il suffit de **modifier** le <port>) :

```
<Set name="port"><Property name="jetty.port" default="<port>"/></Set>
```

Pour accéder à la page d'accueil, il suffit alors de démarrer le serveur et de saisir l'URL suivante dans son navigateur : http://localhost:<port>/EnseirbWebXMLWebapp

Le navigateur devrait afficher une page permettant d'accéder à toutes les tâches, à tous les utilisateurs et aux statistiques ; pour le moment, ces liens ne fonctionnent pas et ils seront opérants à la fin de toutes les sessions de TD.

Note : la structure de la page HTML (le fichier *resources/html/common/welcome.html*) est représentative de toutes les pages que devra renvoyer la WebApp ; elle pourra donc servir de base.

1.2. La Servlet « About »

Afin d'identifier chaque groupe, la Servlet « About » permet de renvoyer les informations concernant les élèves développant la WebApp.

Cette Servlet se trouve à l'endroit suivant : *fr.enseirb.webxml.servlet.AboutServlet*.

Elle doit être modifiée pour renvoyer :

- Le nombre d'élèves composant le groupe
- Le nom/prénom de chaque élève
- La classe et le groupe de TD

1.2.1. Renvoi des noms/prénoms du groupe

Par défaut, la Servlet renverra une chaîne de caractères contenant les noms/prénoms des élèves composant le groupe de TD.

Instructions :

- Modifier la chaîne de caractère renvoyée par le Servlet
- Compiler le code modifié pour mettre à jour la WebApp
- Exécuter le serveur
- Tester avec un navigateur
  - Saisir l'adresse http://localhost:<port>/EnseirbWebXMLWebapp/about

1.2.2. Prise en compte de paramètres pour renvoyer une information précise

La Servlet « *About* » ne renvoie qu'une seule information, et il faut qu'elle permette au client d'avoir accès aux informations suivantes, en fonction d'un paramètre indiqué (le paramètre de nom « action » sera utilisé pour indiquer l'objet de la demande dans l'URL) :

Valeur du paramètre « action »	Donnée devant être renvoyée

studentsNumber	Le nombre d'élève composant le groupe
studentId=<id> avec firstName	Le prénom de l'élève numéro <id> dans le groupe (<id> peut valoir entre 1 et « studentsNumber »)
studentId=<id> avec lastName	Le nom de famille de l'élève numéro <id> dans le groupe
class	La classe des élèves
group	Le nom du groupe de TD (G1, G2, …)
full (ou pas de paramètre)	La chaîne de caractères suivante : <classe> / <groupe de TD> / <prénom élève 1> <nom élève 1> / <prénom élève 2> <nom élève 2>

Instructions :

- Enrichir la Servlet pour qu'elle traite les cas décrits ci-dessus
  - La méthode *parseURLParams* de la classe *ServletToolkit* (du package *util*) permet de récupérer les paramètres de l'URL ; cela, sous la forme d'un objet *Properties* du pacakge Java *java.util*
  - Ensuite, il suffit d'utiliser les méthodes suivantes
    - *containsKey*, pour savoir si un paramètre est présent dans l'URL
    - *getProperty*, pour récupérer la valeur d'un paramètre
- Compiler / exécuter
- Tester avec un navigateur ; l'URL aura la forme suivante
  - http://localhost:<port>/EnseirbWebXMLWebapp/about?<nom_param>=<valeur_param>&<nom_param2>=<valeur_param2>
  - Pour accéder au prénom du premier élève, l'URL sera
  - http://localhost:<port>/EnseirbWebXMLWebapp/about?action=firstName&studentId=1

1.2.3. Rajout de l'accès à la page « about » depuis la page d'accueil

Instructions :

- Modifier le fichier *resources/html/common/welcome.html* pour permettre l'accès à la page « *about* » depuis la page d'accueil
  - Il suffit de s'inspirer du code déjà existant en rajoutant un bloc « <*li*> »
- Exécuter
- Tester en accédant à la page d'accueil et en cliquant sur le nouveau lien

1.3. Envoie de données en POST via AJAX

Pour pouvoir envoyer des données, il y a 2 étapes à suivre :

- Permettre à l'utilisateur de saisir les données
- Permettre à l'utilisateur de les envoyer à proprement dit

Une fois ces données envoyées au serveur, l'utilisateur devra aussi pouvoir les visualiser.

