Chapter 1

Servlet

1.1 Lab description

The purpose of this weeks lab exercise is to

1. Get introduced to the task manager Xml which you will be using during the lab exercises and hand ins throughout the course.
2. Get the Eclipse Java development environment running on your computers.
3. Make a small distributed application (web application) implementing the first version of the task manager using using XML and Java Servlets/JSP.

As part of the exercise, you are required to extend the above Task Manager web application with the following functionality.

1. Tasks by the Id: Get the task information or Xml for a supplied task id.
2. Tasks by the owner: Get the tasks owned by a given user name.
3. Tasks by the date: Get the tasks for a given date.

Develop suitable Servlets or JSP pages to offer the above functionality in the Task Manager Application. Feel free to choose a simple design and user interface for taking input parameters and passing them to the server. For example, in case of developing JSP pages one may choose HTML input elements for taking the user input and use POST/GET method to pass the input parameters to server. On the other hand, if you developing servlets, then one may simply pass the input parameters to the server in the form of query string using GET method. In case of new servlets, dont forget to add servlet mappings in the web configuration file (web.xml).

1.2 Solution

Our solution is implemented using Jsp-documents. Jsp is a construct similar to Php: code sections written in a scripting language is embedded in a mark-up language, and gets called whenever the resource is requested in order to dynamically construct a web response. In our documents, Java programming statements are embedded in an Html-page. The statements uses the XmlOutputter class to have the Jsp-tag in which it is embedded evaluate to an Xml-construct. The class JDOMParser provided with the assignment, is used to read from the task-manager-xml.xml document, and execute XPath queries on its contents.
We have implemented three Jsp-documents, offering the functionality described in the assignment:

- GetAllTasks.jsp simply returns all available tasks found in the document.
- GetTaskById.jsp searches for "task" elements in the Xml-structure that have in "id" attribute whose value is equal to the provided id. The id to look for is provided by the client as a part of the Http-formatted URI requesting the resource.
  As an invariant of the task collection back-end is that only a single task with a given id exists, this request should be expected to always evaluate to a single "task" element, although this is not explicitly enforced in the code.
- GetTasksByDate.jsp is nearly identical in structure to GetTaskById.jsp. It searches the document for tasks with a "date" attribute matching the provided URI-parameter.

The assignment requires that an additional piece of functionality, allowing tasks to be identified by their "owner". As the Xml-schema provided did not include any data for an "owner" of a task, this was regarded to be a mistake. Had their been such data included, solving this part of the assignment would likely not exemplify anything not covered in the former functions.

1.3 Test run

The following section shows a request made to each of the three documents, and the resulting server response. During test run the servlets are running on localhost at port 8080.
The Html surrounding the Xml-response have been removed as irrelevant to the content of the response.

- Get all tasks.
  request:
  http://localhost:8080/BMDS-TaskManager-01/GetAllTasks.jsp

response:
¡?xml version=”1.0” encoding=”UTF-8”?¿ ¡tasks¿¡task id=”handin-01” name=”Submit assignment-01” date=”16-12-2013” status=”not-executed” required=”false”¿ ¡description¿ Work on mandatory assignment. ¡/description¿ ¡attendants¿student-01, student-02¡/attendants¿ ¡/task¿¡task id=”review-01” name=”Review and check assignment” date=”17-12-2013” status=”not-executed” required=”false”¿ ¡description¿ Check the assignment send by students and decide whether to approve or ask them to re-submit the assignment. ¡/description¿ ¡attendants¿TA-01, Rao¡/attendants¿

¡/task¿¡task id=”reject-01” name=”ask to resubmit the assignment” date=”17-12-2013” status=”not-executed” required=”false”¿ ¡description¿ Ask students to resubmit assignment by providing necessary feedback. ¡/description¿ ¡attendants¿TA-01, Rao¡/attendants¿

