CSC 415/515 ONLINE PHOTO ALBUM

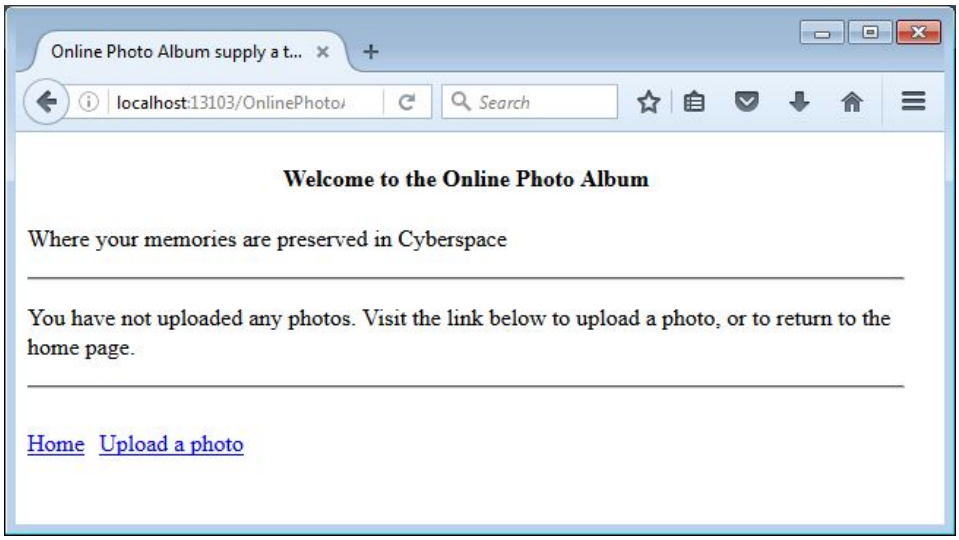
PROFESSOR GODFREY MUGANDA

The goal of this project is to write a web application that allows a person to maintain their photographs in "the cloud." A person using the web app can upload a photograph with a short description, or view previously uploaded photographs.

1. How the Application Works

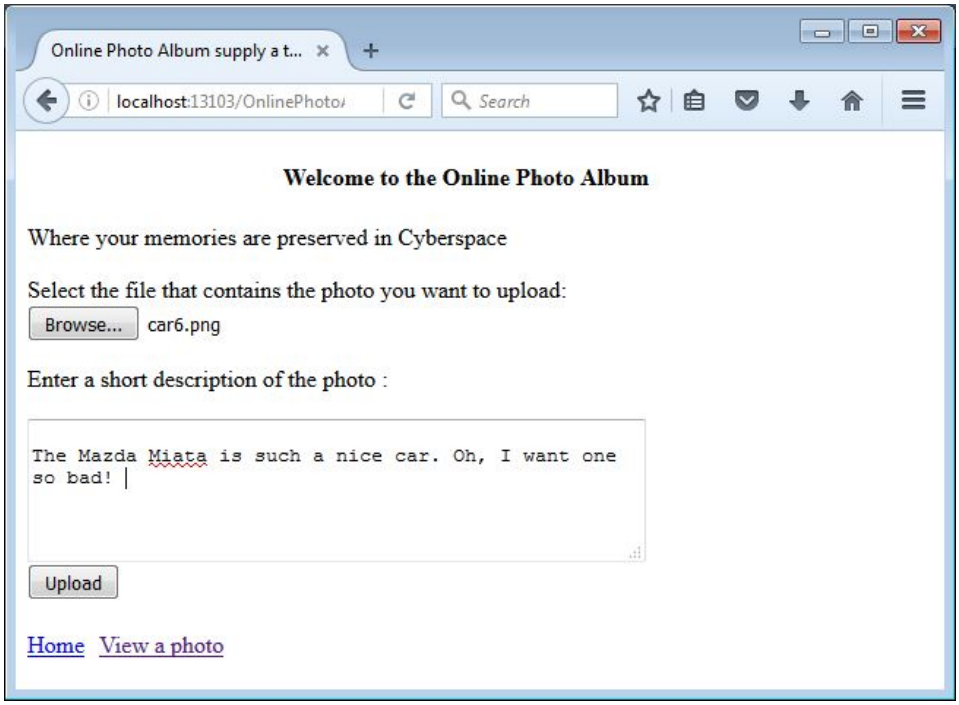
Here is now the web app works. A person's first encounter with the app is the following welcome page:

As you can see, the welcome page has two links. Hitting the first link when there are no photos to view takes you to the page shown below:



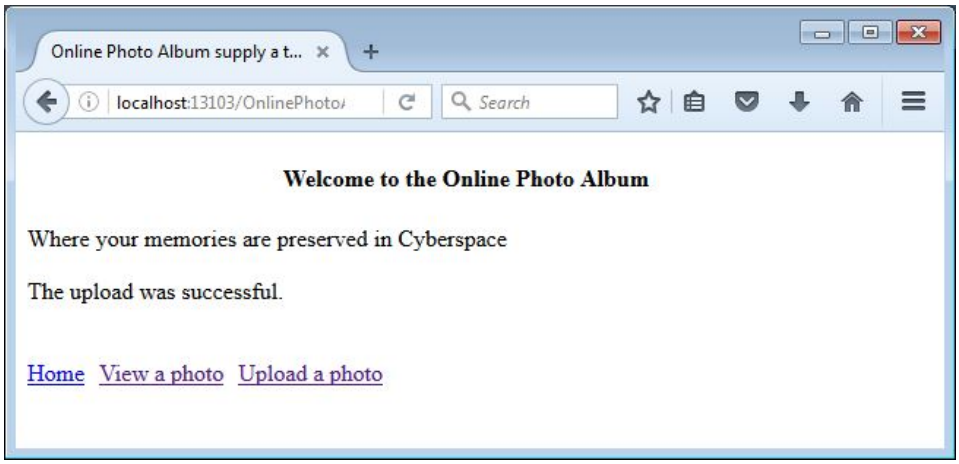
This page announces the fact that there are no photos to view. It also provides two links at the bottom, one to transition back to the home page, and the other to upload a photo.

Clicking the link to upload a photo on either of the first two pages shown brings up the HTML page below.



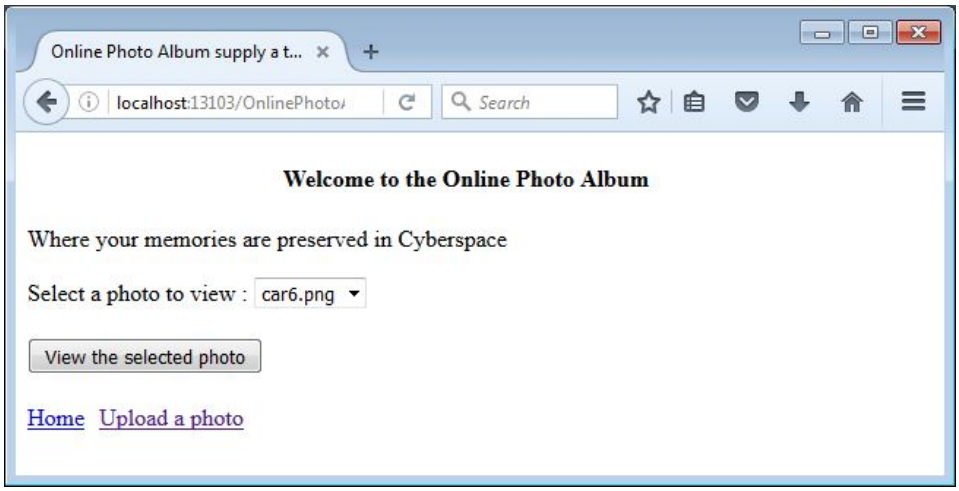
This page has a form that allows a user to select a photo to upload and enter a short description into a text area element.

