LAZY TABLE

An alternative solution to JSF “dataTable” session overload

User Guide

The Problem

Java Server Faces or JSF is a popular MVC framework for developing J2EE web-applications. This framework is a component based, event driven framework.

Many of the UI components like data table, calendar, data scroller, etc are available out of the box when using one of the many libraries available for JSF components. However the most used component of data table is designed in a manner that defeats its use in an enterprise web application which caters to concurrent users. This is because the data table component stores the entire list of data objects in the session scope and but displays only one page of data at any time. The entire list is only required to find the total count and when the user submits a request for the next page or for sorting the data on a particular field, it performs this operation with the entire set of data in session.

This can over-load the application server's memory all too soon as the size of the list and number users maintained in memory can increase exponentially.

The following solution tries to negate the loading of the complete data list in memory by using AJAX calls to fetch one page of data at a time and hence lazy load/ refresh of the data table section of the screen alone.

The Solution: A deep dive!

The three major components in our solution, which play a role in implementing the Lazy Table or the remote/server side pagination and sorting, are as below:

- **Data Access Object (DAO/Entity/Service/EJB)**
- **Bean (servlet/action)**
- **Java Server Page (JSP/web page)**

Data Access Object

The Data Access Object will undergo a change predominantly in the queries:

Where previously there was a single search query returning the entire list of results, the modified data access object will now use two queries.

- The first query will return only a page size worth results (eg: 10 of 100, if page size is configured as 10).
- The second query will return an integer value indicative of the total number of results (eg: 100)

The actual query will vary from one RDBMS to another and will also include the sort by and sort order value. Below is the demonstration of the query to return 5 records (ranging from 1to 5), ordered by the field last name in Ascending order.

- In PostgresSQL:

```
SELECT * FROM user WHERE role_id < 2
ORDER BY lastname ASC
LIMIT 5 OFFSET 1;
```

- In MYSQL:

```
SELECT * FROM user WHERE role_id < 2
ORDER BY lastname ASC
LIMIT 1, 5;
```

- In Oracle:

```
SELECT * FROM user WHERE role_id < 2
ORDER BY lastname ASC
WHERE ROWNUM BETWEEN 1 AND 5;
```

These queries will return a list and the list will be used to feed the JSP page to display results for the page selected.

The count query will be the same in most RDBMS and would be of the format:

```
SELECT COUNT(1) FROM user WHERE role_id < 2;
```

The above is demonstrated in the code in the class: UserServiceImpl.java. The methods of particular interest are searchUser and searchUserCount as shown here:

```
/**
     * Search user as per input criteria
     * @param roleId
     * @param firstName
     * @param lastName
     * @param email
     * @param login
     * @return list of users one page long.
     */
    @Override
    public List<User> searchUser(int roleId, String firstName, String lastName, String
email, String login, String sortParam, String sortOrder, int rowCount, int rowOffset)
{
        String query = "SELECT * FROM user WHERE role_id < ? ";

        if (firstName != null && !firstName.trim().equals("")) {
            query += "AND firstName LIKE '" + firstName.replaceAll("'", "''") + "%'";
        }
        if (lastName != null && !lastName.trim().equals("")) {
            query += "AND lastName LIKE '" + lastName.replaceAll("'", "''") + "%'";
        }
        if (email != null && !email.trim().equals("")) {
            query += "AND email LIKE '" + email.replaceAll("'", "''") + "%'";
        }
        if (login != null && !login.trim().equals("")) {
            query += "AND login LIKE '" + login.replaceAll("'", "''") + "%'";
        }

        if (sortParam != null && !sortParam.trim().equals("")
                && sortOrder != null && !sortOrder.trim().equals("")) {
            query += "ORDER BY " + sortParam + " " + sortOrder + "";
        }

        query += " LIMIT " + rowCount + ", " + rowOffset + "";

        System.out.println("search query: " + query);

        List<User> users = new ArrayList<User>();
        users = em.createNativeQuery(query, User.class).setParameter(1,
roleId).getResultList();
        return users;
    }

@Override
    public int searchUserCount(int roleId, String firstName, String lastName, String
email, String login) {
        Long count;

        String query = "SELECT COUNT(1) FROM user WHERE role_id < ? ";

        if (firstName != null && !firstName.trim().equals("")) {
            query += "AND firstName LIKE '" + firstName.replaceAll("'", "''") + "%'";
        }
        if (lastName != null && !lastName.trim().equals("")) {
            query += "AND lastName LIKE '" + lastName.replaceAll("'", "''") + "%'";
        }
        if (email != null && !email.trim().equals("")) {
            query += "AND email LIKE '" + email.replaceAll("'", "''") + "%'";
        }
        if (login != null && !login.trim().equals("")) {
            query += "AND login LIKE '" + login.replaceAll("'", "''") + "%'";
        }
        count = (Long) em.createNativeQuery(query).setParameter(1,
roleId).getSingleResult();
        return count.intValue();
    }
```