1.3.1. Formulaire de saisie du nom du professeur

Un fichier HTML est déjà créé dans *resources/html/common/about_teacher.html* ; il offre la possibilité de saisir du texte.

Instructions :

- Modifier la Servlet *AboutServelt* pour qu'elle renvoie ce fichier dans le cas de l'URL *about/teacher*
  - Pour savoir si l'URL contient un certain texte, il suffit d'utiliser le code suivant

```
if (request.getRequestURI().contains("about/teacher"))
```

  - Pour renvoyer un fichier HTML, reprendre le code de *DefaultServlet*
- Compiler puis Exécuter
- Tester en accédant à la page
  - http://localhost:<port>/EnseirbWebXMLWebapp/about/teacher

1.3.2. Envoi des données en POST en utilisant AJAX

Le fichier *about_teacher.html* précédent exécute une méthode *postTeacher(<nom enseignant>)* quand l'utilisateur clique sur « Stocker » ; il s'agit ici de créer cette fonction.

Un fichier *toolkit.js* permet de gérer l'objet XMLHttpRequest et ainsi facilite l'envoi de données en AJAX. De plus, le support de cours contient le code à ajouter ici.

Instructions :

- Ecrire un fichier JavaScript qui contient une fonction *postTeacher* qui prend un paramètre en entrée correspondant au nom de l'enseignant
  - La méthode doit envoyer les données au serveur selon l'URL *about/teacher/post* et selon la commande HTTP POST
  - Le fichier JavaScript devra être placé dans *resources/js*
  - Le fichier JavaScript devra être référencé depuis le fichier *about_teacher.html* comme l'est le fichier *toolkit.js*
- Modifier *AboutServlet* pour qu'elle traite l'URL *about/teacher/post* et qu'elle stocke ce qui est envoyé par le POST dans une variable de classe
  - Pour accéder au contenu du POST, il suffit d'utiliser la méthode suivante (la classe *ServletToolkit* se trouve dans le package *fr.enseirb.webxml.util* **; il faut penser à l'importer en haut de la classe de Servlet**)

```
String postData = ServletToolkit.getPostData(request);
```

  - Attention à l'ordre des traitements d'URL (*about/teacher/post* doit être traité avant *about/teacher)*
- Compiler et exécuter
- Tester en cliquant sur « Stocker » après avoir saisi le nom de l'enseignant

1.3.3. Visualisation des données envoyées

Afin d'accéder au nom de l'enseignant préalablement saisi, il faut rajouter un cas dans *AboutServlet* pour qu'elle renvoie ce nom dans le cas de l'URL *about ?action=teacher*

Instructions :

- Modifier *AboutServlet* pour traiter ce cas de paramètre supplémentaire
- Compiler puis Exécuter
- Tester en accédant à la page

- http://localhost:<port>/EnseirbWebXMLWebapp/about?action=teacher

1.4. « Prise de recul »

Proposer un schéma d'architecture qui

- Identifie les composants du système en explicitant qui est « client » / « serveur » de qui
- Indique les flux d'échange, leurs sens et le format d'échange de données
  - Dans le cas du GET
  - Et dans le cas du POST via AJAX

2. UTILISATION DE XML

Le but est ici de gérer les données en entrée et en sortie du serveur au format XML.

Des nouvelles *Servlet* vont être créées pour donner de nouveaux points d'entrée au serveur.

Les objets JAVA *User* et *Task* vont être manipulés ; ils sont tous les 2 du package *fr.enseirb.webxml.data.model*. Ils permettent de stocker (méthodes *setXXX*) et de récupérer (méthodes *getXXX*) des valeurs les caractérisant. Leurs propriétés correspondent aux attributs qui transiteront à travers les flux XML.