Hitting the upload button performs the requested tasks and displays a confirmation page:

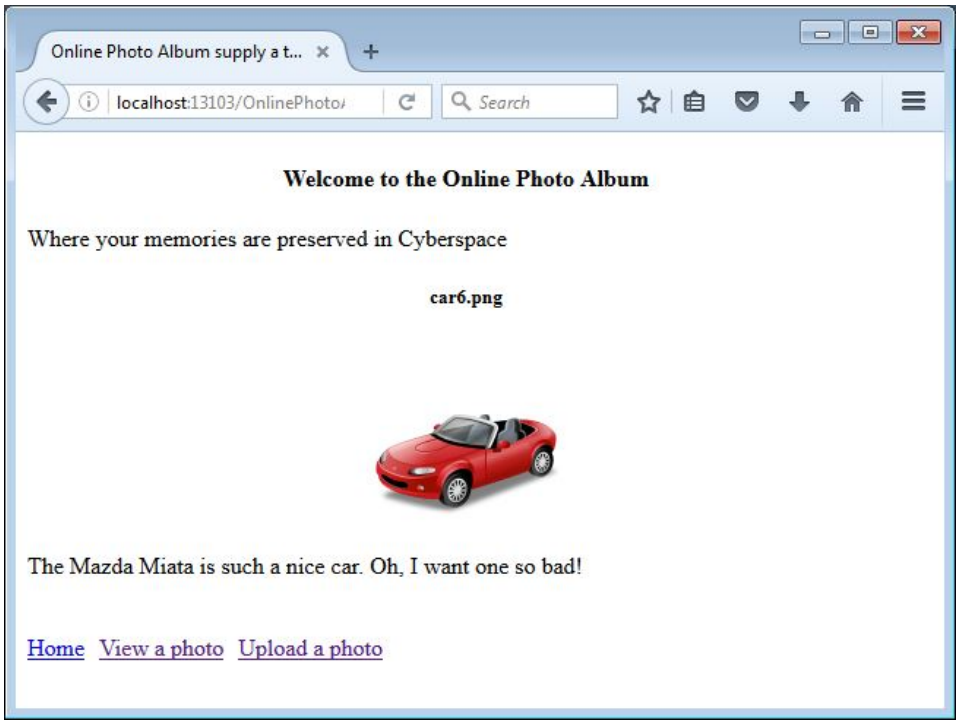


On this page, a user can return to the home page, go to the page that allows viewing of uploaded photos, or return to the photo upload page.

The photo view page displays a drop-down list of all previously uploaded photos (the drop-down identifies photos by their file names).



Selecting a photo to view displays a page that shows both the photo and the short description.



2. Sketch of Design

This application will use a flat-file as a database for uploaded photos (Relational databases will come later). Begin by creating a web application in Netbeans, and then add a class

```
package PhotoAlbum;
import java.io.Serializable;

public class Photo implements Serializable
{
    public String name;
    public String description;
    public Photo(){}
    public Photo(String nm, String descr)
    {
        this.name = nm;
        this.description = descr;
    }
}
```

Notice that this class implements the `Serialiazable` interface. Classes that implement this interface can be written and read from a flat file or any IO stream with a single method. The `name` of a photo is the same as name of the file containing the image.

Your application will maintain its database in the form of a map that maps file names to `Photo` objects. The serialized version of the map, as well as uploaded files containing the photos, will be stored in a file called `photos.data` in the created sub-folder of `Temp` when the application is not running. Whenever the application

is deployed, it will read this map into memory and make it available to all servlets as an attribute of the servlet context. The application will also copy the image files a sub-folder `images/storedphotos` of the document root's web folder.

Follow the following steps:

1. Create a folder `Temp` on the C: drive of your computer, and then inside that folder, create a sub-folder whose name is your last name. You may use another location, but this is a good option.