Bean

In JSF the Bean performs the role of a controller or servlet which handles the various calls from the JSP. This bean in our particular implementation is the UserListBean.java.

The UserListBean has various methods which are of importance that are listed below for the server side pagination and sorting:

-- searchUser

-- changeSorting

-- getFirstPage

-- getPrevPage

-- getNextPage

-- getLastPage

- **searchUser**: This method is the heavyweight, which invokes the corresponding data access methods in UserServiceImpl.

```
    /**
     * Search user, this method is called when the form is loaded
     * @return
     */
    @PostConstruct
    public String searchUser() {
        FacesContext context = FacesContext.getCurrentInstance();
        List<UserModel> userModelList = new ArrayList<UserModel>();
        User sessionUser = (User)
context.getExternalContext().getSessionMap().get("user");

        if (null == sessionUser) {
            HttpSession session = (HttpSession)
context.getExternalContext().getSession(false);
            session.invalidate();
            return "login";
        } else {
            int roleId = new Integer(sessionUser.getRoleId());
            if (sessionUser != null && roleId == new Integer(new
MessageProvider().getValue("admin")).intValue()) {
                displaySearch = true;

                if(sortParam == null){
                    sortParam = "user_id";
                }
                if(sortOrder == null){
                    sortOrder = "ASC";
                }

                if(currentPage == null){
                    this.setCurrentPage(1);
                }

                String currPage =
FacesContext.getCurrentInstance().getExternalContext().getRequestParameterMap
().get("currentPage");
                System.out.println("Params currPage: " + currPage);
                if (currPage != null) {
                    this.setCurrentPage(new Integer(currPage));
                }

                System.out.println("currentPage: " +  this.getCurrentPage());


                numberOfRecords = userService.searchUserCount(roleId,
firstName, lastName, email, login);
                System.out.println("numberOfRecords: " + numberOfRecords);
                numberOfPages = numberOfRecords / pageSize + 1;
                pages = new ArrayList<Integer>();
                for(int i=0;i<numberOfPages ; i++ ){
                    pages.add(i+1);
                }

                System.out.println("numberOfPages: " + numberOfPages);

                int startRecord = (this.getCurrentPage()-1) * pageSize;
                List<User> users = userService.searchUser(roleId, firstName,
lastName, email, login, sortParam, sortOrder, startRecord, pageSize);
                System.out.println("Here in search users: " + users.size() +
" ");
                for (User user : users) {
                    UserModel userModel = new UserModel(user.getUserId(),
user.getLogin(),
                            user.getPassword(), user.getFirstName(),
user.getLastName(), user.getEmail());
                    userModelList.add(userModel);
                }
            } else {
                displaySearch = false;
                UserModel userModel = new UserModel(sessionUser.getUserId(),
sessionUser.getLogin(),
                        sessionUser.getPassword(),
sessionUser.getFirstName(), sessionUser.getLastName(),
sessionUser.getEmail());
                userModelList.add(userModel);
            }
            this.setUserList(userModelList);
            return null;
        }
    }
```

- **changeSorting**: Performs the logic to identify the sort order and the sort by parameter and then invoking searchUser method.

```
public String changeSorting() {
        String sort =
FacesContext.getCurrentInstance().getExternalContext().getRequestParame
terMap().get("sortParam");

        if (sort != null) {
            if (sortParam.equalsIgnoreCase(sort)) {
                if (sortOrder.equalsIgnoreCase("ASC")) {
                    sortOrder = "DESC";
                } else {
                    sortOrder = "ASC";
                }
            } else {
                sortParam = sort;
                sortOrder = "ASC";
            }
        }
        this.setCurrentPage(1);
        searchUser();
        return null;
    }
```

