Enriching Photo Album System with JavaScript

CSCI4140 - Individual assignment #2

Version 1.2: 2014 March 14, 4:40pm

Abstract

In the previous assignment, you have finished the basic implementation of a basic online photo management system. In this assignment, you are going to add JavaScript into the page so that you can add more fancy functions to the system.

1 Your Task

Your task is to add the following features.

- Using DOM manipulation to display the raw image (not the thumbnail), instead of opening the image by changing the browser's location.
- Using event handling to drag-and-drop the displaying image and also to resize the displaying image.
- Using drag-and-drop events and the `XMLHttpRequest` object to upload the target image.

2 Development and Demonstration Environment

We require you to work with the following constraints:

- On the server side, you are allowed to implement the system using any programming language and packages you love. However, we only provide supports on **Perl** and **PHP** only.
- On the client side, you are allowed to use **JavaScript**, **CSS**, and **HTML** only. Also, you should not use any framework or JavaScript libraries to implement the task. The UI and related JavaScript should be written by your own. The only set of external codes that you can use all the codes provided by us during the lectures and the tutorials.

- You are recommend to extend your basic system from Assignment 1. If you think that your system is too hard to extend, you are welcome to re-write your codes used in Assignment 1.
- Again, Our primary development environment is *OpenShift* - `http://openshift.com`. Since we only provide supports on PHP and Perl, we recommend using the Perl cartridge with the MySQL database support as well as the PHP cartridge with the MySQL database support as your development platform. Please make sure that your finalized work can work on OpenShift even though you are not using OpenShift as the development environment.
- For the interface design, you are more than welcome to write **ugly** ones. Remember, beautiful and fancy interfaces do not result in high marks.

Demonstrations will be our only grading process.

- During the demonstration, we will restore your submitted git repo into your OpenShift cartridge.
- For the client side environment, your system should be able to run (1) Firefox version 4.0 or above, or (2) Google Chrome 32.0 or above. Last but not least, during the demonstration, we may be using Mac, Linux or Windows as the client platform.

3 Task Description

In the following, we describe the **three tasks** involved in this assignment.

3.1 New album display layout

The first task is to make the album layout fancier. To lift your burden, we remove some features done in Assignment 1. Note that the set of image formats used in Assignment 1 remains the same.

3.1.1 Removed features

- No login interface was needed. Every user will have the privilege to view and to update the album.
- You are not required to let the user to change the order of the photos. The display must be sorted by the upload time in a descending order, i.e., show the latest first.

- You are not required to let the user to change the thumbnail array dimension. For each row, there are four thumbnails only.
- You are not required to break the album into pages; you should show all the thumbnails in one page.
- You are not required to show the name of each photo any more.
- You are not required to check for duplicated files in the server. If a user is uploading a file with a duplicated file name in the server storage, then the uploading file overwrites the existing one.
- You are not required to keep the "*Installation Script*" in the system although it is always good to keep it there for the ease of system installation.

3.1.2 New layout

Figure 1 shows the new album layout. The album layout is still showing the thumbnails of the original photos. Yet, you have to implement **three** new (or updated) features.

1. **Mouseover requirement**. When the mouse cursor is over a particular image, the **border** of the image will be highlighted. In terms of CSS, the following shows the suggested style for the mouseover and the mouseout events.

`mouseout`	`mouseover`
`border: 0px;`	`border: 2px solid red;`

   Of course, the above is only a suggestion and you can always choose your favorite style.

2. **Image button requirement**. Referring to Figure 1, there are two images, namely the "`Delete image`" and the "`Edit image`", showing on top of the photo **when the mouse cursor is over the photo**. The images act as buttons which trigger the following functions:

   - **"`Delete image`" – delete a photo**. Similar to Assignment 1, we allow the deletion of a selected photo. In Assignment 2, the trigger is the "*click*" event on the "`Delete image`".

     (a) When the "`click`" event is triggered, you have to show a confirmation dialog, i.e., "`window.confirm`", asking for the confirmation from the user.

