Document Manager JSP Page and Supporting Classes

Your Name

May 19, 2024

Contents

1 Overview

This document provides detailed documentation for the JSP page and supporting Java classes used to manage documents in a web application. The JSP page includes functionalities for adding, updating, deleting, and downloading documents, as well as managing access permissions.

2 JSP Page: Document Manager

The JSP page serves as the user interface for managing documents. It includes import statements, HTML and CSS for layout and styling, and dynamic content rendering using JSP scriptlets.

2.1 Import Statements

The following import statements include the necessary Java classes for the page:

```
<%@page import="com.jee.beans.User"%>
<%@page import="com.jee.beans.Access"%>
<%@page import="com.jee.beans.Document"%>
<%@page import="java.util.List"%>
```

2.2 HTML and CSS

The HTML and CSS are used to create a user-friendly interface.

2.2.1 CSS for Styling

```
<style>
body {
    font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
    background-color: #f0f2f5;
    color: #333;
    margin: 0;
    padding: 0;
    display: flex;
    flex-direction: column;
    align-items: center;
}

h1, h3 {
    color: #2c3e50;
}

/* ... additional CSS omitted for brevity ... */
</style>
```

2.2.2 HTML Structure

```
<div class="container">
    <h1 class="info">My Documents Manager</h1>
    <h3 class="title">Hello, <%=user.getUsername()%></h3>
```

```
    <div id="add-div">
        <h4>add File process :</h4>
        <!-- JSP scriptlets for form steps -->
    </div>

    <table>
        <tr>
            <th>Document Title</th>
            <th>Document Owner</th>
            <th>Permission</th>
            <th colspan="4">Actions</th>
        </tr>
        <!-- Dynamic rows generated by JSP scriptlets -->
    </table>
    <a href="signin.jsp"><button id="signout-btn" class="btn">sign out</button></a>
</div>
```

2.3 Dynamic Content Rendering

Dynamic content is generated using JSP scriptlets:

```
<%
User user = (User) request.getAttribute("user");
List<Access> userDocsPermissions = (List<Access>) request.getAttribute("userDocsPermissions");
List<User> userDocsOwners = (List<User>) request.getAttribute("userDocsOwners");
List<Document> userDocs = (List<Document>) request.getAttribute("userDocs");
String step = (String) request.getAttribute("step");
if (step == null) {
    step = "initial";
}
%>
```

This section retrieves the necessary data from the request attributes.

2.4 Adding a Document

The process of adding a document is divided into steps.

2.4.1 Initial Step

```
<form method="post" action="storeFile.dostart">
    <input type="hidden" name="initiate" value="true">
    <button>Start</button>
</form>
```

2.4.2 Step 1: Title and Description

```
<form method="post" action="storeFile.dostep1">
    <input type="text" name="doc_title" placeholder="Document Title">
```

```
    <input type="text" name="doc_desc" placeholder="Document Description">
    <button>Next</button>
</form>
```

2.4.3 Step 2: File Upload

```
<form method="post" action="storeFile.dostep2" enctype="multipart/form-data">
    <input type="file" name="uploadedFile">
    <button>Save</button>
</form>
```

3 Supporting Classes

The supporting classes handle the backend logic for storing and downloading documents.

3.1 StoreDocumentAction Class

This class handles the document storing process in multiple steps.

```
package com.jee.presentation;

import java.io.*;
import com.jee.beans.Access;
import com.jee.beans.Document;
import com.jee.beans.User;
import com.jee.business.BusinessFacade;
import com.jee.business.LocalDocsManager;
import jakarta.servlet.ServletException;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import jakarta.servlet.http.HttpSession;
import jakarta.servlet.http.Part;

public class StoreDocumentAction extends Action {
    LocalDocsManager docsdb;

    public StoreDocumentAction(BusinessFacade facade, LocalDocsManager docsdb) {
        super(facade);
        this.docsdb = docsdb;
    }

    @Override
    public String execute(HttpServletRequest request, HttpServletResponse response) {
        if (request.getRequestURI().contains("step2")) {
            // Handling file upload step
            // ... code omitted for brevity ...
        } else if (request.getRequestURI().contains("step1")) {
            // Handling title and description step
            // ... code omitted for brevity ...
        } else if (request.getRequestURI().contains("start")) {
            // Handling initial step
```

```
            // ... code omitted for brevity ...
        }
        return null;
    }
}
```

3.2 DownloadDocumentAction Class

This class handles the document downloading process.

```
package com.jee.presentation;

import java.io.*;
import com.jee.beans.Document;
import com.jee.business.BusinessFacade;
import com.jee.business.LocalDocsManager;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import jakarta.servlet.http.HttpSession;

public class DownloadDocumentAction extends Action {

    private LocalDocsManager docsdb;
    private static final int DEFAULT_BUFFER_SIZE = 10240;

    public DownloadDocumentAction(BusinessFacade facade,
   LocalDocsManager localdb) {
        super(facade);
        docsdb = localdb;
    }

    @Override
    public String execute(HttpServletRequest request,
   HttpServletResponse response) {
        // Handling file download
        // ... code omitted for brevity ...
        return null;
    }

    private static void close(Closeable resource) {
        if (resource != null) {
            try {
                resource.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```