- **getFirstPage**: Performs the logic of getting the first page of data and displaying the same by invoking searchUser method.

```
public String getFirstPage(){
     this.setCurrentPage(1);
      searchUser();
      return null;
  }
```

- **getPrevPage**: Performs the logic of getting the previous page of data and displaying the same by invoking searchUser method.

```
 public String getPrevPage(){
     this.setCurrentPage(this.getCurrentPage()- 1);
     searchUser();
     return null;
 }
```

- **getNextPage:** Performs the logic of getting the next page of data and displaying the same by invoking searchUser method.

```
  public String getNextPage() {
        this.setCurrentPage(this.getCurrentPage()+ 1);
        searchUser();
        return null;
    }
```

- **getLastPage**: Performs the logic of getting the last page of data and displaying the same by invoking searchUser method.

```
   public String getLastPage() {
        this.setCurrentPage(numberOfPages) ;
        searchUser();
        return null;
    }
```

JSF

The JSF changes predominantly demonstrate the integration with AJAX tags. We use the Ajax4Jsf library and hence use the a4j

tags to refresh only the result list, or the resultant data table, without refreshing the entire page. The appropriate methods are invoked

on click of the necessary Hyperlinks and the Command buttons.

Hence instead of usage of tags- "h:commandLink" we have used "a4j:commandLink" and replaced "h:commandButton" with "a4j:commandButton".

The a4j tags provide us with the all-important reRender attribute which allows us to specify the region to re-load.

On clicking one of the column headers, the userListBean.changeSorting method of the bean is invoked with the column header passed in as the parameter.

Similarly on clicking on the pagination methods, appropriate methods (getFirstPage, getPrevPage, getNextPage, getLastPage) in the bean are invoked.

Below is the extract of the result list from the searchusers.jsp.