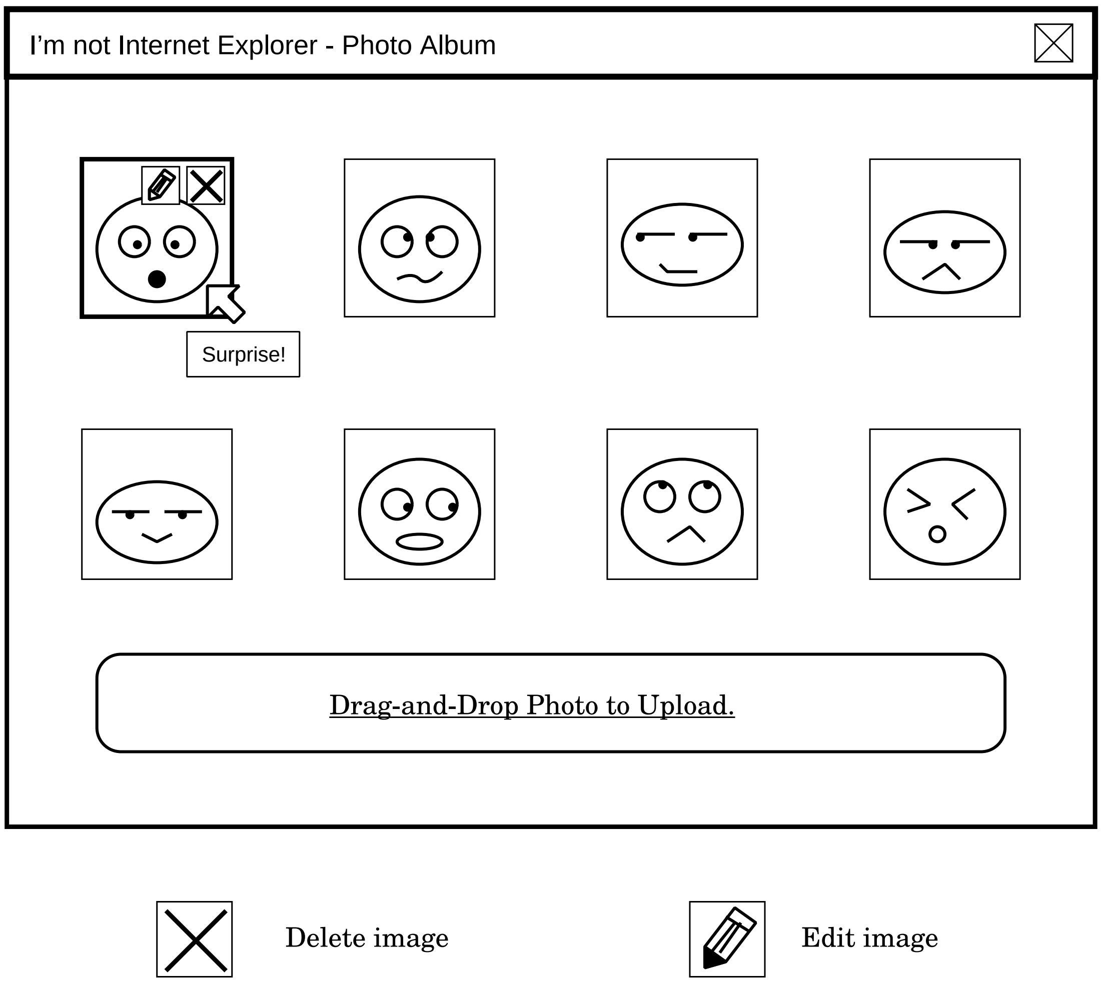
I'm not Internet Explorer - Photo Album
Surprise!
Drag-and-Drop Photo to Upload.
Delete image
Edit image

Figure 1: The rough display of the new album layout design.

(b) If the user confirms, you have to use an asynchronous `XMLHttpRequest` call to invoke the actual deletion on the server side.

(c) Last, the deleted photo should disappear from the album display.

