Detailed Explanation of StoreDocumentAction and DownloadDocumentAction

Your Name

May 19, 2024

Contents

1 Overview **2**

2 StoreDocumentAction Class **2**

2.1 Class Overview 2

2.2 Handling Initial Step 3

2.3 Handling Title and Description Step 3

2.4 Handling File Upload Step 3

3 DownloadDocumentAction Class **4**

3.1 Class Overview 4

3.2 Handling File Download 5

4 Conclusion **6**

1 Overview

This document provides a detailed explanation of the ‘StoreDocumentAction‘ and ‘DownloadDocumentAction‘ classes. These classes handle the processes of storing and downloading documents, including file handling and multipart form data processing.

2 StoreDocumentAction Class

The ‘StoreDocumentAction‘ class manages the process of storing documents in multiple steps.

2.1 Class Overview

```java
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;
    }
}
```

2.2 Handling Initial Step

In the initial step, a form submission starts the document storing process.

```java
if (request.getRequestURI().contains("start")) {
    // Handling initial step
    request.setAttribute("step", "step1");
    return "document_manager.jsp";
}
```

This snippet sets the step attribute to ”step1” and forwards the user to the document manager page.

2.3 Handling Title and Description Step

In the second step, the user inputs the document’s title and description.

```java
else if (request.getRequestURI().contains("step1")) {
    String docTitle = request.getParameter("doc_title");
    String docDesc = request.getParameter("doc_desc");
    HttpSession session = request.getSession();
    session.setAttribute("docTitle", docTitle);
    session.setAttribute("docDesc", docDesc);
    request.setAttribute("step", "step2");
    return "document_manager.jsp";
}
```

This code retrieves the document title and description from the request and stores them in the session.

2.4 Handling File Upload Step

In the final step, the user uploads the document file.

```java
else if (request.getRequestURI().contains("step2")) {
    HttpSession session = request.getSession();
    User user = (User) session.getAttribute("user");
    String docTitle = (String) session.getAttribute("docTitle");
    String docDesc = (String) session.getAttribute("docDesc");

    Part filePart = request.getPart("uploadedFile");
    String fileName = filePart.getSubmittedFileName();
    InputStream fileContent = filePart.getInputStream();

    // Save the file to disk
    File uploads = new File("uploads");
    if (!uploads.exists()) {
        uploads.mkdir();
    }
    File file = new File(uploads, fileName);
    try (OutputStream out = new FileOutputStream(file)) {
        byte[] buffer = new byte[1024];
        int bytesRead;
        while ((bytesRead = fileContent.read(buffer)) != -1) {
            out.write(buffer, 0, bytesRead);
        }
    }

    // Store document metadata in the database
    Document document = new Document(docTitle, docDesc, file.getPath(), user.getId());
    docsdb.storeDocument(document);

    request.setAttribute("step", "initial");
    return "document_manager.jsp";
}
```

This snippet handles the file upload process: 1. Retrieves the document details from the session. 2. Gets the uploaded file as a 'Part' object. 3. Saves the file to the disk. 4. Stores the document metadata in the database.

3 DownloadDocumentAction Class

The 'DownloadDocumentAction' class manages the process of downloading documents.

3.1 Class Overview

```java
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
        String docId = request.getParameter("docId");
        Document document = docsdb.getDocumentById(Integer.parseInt(docId));

        File file = new File(document.getFilePath());
        response.setHeader("Content-Type", "application/octet-stream");
        response.setHeader("Content-Length", String.valueOf(file.length()));
        response.setHeader("Content-Disposition", "attachment; filename=\"" + file.getName() + "\"");

        try (BufferedInputStream input = new BufferedInputStream(new FileInputStream(file), DEFAULT_BUFFER_SIZE);
             BufferedOutputStream output = new BufferedOutputStream(response.getOutputStream(), DEFAULT_BUFFER_SIZE)) {
            byte[] buffer = new byte[DEFAULT_BUFFER_SIZE];
            int bytesRead;
            while ((bytesRead = input.read(buffer)) != -1) {
                output.write(buffer, 0, bytesRead);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
        return null;
    }

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

3.2 Handling File Download

This code handles the process of downloading a file.

```java
String docId = request.getParameter("docId");
Document document = docsdb.getDocumentById(Integer.parseInt(docId));

File file = new File(document.getFilePath());
response.setHeader("Content-Type", "application/octet-stream");
response.setHeader("Content-Length", String.valueOf(file.length()));
response.setHeader("Content-Disposition", "attachment; filename=\"" + file.getName() + "\"");

try (BufferedInputStream input = new BufferedInputStream(new FileInputStream(file), DEFAULT_BUFFER_SIZE);
     BufferedOutputStream output = new BufferedOutputStream(response.getOutputStream(), DEFAULT_BUFFER_SIZE)) {
    byte[] buffer = new byte[DEFAULT_BUFFER_SIZE];
    int bytesRead;
    while ((bytesRead = input.read(buffer)) != -1) {
        output.write(buffer, 0, bytesRead);
    }
} catch (IOException e) {
    e.printStackTrace();
}
```

This snippet: 1. Retrieves the document ID from the request. 2. Gets the document metadata from the database. 3. Prepares the response headers for file download. 4. Streams the file content to the response output stream.

4 Conclusion

The ‘StoreDocumentAction‘ and ‘DownloadDocumentAction‘ classes provide robust mechanisms for handling file uploads and downloads in a web application. This document detailed their implementation, focusing on multipart handling and file processing.