2.1. Accès à la liste des utilisateurs en XML

Instructions :

- Créer une nouvelle *Servlet* de nom *ListUsersServlet* et la référencer dans le *web.xml* de la WebApp
  - Pour la *Servlet*, il suffit de s'inspirer des autres Servlet déjà créées
  - Pour le fichier *web.xml*, il suffit de s'inspirer de ce qui est fait pour les autres *Servlets* et de bien vérifier la syntaxe XML
  - La *Servlet* devra être accédée sous l'URL : *user/list/xml*
- La Servlet devra s'appuyer sur la méthode *getUsers()* de la classe *XMLMediator* (du package *fr.enseirb.webxml.data.xml*) pour renvoyer le flux XML
  - Compléter la méthode *getUsers* pour qu'elle renvoie un flux XML de la forme suivante

```
<users>
   <user name="xxx"/>
   <user name="yyy"/>
   ...
</users>
```

  - L'invoquer depuis la *Servlet* et indiquer au client que la réponse est au format XML (et non au format HTML), le code à utiliser est le suivant :

```
response.setHeader("Content-Type", "application/xml");
```

  - Compiler, exécuter et tester

2.2. Accès à la liste des tâches en XML

Le mécanisme à mettre en place est le même que précédemment pour les utilisateurs en renvoyant cette fois la liste des tâches et avec les différences suivantes :

- La *Servlet* pourra s'appeler *ListTasksServlet*
- L'URL devra être *task/list/xml*
- La méthode à compléter dans *XMLMediator* est *getTasks()* et elle doit renvoyer un flux de la forme suivante

```
<tasks>
   <task id="xxx" title="ttt" deadline="dd/MM/yyyy" priority="123" done="true"
creationDate="dd/MM/yyyy">
      <description>ddd</description>
      <asker>aaa</asker>
      <owner>ooo</owner>
   </task>
   <task id="yyy" title="ttt" deadline="dd/MM/yyyy" priority="123" done="false"
creationDate="dd/MM/yyyy">
      <description>ddd</description>
      <asker>aaa</asker>
      <owner>ooo</owner>
   </task>
   ...
</tasks>
```

2.3. Accès à une tâche particulière en XML

Il s'agit ici d'avoir le flux XML correspondant à 1 seule tâche ; celle-ci sera repérée par son id qui sera passée dans l'URL.

Il faut alors :

- Modifier *ListTasksServlet* pour qu'elle gère le cas où le paramètre « id » est passé dans l'URL
- Compléter la méthode *getTask(int)* de *XMLMediator* pour qu'elle renvoie le flux XML correspondant à une tâche

2.4. Ajout d'un utilisateur via l'URL puis XML

L'objectif est de pouvoir ajouter un nouvel utilisateur en passant son nom via l'URL puis de faire transiter les données en XML en interne du serveur.

Instructions

- Rajouter une nouvelle *Servlet* appelée par exemple *CreateUserServlet* qui correspondra à l'URL */user/create/url*
- Gérer la prise en compte du paramètre *name* pour créer un nouvel utilisateur
- Compléter la méthode *addUser(Element)* de *XMLMediator*
- Utiliser la méthode *addUser(String)* (qui appelle la précédente ou directement la méthode précédente selon l'implémentation choisie) pour ajouter un utilisateur à partir d'un flux XML qui aura été construit au préalable dans *CreateUserServlet* à partir du paramètre d'URL

Pour tester, il suffit d'invoquer les URLS suivantes :

- http://localhost:<port>/EnseirbWebXMLWebapp/user/create/url?name=<name> pour créer un nouvel utilisateur
- http://localhost:<port>/EnseirbWebXMLWebapp/user/list/xml pour vérifier que le nouvel utilisateur est bien pris en compte

2.5. Ajout d'une tâche via l'URL puis XML

Le mécanisme à mettre en place est le même que précédemment, mais cette fois en ajoutant une tâche et non un utilisateur et avec les différences suivantes :