- "`Edit image`" – **editing the photo description.** Similar to Assignment 1, every photo should have a description (see "*Surprise!*" in Figure 1). In Assignment 2, you have to provide a way for the user to update the description of each photo.

  (a) When the image is click, you have to show an input dialog, i.e., "`window.prompt`", asking for the new description.

  (b) Then, you have to use an asynchronous `XMLHttpRequest` call to have the actual change on the server side.

(c) Remember that the same set of restrictions on the description as those in Assignment 1 still holds. Last, the change should be reflected on the client side.

For the above both buttons, you should take care of the following three important requirements:

Important requirement #1. The "`click`" event should not be propagated further.

Important requirement #2. There should not be any page loading. The album should not be reloaded nor leave the current page, meaning that **every action should be completed using DOM scripting**.

Important requirement #3. Last but not least, when the mouse is over either the "`Delete image`" or the "`Edit image`", the border of the image will still be highlighted.

Not a requirement. Both the "`Delete image`" and the "`Edit image`" can be any images that can correctly hint us their usages. E.g., an image showing the string "`Del`" is good enough to hint us that it is the "`Delete image`".

3.2 Displaying original image

Unlike Assignment 1, which shows the original image by changing the browser's location, you are now required to display the original image by using two layers (or div objects). Figure 2 shows the expected design of this required feature.

3.2.1 The background layer

The **background layer** is a "`div`" element and it is the **second top-most layer** in the screen.

- This layer acts as a cover over the screen. You just need to have a best-effort implementation on the coverage: to cover 100% of the screen display. **Hint:** the variables "`window.innerWidth`" and "`window.innerHeight`" should be helpful.

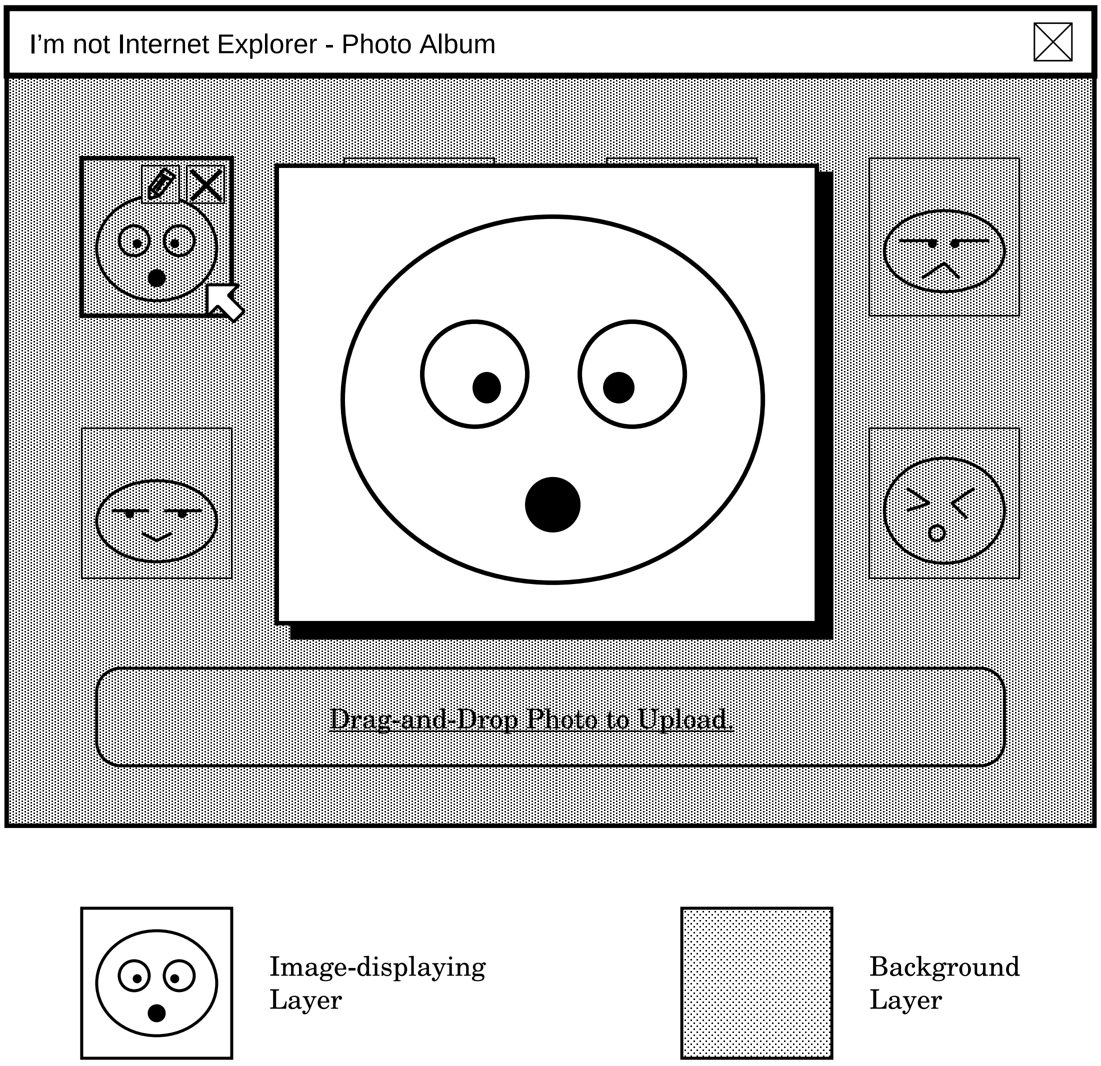
I'm not Internet Explorer - Photo Album
Drag-and-Drop Photo to Upload.
Image-displaying Layer
Background Layer