¡/task¿¡task id=”approve-01” name=”mandatory assignment 01 approved” date=”17-12-2013” status=”not-executed” required=”false”¿ ¡description¿ Approve mandatory assignment 01 and inform students. ¡/description¿ ¡attendants¿TA-01, Rao¡/attendants¿

¡/task¿¡task id=”qualify-for-exam” name=”qualify for written exam” date=”21-12-2013” status=”not-executed” required=”false”¿ ¡description¿ Qualify students to take up written exam. ¡/description¿ ¡attendants¿Thomas¡/attendants¿ ¡/task¿¡/tasks¿

- Get task by id.
  request:
  http://localhost:8080/BMDS-TaskManager-01/GetTaskById.jsp?id=”handin-01”

  reponse:
  ¡?xml version=”1.0” encoding=”UTF-8”?¿ ¡tasks¿¡task id=”handin-01” name=”Submit assignment-01” date=”16-12-2013” status=”not-executed” required=”false”¿ ¡description¿ Work on mandatory assignment. ¡/description¿ ¡attendants¿student-01, student-02¡/attendants¿ ¡/task¿¡/tasks¿

- Get tasks by date.
  request:
  http://localhost:8080/BMDS-TaskManager-01/GetTaskByDate.jsp?date=”17-12-2013”

response:

¡?xml version=”1.0” encoding=”UTF-8”?¿ ¡tasks¿¡task id=”review-01” name=”Review and check assignment” date=”17-12-2013” status=”not-executed” required=”false”¿ ¡description¿ Check the assignment send by students and decide whether to approve or ask them to re-submit the assignment. ¡/description¿ ¡attendants¿TA-01, Rao¡/attendants¿

¡/task¿¡task id=”reject-01” name=”ask to resubmit the assignment” date=”17-12-2013” status=”not-executed” required=”false”¿ ¡description¿ Ask students to resubmit assignment by providing necessary feedback. ¡/description¿ ¡attendants¿TA-01, Rao¡/attendants¿

¡/task¿¡task id=”approve-01” name=”mandatory assignment 01 approved” date=”17-12-2013” status=”not-executed” required=”false”¿ ¡description¿ Approve mandatory assignment 01 and inform students. ¡/description¿ ¡attendants¿TA-01, Rao¡/attendants¿

¡/task¿¡/tasks¿

1.4 Reflection

The exercise demonstrates a simple example of a client-server distribution architecture, the use of Http to request specific resources, and mechanisms of dynamically generated web pages.
A client-server architecture is characterized by a fundamental difference between every two parties communicating; the client initiates the transaction, makes requests, remembers session state and previous responses, and have access to all servers on the network. Servers respond to requests, can receive requests from all clients using the network, does often not store session state (see section XXXXXXX), and can act as clients themselves, forwarding the requests to other servers.
This architecture has the benefits of centralizing services to single server entry-points, and decoupling the abstract service being provided from the machine executing the processing needed to provide the service. This allows for very agile and scalable systems, as the client only needs to know about the URI defining the position of the service, while the execution can be forwarded and distributed among as many servers as is needed. A service is only characterized by its URI, and not by its physical position, allowing for physical machines to be exchanged and removed while the service stays the same.
The solution implemented responds to requests with dynamic web pages. Our solution could be extended to exemplify this scalable architecture, by having each of the three pages be stored on an individual machine, and have the entry-point forward requests to these machines for execution, based on the format of the URI. In a larger system, one could take advantage of the inherently hierarchical structure of URIs by having all requests with URIs starting with a given prefix be forwarded to a corresponding server.

1.5 Conclusion

In this exercise we have managed to implement a Http-service that exposes Task entities from the provided document "task-manager-xml.xml". The solution uses .jsp documents to execute user queries and generate dynamic responses, and uses xpath statements to address the contents of the xml document.
We have not implemented the functionality described in the assignment as identifying tasks by "owner", as this information did not exist in the xml schema.