- La Servlet pourra s'appeler *CRUDTaskServlet*
- Elle devra être accessible via l'URL *task/create/url*
- Les paramètres devront être les suivants
  - *id*, l'identifiant unique de la tâche (cela peut être utile quand on souhaite modifier une tâche et non l'ajouter)
  - *title*, le titre de la tâche
  - *description*, une description de la tâche
  - *deadline*, la date d'échéance de la tâche, au format jj/MM/aaaa
  - *priority*, une priorité (un entier compris entre 1 et 5) pour la tâche
  - *done*, l'état de la tâche (booléen)
  - *asker*, l'émetteur de la tâche
  - *owner*, le responsable de la tâche
- La méthode à compléter dans *XMLMediator* est *addOrModifyTask(Element)* et il faut l'appeler (ou appeler la méthode *addOrModifyTask(String)* selon l'implémentation choisie) pour stocker la nouvelle tâche

Pour tester, il suffit d'invoquer les URLS suivantes :

- http://localhost:<port>/EnseirbWebXMLWebapp/task/create/url?title=<title>&description=<desc>&deadline=10/11/2011&priority=1&asker=<asker>&owner=<owner> pour créer une nouvelle tâche
- http://localhost:<port>/EnseirbWebXMLWebapp/task/list/xml pour vérifier que la nouvelle tâche est bien prise en compte

2.6. « Prise de recul »

Pour toutes les utilisations faites de XML (configuration du serveur et gestion des données), indiquez :

- Qui produit la donnée et quand cela arrive
- Qui consomme la donnée et quand cela arrive

3. UTILISATION DE XSD

L'objectif est de définir les interfaces qu'offre le serveur :

- Le format des données qu'il renvoie
- Le format des données qu'il est capable d'interpréter

3.1. Définition du schéma XML pour la liste des tâches

Instructions :

- Créer le XSD qui décrit une liste de tâches (cf. XML exemple plus haut) et placer le dans */resources/xsd*
  - Note : le type pour les dates peut rester « string » et non « date » (afin de simplifier l'écriture du XSD)
  - Note : il faut placer les éléments « sequence » avant les éléments « attribute » dans le XSD
- Utiliser ce schéma pour vérifier que les données en retour de la méthode *getTasks()* de *XMLMediator* sont bien formées (le code est à rajouter à la fin de la méthode) ; pour cela utiliser le code suivant

```
XMLToolkit.isXMLValid(fluxXML, pathXSD)
```

- Pour tester, rejouer le test accédant à la Servlet listant toutes les tâches

3.2. Définition du schéma XML pour la liste des utilisateurs

Le mécanisme à mettre en place est le même que précédemment, avec les différences suivantes :

- Le schéma devra décrire la liste des utilisateurs (cf. XML exemple plus haut)
- La méthode à enrichir est *getUsers()* de *XMLMediator*
- Pour tester, il faut accéder à la Servlet listant les utilisateurs

3.3. Définition du schéma XML pour une tâche

Instructions :

- Créer le XSD qui décrit une seule tâche
- Utiliser ce schéma pour vérifier que les données en entrée de la méthode *addOrModifyTask(XMLTask)* de *XMLMediator* sont valides (le code est à rajouter au début de la méthode)
- Pour tester, accéder à la Servlet de création d'une nouvelle tâche via les paramètres dans l'URL

3.4. Définition du schéma XML pour un utilisateur

Le mécanisme à mettre en place est le même que précédemment, avec les différences suivantes :

- Le schéma devra décrire un utilisateur
- La méthode à enrichir est *addUser(XMLUser)* de *XMLMediator*
- Pour tester, il faut accéder à la Servlet de création d'un utilisateur via les paramètres d'URL

3.5. « Prise de recul »

Schématisez les flux d'entrée / sortie impliquant les schémas XSD et indiquez à quel moment la validation intervient.

Précisez les cas où cette validation vous paraît opportune et proposez des nouveaux cas où celle-ci serait utile.

4. UTILISATION DE XPATH

L'objectif est de bâtir un flux XML de statistiques en s'appuyant sur les flux XML des tâches et des utilisateurs.