Figure 2: The rough display of the way to show the original image.

Note that you may fail to have a 100% coverage when the vertical and the horizontal scrollbars appear. So, during the demonstration, we will try to avoid having the scrollbars appeared. (Of course, although it is not a requirement, it would be nice to have the scrollbars taken into account.)

- This layer should be shown in the display regardless of the scrolling position. **Hint:** the variables "`window.PageXOffset`" and "`window.PageYOffset`" should be helpful.

- The mouse wheel should not be working when this layer is shown. **Hint:** the "`DOMMouseScroll`" event (for Firefox) or the "`mousewheel`" event (for Google Chrome) should be helpful.

- The user would not hit buttons that would trigger page scrolling when both layers are shown. That means "`Page-Up`" nor "`Page-Down`" button would **not** be pressed by the user.
- Any mouse click on this layer will cause this layer to be disappeared, but the click **should not propagate** to other components of the page.
- You have the freedom to choose the display style of the background layer, e.g., the color, the border, the opaqueness, etc.

3.2.2 The image-displaying layer

The image-displaying layer is another "`div`" element, and it should be the **top-most layer** in the screen. **Hint.** "`style.zIndex`" is an important JavaScript styling.

- This layer should contain the target image to be displayed, using the "`img`" tag.
- **Fit-to-screen requirement**. This layer should fit the browser display with the following two requirements.

  1. **Fit-to-screen requirement #1**. The starting width and the staring height of the image must be smaller than or equal to 70% of the width and the height the browser display, respectively.

     If any one of the dimensions is too large, then you have to reduce the image and to keep the concerned dimension under the 70% restriction. Note that we will test the implementation of this requirement using *human eyes only.*

     Hint: Variables "`[element].offsetHeight`" is read-only while "`[element].style.height`" can be updated. Yet, the above applies to both the width and the height of an element.

  2. **Fit-to-screen requirement #2**. The starting position of the image layer must be at, approximately, the center of the browser display.

Hint: Variables "`[element].offsetLeft`" and "`[element].offsetTop`"are read-only while "`[element].style.left`" and "`[element].style.top`" can be updated.

- **Drag-and-drop requirement**. This layer should be able to be drag-and-drop: the user drags the image pressing down the mouse button on the image body. You have to handle the following cases properly:

  – What if the cursor goes beyond the browser window? **Answer**: any part of the image layer should not be dragged beyond the browser.