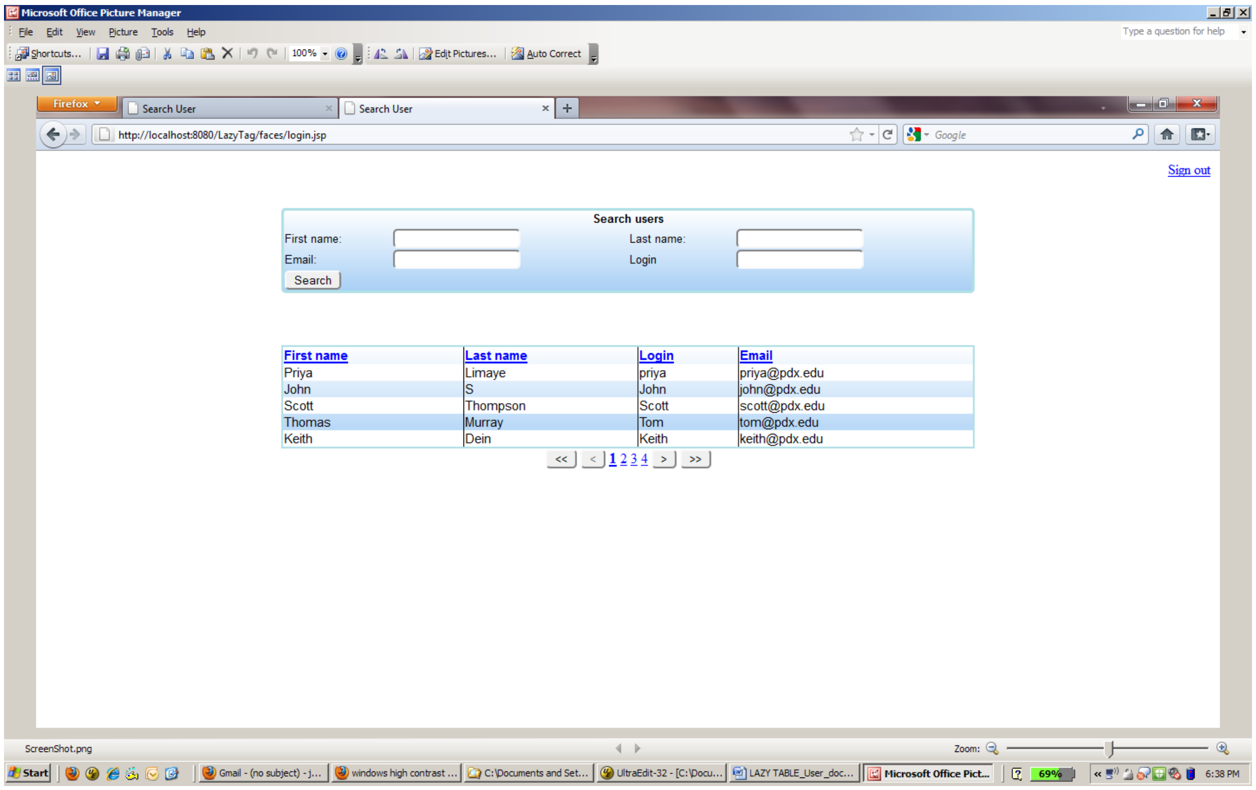
```
 <a4j:outputPanel id="eheader" >

                        <t:dataTable id="userTable"
value="#{userListBean.userList}" var="userData"
                                     styleClass="table1"
rowClasses="oddRow,evenRow" width="800px" rules="COLS" border="2">
                            <t:column style="border=10;">
                                <f:facet name="header">
                                    <a4j:commandLink
value="#{msg.firstname1}" action="#{userListBean.changeSorting}"
styleClass="headerRow" reRender="eheader">
                                        <f:param name="sortParam"
value="firstName" />
                                    </a4j:commandLink>
                                </f:facet>
                                <h:outputText value="#{userData.firstName}"/>
                            </t:column>
                            <t:column>
                                <f:facet name="header">
                                    <a4j:commandLink value="#{msg.lastname1}"
action="#{userListBean.changeSorting}" styleClass="headerRow"
reRender="eheader">
                                        <f:param name="sortParam"
value="lastName" />
                                    </a4j:commandLink>
                                </f:facet>
                                <h:outputText value="#{userData.lastName}"/>
                            </t:column>
                            <t:column>
                                <f:facet name="header">
                                    <a4j:commandLink value="#{msg.login1}"
action="#{userListBean.changeSorting}" styleClass="headerRow"
reRender="eheader">
                                        <f:param name="sortParam"
value="login" />
                                    </a4j:commandLink>
                                </f:facet>
                                <h:outputText value="#{userData.login}"/>
                            </t:column>
                            <t:column>
                                <f:facet name="header">
                                    <a4j:commandLink value="#{msg.email1}"
action="#{userListBean.changeSorting}" styleClass="headerRow"
reRender="eheader">
                                        <f:param name="sortParam"
value="email" />
                                    </a4j:commandLink>
                                </f:facet>
                                <h:outputText value="#{userData.email}"/>
                            </t:column>
                        </t:dataTable>
                        <a4j:commandButton value="<<" 
action="#{userListBean.getFirstPage}"  reRender="eheader"/>
                        <h:outputText value=" "/>
                        <a4j:commandButton value="<" 
action="#{userListBean.getPrevPage}" disabled="#{userListBean.currentPage == 
1}" reRender="eheader" />
                        <h:outputText value=" "/>
                        <t:dataList value="#{userListBean.pages}" var="page">
                            <a4j:commandLink 
action="#{userListBean.searchUser}" value="#{page}"  reRender="eheader" 
rendered="#{userListBean.currentPage != page}">
                                <f:param name="currentPage" value="#{page}"/>
                            </a4j:commandLink>
                            <a4j:commandLink 
action="#{userListBean.searchUser}" value="#{page}"  reRender="eheader" 
rendered="#{userListBean.currentPage == page}"
                                             style="font-size:18px;font-
weight:bold">
                                <f:param name="currentPage" value="#{page}"/>
                            </a4j:commandLink>
                            <h:outputText value=" "/>
                        </t:dataList>
                        <a4j:commandButton value=">" 
action="#{userListBean.getNextPage}" disabled="#{userListBean.currentPage == 
userListBean.numberOfPages}" reRender="eheader">
                        </a4j:commandButton>
                        <h:outputText value=" "/>
                        <a4j:commandButton value=">>" 
action="#{userListBean.getLastPage}" reRender="eheader"/>
                        <h:outputText value=" "/>

                    </a4j:outputPanel>
```

Outcome

The final look and feel of the data table with AJAX and lazy loading is as shown below.



First name	Last name	Login	Email
Priya	Limaye	priya	priya@pdx.edu
John	S	John	john@pdx.edu
Scott	Thompson	Scott	scott@pdx.edu
Thomas	Murray	Tom	tom@pdx.edu
Keith	Dein	Keith	keith@pdx.edu