4.1. Création de la Servlet de statistiques

Instructions :

- Rajouter une nouvelle servlet, par exemple de nom *StatsServlet*, dont l'URL d'accès sera */stats/xml*
- La faire renvoyer ce que renvoie la méthode *buildStats()* de *XMLMediator* en modifiant la chaîne de caractère renvoyée pour pouvoir tester
- Compiler, exécuter et tester

4.2. Création d'un flux XML de statistiques à partir de requêtes XPath

Le flux devant être construit doit être de la forme suivante :

```
<stats>
  <user name="uuu">
    <tasks number="2">
      <done number="1">
        <task title="ttt1"></task>
      </done>
      <todo number="1">
        <late number="1">
          <task title="ttt2"></task>
        </late>
        <intime number="0"></intime>
      </todo>
    </tasks>
  </user>
  <user name="uuu2">
    <tasks number="2">
      <done number="0"></done>
      <todo number="2">
        <late number="0"></late>
        <intime number="2">
          <task title="ttt3"></task>
          <task title="ttt4"></task>
        </intime>
      </todo>
    </tasks>
  </user>
  ...
</stats>
```

Pour pouvoir le construire, il faut parcourir la liste des utilisateurs, puis, pour chacun d'entre eux, il faut comptabiliser le nombre de tâches

- Qu'il a déjà fait (done)
- Qu'il doit encore faire (todo) et parmi celles-ci
  - Celles pour lesquelles il est en retard (late)
  - Et celles pour lesquelles il n'est pas en retard (intime)

Pour chaque tâche, le titre doit être inclus.

Les méthodes utiles sont les suivantes :

- Pour accéder à une liste de nœuds répondant à une requête XPath

```
XMLToolkit.getXPathValues(fluxXML, requêteXPath)
```

- Pour accéder à une valeur répondant à une requête XPath

```
XMLToolkit.getXPathValue(fluxXML, requêteXPath)
```

Par ailleurs, un exemple de code est déjà présent dans la méthode *buildStats* pour comparer les dates afin de savoir si une tâche est en retard ou non.

Instructions :

- Compléter la méthode *buildStats()* de *XMLMediator*
  - En parcourant le flux XML des Users afin de tous les récupérer
  - En parcourant le flux XML des Tasks afin d'effectuer les 3 requêtes XPath pour connaître les statistiques de chaque utilisateur qui correspondent aux critères énoncés ci-dessus (faites, non faites en retard, non faites non en retard)
  - En rassemblant toutes ces informations dans un flux XML à renvoyer
- Compiler, exécuter et tester

4.3. « Prise de recul »

Ébauchez la construction du même document XML en utilisant les méthodes de l'API DOM.

Indiquez les avantages / inconvénients de chaque méthode.

5. UTILISATION DE XSLT

L'objectif de ce TD est de permettre de visualiser les données que renvoie le serveur au format HTML.

Ainsi, en utilisant une transformation XSL, les données actuellement renvoyées en XML pourront être formatées et renvoyées au format HTML.

5.1. Affichage de la page pour créer un utilisateur

Le fichier XSL permettant de créer un utilisateur via un formulaire HTML est fourni dans */resources/xsl/common* et s'appelle *create_user.xsl*.

Instructions :

- Compléter la Servlet *CreateUserServlet* (et le web.xml) pour
  - Qu'elle réagisse à l'URL */user/create* (la Servlet *CreateUserServlet* devra donc être désormais capable de traiter 2 types de demande, l'ancienne */user/create/url* et la nouvelle */user/create* et d'avoir le comportement dédié à chaque cas)
  - Qu'elle renvoie le HTML construit à partir de la feuille XSL *create_user.xsl* et d'un flux XML quelconque (par exemple « <noXML/> ») ; le code permettant d'appliquer une transformation XSL à un flux XML et de passer des paramètres est le suivant

```
Map<String, String> xslParams = new HashMap<String, String>();
xslParams.put("pageTitle", "Ajout utilisateur");
xslParams.put("htmlTitle", "Utilisateur à ajouter");
XMLToolkit.transformXML(fluxXML, XSLPath, xslParams);
```