2. Add a web listener to your project that can respond to context events. It is good idea to put this in a package `PhotoAlbum`.

```
public class PhotoWebAppListener implements ServletContextListener
{
    // Opens up the data location and copies the serialized map
    // into an in-memory map, and then copies the image files
    // into the images/storedphotos web folder so they are
    // accessible to hyperlinks. Puts the map as an attribute
    // of the servlet context.
    @Override
    public void contextInitialized(ServletContextEvent sce)
    {

    }

    // Stores the map back into the Temp folder on the file system.
    @Override
    public void contextDestroyed(ServletContextEvent sce)
    {

    }
}
```

Use serialization to read the map from disk upon deployment and write the map to disk when the application is being undeployed. To copy the images files from the subfolder of `Temp` into the `images/storedphotos` folder, use the `copy` methods of the `Files` class. You should also look at the `Path` class when you need to work with path names of files. You may also need to use `Files.exists()` and `Files.delete()`.

To test while developing, you may want to add a debugging servlet that can look at various values, such as real paths of resources, and send them to a browser as a text file. The debug servlet can be in the default package.

3. Add a photo upload servlet. The `doGet()` method of this servlet might use a request dispatcher to return a HTML form that allows the user to upload an image file accompanied with a short description of the image:

```
<!DOCTYPE html>
<html>
    <head>
        <title>Online Photo Album supply a title</title>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width,
                                       initial-scale=1.0">
```

```
        <style>
            body {width:600px;}
            h4 {
                text-align: center;
            }
            a {margin-right : 10px;}
        </style>
    </head>
    <body>
        <h4>Welcome to the Online Photo Album</h4>
        <p>Where your memories are preserved in Cyberspace</p>

        <form action="PhotoUploadServlet" method="POST"
                              enctype="multipart/form-data">
            Select the file that contains the photo
            you want to upload:
            <input type="file" name="uploadedFile"  />   <br/>

            <p>Enter a short description of the photo : </p>

            <textarea  name = "description" rows="5" cols="50">

            </textarea> <br/>

            <button> Upload </button>
        </form>
        <br/>
        <a href="index.html" style="float: left">Home</a>
        <a href="PhotoViewServlet" style="float: left">
           View a photo
        </a>
    </body>
</html>
```

This form posts back to the same servlet. Upon receiving the POST request, the servlet will use the name of the uploaded file and the content of the part that contains the short description build a `Photo` object and add it to the map. At the same time, it copies the uploaded image to two places, the appropriate sub-folder of `Temp`, and the `images/storedphotos` web sub-folder.

4. Add a photo view servlet. This servlet will have a `doGet()` method that returns a form that has a HTML `<select>` element and a submit button. The form posts back to the same servlet, allowing `doPost()` to retrieve the name of the image file from the request. The `doPost()` method then uses information in the map to get the corresponding short description. This information is embedded into a dynamically-generated HTML file that also contains an `<img>` element whose source is the name of the file (prefixed with the appropriate folder).

You should use string variables for HTML templates as demonstrated in class. Here is an example of a HTML template for displaying the name of a photo, its image, and its short description

```
<!DOCTYPE html>
<html>
```

```
    <head>
        <title>Online Photo Album supply a title</title>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width,
                                       initial-scale=1.0">
        <style>
            body {width:600px;}
            h4, h5 {
                text-align: center;
            }
            IMG.displayed {
                  display: block;
                  margin-left: auto;
                  margin-right: auto;
            }
            a {margin-right : 10px;}
        </style>
    </head>
    <body>
        <h4>Welcome to the Online Photo Album</h4>
        <p>Where your memories are preserved in Cyberspace</p>

        <h5> %s </h5>

        <img src ="images/storedphotos/%s" alt="image here"
                                          class="displayed"/>
        <p>
           %s
        </p>
        <br/>
        <a href="index.html" style="float: left">Home</a>
        <a href="PhotoViewServlet" style="float: left">
            View a photo
        </a>
        <a href="PhotoUploadServlet" style="float: left">
            Upload a photo
        </a>
    </body>
</html>
```

This assignment is due Monday of Week 6.