CS580 Advanced Software Engineering
Assignment 4 - Photo Management

Due Date
Wednesday, October 22, 2014

Score
20

Questions and Directions

In this assignment, you will start working on the core functions of the iphoto-web project - managing photos for different users. Specifically, we want to support 1) add a photo; 2) get a photo; 3) list the user's photos; 4) delete a photo.

The goal of this assignment is to let you design your software module in the clean and extensible way. You should create a `PhotoManager` interface/class to handle the different tasks, and call the `PhotoManager` in your `WebController`. Please take a look at the `UserManager.java` and `FSUserManager.java`, and see how this type of class/interface hierarchy is being applied in `App.java` and `WebController.java`.

All the photos files should be **stored as files locally in your disk**. There is NO database or cloud storage involved with this project.

For different users, we need to separate their photo files. Ideally, your file system structure should be something like below, where iphoto is the root folder for your project and different users have their own folders to store photo files:

```
- iphoto/
    - user1/
        - photo1.jpg
        - photo2.jpg
        - myphoto.jpg
    - user2/
        - photo1.jpg
        - home1.jpg
        - home2.jpg
```

To learn how to handle the file path and structure, please take a look at `ResourceResolver.java` in your current project. It shows an example of how to create a `<user>.json` file in the user folder. You can handle the photo files in the similar way.

1. Add a photo

You need to implement the method `addPhoto` defined the `WebController.java`.

```
@RequestMapping(value = "/iphoto/{userId}/photo", method = RequestMethod.POST)
void addPhoto(
        @PathVariable("userId") String userId,
        @RequestParam("photoFile") MultipartFile file,
        HttpServletResponse response) throws IOException {

}
```

This method takes two input parameters - *user ID* and photo *file* being uploaded. `userId` could be used to determine the path to store your file as the file system suggested above (each user has a specific folder to store all the photos). The file input is a type of *MultipartFile*, which is a representation of an uploaded file received in the HTTP request. You can google "Spring MultipartFile" to learn more about this class if interested. The key thing to know here is that once you upload a photo file through the HTTP request, you can read and access the file data here, and get the data with **`file.getInputStream()`** method.

Therefore, the key function of this method is to convert the photo file input stream to a file in your local directory. You can google how to "save Java input stream as a file". One simple solution I suggest is the example code below:

```
            OutputStream os = new FileOutputStream(file);
            IOUtils.copy(fileInputStream, os);
            os.close();
            fileInputStream.close();
```

`IOUtils` is a class from the 3rd-party library (http://commons.apache.org/). You need to import the library through Maven dependency:

```
            <dependency>
                  <groupId>org.apache.commons</groupId>
                  <artifactId>commons-io</artifactId>
                  <version>1.3.2</version>
            </dependency>
```

To test the method, the simplest way is to use the test page we provided. Go to the URL: http://localhost:8080/iphoto/user1/home (you can use any user ID) and upload the photo. If your implementation is correct, check if your method creates the file successfully in your disk.

You can also use other tools to send the HTTP requests to test the method, such as curl.

2. Get a photo

The basic logic for this method has been provided in the method below. The input `photoId` is **the file name of the photo WITHOUT extension name**. For instance, if the request is http://localhost:8080/iphoto/user1/photo/home1 , it should returns the photo file `home1.jpg` located in the user folder `user1`. We can assume all the photos have the same extension name jpg.

```
@RequestMapping(value = "/iphoto/{userId}/photo/{photoId}", method = RequestMethod.GET)
void getPhoto(
            @PathVariable("userId") String userId,
            @PathVariable("photoId") String photoId,
            HttpServletResponse response) {

       // 1. Get the file of your photo based on the userId and photoId
       File photoFile = …..

       // 2. Return the photo file as an output stream using the code below
       IOUtils.copy(new FileInputStream(photoFile), response.getOutputStream());
       response.setContentType("image/jpeg");
       response.flushBuffer();
}
```

The code to return the file in the HTTP response has been provided here. It simply converts your file to an output stream as the right MIME type, so that your web browser can display the photo. Make sure you added the Maven dependency as described in Question 1.

To test this method, you can use the URL with an valid `photoId` to check if the photo you just added can be returned and displayed in your browser.

3. List the user photo IDs

This method returns a list of photo IDs owned by the given user.

If you store all the photos for a given user in the user's folder, you can simply iterate all the photo files in that folder, and return the file names as a list of strings. (photo ID only contains the file name, WITHOUT extension name). You might want to google how to iterate files in a directory in Java.

To test, try the list URL directly in your browser:

http://localhost:8080/iphoto/user1/photos

4. Delete a photo

Delete photo method also accepts `userId` and `photoId` as the input parameters. You need to delete the photo file in your user’s folder.

To test, go to the test page:

http://localhost:8080/iphoto/user1/home

If you have already uploaded photo for your user and implemented Question 3 correctly, you should be able to see the list of photos and the buttons to delete.

Note

The current implementation contains a bug that affects the test web page displaying the photos. Please change the following code in `WebController.java` from

```
@RequestMapping(value = "/iphoto/filters/list", method = RequestMethod.GET)
List<String> getAvailableFilters() {
    // TODO
    return null;
}
```

into

```
@RequestMapping(value = "/iphoto/filters/list", method = RequestMethod.GET)
List<String> getAvailableFilters() {
    // TODO
    return new ArrayList<String>();
}
```