  - Compiler, exécuter et tester via l'URL
    - http://localhost:<port>/EnseirbWebXMLWebapp/user/create
    - Le test à effectuer consiste uniquement à visualiser la page sans pouvoir créer l'utilisateur à proprement dit

5.2. Affichage de la page pour créer une nouvelle tâche

Le mécanisme est similaire au précédent, aux différences près suivantes :

- Le fichier XSL s'appelle *crud_task.xsl*
- La Servlet à compléter est CRUDTaskServlet (si vous l'avez nommée ainsi)
- C'est la méthode *getTask(idTask)* de *XMLMediator* qui permet d'obtenir la liste des tâches au format XML et dans le cas d'une création, la méthode *createEmptyTaskXML* de *XMLMediator* pourra être utilisée pour simuler une tâche à fusionner avec le XSL
- La feuille XSL attend un paramètre supplémentaire *pageState* qui indique si les données affichées sont modifiables ou non
  - L'URL d'accès sera */task/create* pour la création d'une nouvelle tâche *(pageState = « CREATE »)*
  - L'URL d'accès sera */task/modify* pour la création d'une nouvelle tâche *(pageState = « MODIFY »)* ; dans ce cas, le paramètre *id* sera utilisé pour savoir quelle est la tâche à modifier
  - L'URL d'accès sera */task/view* pour la création d'une nouvelle tâche *(pageState = « READONLY »)* ; dans ce cas, le paramètre *id* sera utilisé pour savoir quelle est la tâche à visualiser
- Compiler, exécuter et tester via l'URL

- http://localhost:<port>/EnseirbWebXMLWebapp/task/create (modify?id=XXX et view?id=YYY)
- Le test à effectuer consiste uniquement à visualiser la page sans pouvoir créer ou modifier la tâche à proprement dit ; le libellé du bouton en fin de formulaire permet de savoir si le paramètre *pageState* est passé correctement

5.3. Affichage de la liste des utilisateurs

Instructions :

- En se basant sur l'exemple HTML *user_list.htm* (fourni dans l'arborescence du projet sous *html_samples*), extraire une feuille de style XSL qui permet de rendre dynamique la liste des utilisateurs (l'affichage et les liens) ; la feuille sera placée dans *resources/xsl*
  - La transformation prendra en entrée la liste des utilisateurs au format XML (réalisée dans les précédentes TDs)
  - La feuille itérera sur les utilisateurs pour construire le HTML à renvoyer
  - Les utilisateurs seront triés selon leur nom
- Compléter la Servelt de nom *ListUsersServlet* (si c'est bien le nom que vous aviez utilisé), qui correspondra à l'URL */user/list*
- Pour tester
  - Invoquer directement l'URL */user/list*
  - Se placer sur la page d'accueil et cliquer sur « Tous les utilisateurs »

5.4. Affichage de la liste des tâches

Le mécanisme est similaire à ce qui a été fait juste avant, aux différences suivantes :

- Il y a 2 fichiers d'exemple :
  - *task_list.htm* pour l'affichage de toutes les tâches
  - *user_task_list.htm* pour l'affichage des tâches d'un utilisateur précis
    - l'utilisateur sera fourni par le paramètre *ownerNameFilter*
- Le flux XML de base sera celui de la liste des tâches au format XML déjà fabriqué dans les TDs précédents
- Les tâches seront triées selon leur date de deadline puis selon leur priorité
  - Pour trier selon la date, il faudra trier d'abord selon l'année, puis le mois et enfin le jour ; pour obtenir l'année à partir de la date, s'appuyer sur le code suivant

```
substring(@deadline,7,4)
```

  - L'URL d'appel doit être */task/list*
  - Si une tâche est terminée, elle devra être rayée, le style doit être le suivant :

```
text-decoration:line-through
```

  - Pour tester
    - Invoquer directement l'URL /task/list
    - Se placer sur la page d'accueil et cliquer sur « Toutes les tâches »
    - Se placer sur la page listant les utilisateurs et cliquer sur l'un d'entre eux
    - Accéder à la page de modification / visualisation d'une tâche en cliquant sur les icônes dédiés sur la page qui liste les tâches

5.5. « Prise de recul »

Dessinez un diagramme de séquence indiquant

- Les étapes permettant d'obtenir une page HTML à partir de données XML et d'une transformation XSL

- La navigation entre les pages ; depuis la page d'accueil et vers toutes les pages HTML que peut renvoyer le serveur

6. ANNEXE

6.1. Hypothèses de comportement de l'application

Une série de tests automatiques permet de vérifier la bonne implémentation des 5 TDs précédents. Notamment, les éléments suivants vont être vérifiés :

- XSD
  - Tâche
    - Les attributs « title », « deadline », « priority » et « done » sont obligatoires
    - Les attributs « id » et « creationDate » sont facultatifs (**note importante** : ces 2 attributs doivent être renvoyés dans le cas d'une lecture des tâches, mais ne sont pas obligatoires dans le cas d'une écriture car c'est la base de données qui décide de leur valeur)
    - La priorité doit être comprise entre 1 et 5 inclus
    - Aucun autre attribut que ceux cités ci-dessus n'est autorisé
    - Les seuls sous-éléments autorisés sont « description », « owner » et « asker »
  - Utilisateur
    - L'attribut « name » est obligatoire
    - Aucun sous-élément n'est autorisé
- Toutes les dates manipulées respectent le format jj/MM/aaaa (que ce soit en entrée de la WebApp ou en sortie)
- Accès aux données « brutes » (cas des url commençant par */about*)
  - Cela s'effectuera via l'URL /about avec la gestion des paramètres d'URL comme indiqué dans le tableau du §1.2.2 du support de TD
  - Note : les données ne doivent pas être formatées et doivent uniquement renvoyer l'élément demandé sans ajout de texte (par exemple, pour accéder au nombre d'élèves, il faut uniquement renvoyer un chiffre et non une phrase du type « 2 élèves »)
- Accès aux données XML
  - Toutes les tâches au format XML seront accessibles via l'URL /task/list/xml
  - Une tâche dont l'id est <taskId> le sera via l'URL /task/list/xml?id=<taskId>
  - Tous les utilisateurs le seront via l'URL /user/list/xml
  - Les statistiques le seront via l'URL /stats/xml (sachant que une tâche est estimée en retard si sa « deadline » est strictement inférieure à la date du jour ; en d'autres termes si deadline => date du jour, alors on la considère comme « intime »)
- Accès aux données HTML
  - La page d'accueil sera accédée via l'URL /
  - Le formulaire d'affectation d'un professeur le sera via l'URL /about/teacher
  - La liste des utilisateurs le sera via l'URL /user/list
  - La liste des tâches le sera via l'URL /task/list
  - Le formulaire de saisie d'un nouvel utilisateur le sera via l'URL /user/create
  - Le formulaire de saisie d'une nouvelle tâche le sera via l'URL /task/create
  - Le formulaire de visualisation d'une tâche dont l'id est <taskId> le sera via l'URL /task/view?id=<taskId>
  - Le formulaire d'édition d'une tâche dont l'id est <taskId> le sera via l'URL /task/modify?id=<taskId>
- Envoie de données au serveur

- L'affectation du professeur se fera via l'URL /about/teacher/post et des données postées (pas en paramètre d'URL et pas non plus sous la forme d'un flux XML, uniquement le nom envoyé directement sans aucun autre formatage) ; cf. §1.3.2 du support de TD.
- La création d'un utilisateur s'effectuera via l'URL /user/create/url avec des paramètres ; cf. §2.4 du support de TD
- L'ajout ou la modification d'une tâche s'effectuera via l'URL /task/create/url avec des paramètres ; cf. §2.5 du support de TD
  - **Note importante** : quand l'id est passé en paramètre, le comportement doit être celui d'une modification d'une tâche déjà existante et dont l'id est celui passé en paramètre
  - Quand l'id n'est pas passé, le comportement doit être celui de la création d'une nouvelle tâche