  – What if the user starts the drag, but releases the mouse button outside the browser window? **Answer**: the image layer should not be dragged beyond the browser and the mouse release action should drop the image.

  – What if the user moves the mouse very fast? **Answer**: the image layer should be able to catch up with the fast movement properly.

  For the above three requirements, we will provide a clear demonstration in the lectures.

	Corners				**Dimension**		**Cursor**
Trigger	Top-left	Top-right	Bottom-left	Bottom-right	Width	Height	**Type**
N	—	—	fixed	fixed	fixed	—	n-resize
NE	—	—	fixed	—	—	—	ne-resize
E	fixed	—	fixed	—	—	fixed	e-resize
SE	fixed	—	—	—	—	—	se-resize
S	fixed	fixed	—	—	fixed	—	s-resize
SW	—	fixed	—	—	—	—	sw-resize
W	—	fixed	—	fixed	—	fixed	w-resize
NW	—	—	—	fixed	—	—	nw-resize

Table 1: When the user drags on a particular trigger, some of the attributes of the image change while others are fixed. This table shows what the fixed attributes are.

- **Resizing requirement**. The system should allow users to resize the displayed image. Figure 3 shows the concept. We summarize the resizing requirement in Table 1 together with the

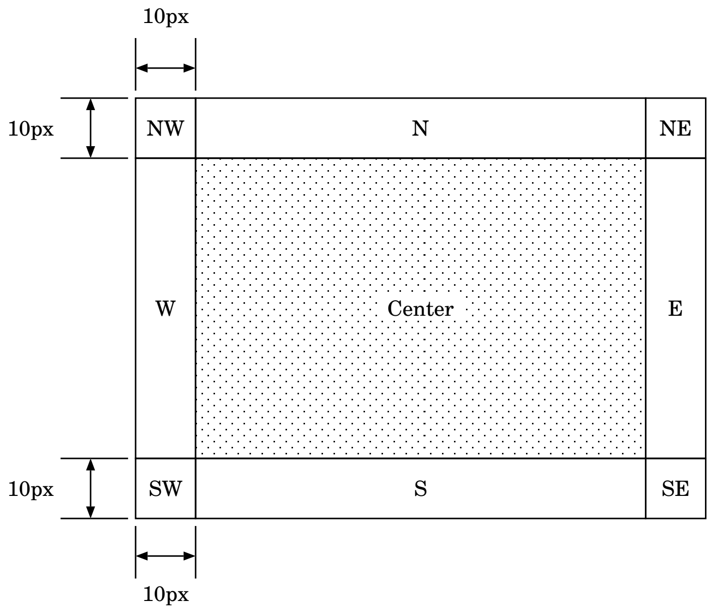

Figure 3: Every zones, except "Center", triggers the resizing of the displayed image with the "`mousedown`" and "`mouseup`" event.

following two points:

Note #1. According to Table 1, the mouse cursor should be showing different styles when it is over different zones. **Hint:** in JavaScript, you have to set the appropriate value for the variable "`[element].style.cursor`".

Note #2. Whether the image covers (a) the "`Center`" area in Figure 3 only or (b) all the nine zones is up to your design. However, it is recommended that you have to understand the pros and the cons before you make your decision. The lecturer would suggest you to implement this feature with a **3 x 3 table**, with the image displayed in the center of the table.

Note #3. During resizing, the user will not move the move outside the browser window so that we can ease your implementation.

Last but not least, you are **not required** to handle the case when the window is resized during both the background and the image-displaying layers are shown.

3.3 Drag-and-drop uploading feature

The way that Gmail and YouTube allow you to upload files using drag-and-drop are are very "*sexy*". you are going to implement this feature onto your album system.

Referring back to Figure 1, you are required to implement a drag-and-drop zone there; the position of the zone is not important. This can be realized with the HTML5 `Drag-and-Drop` and `File` APIs. The technical details will be covered during the tutorials only. Note that we will only upload one file at a time.

In the following, we briefly describe the flow involved. Basically, you have to handle three things.

1. You should use a "*file-type*" input object as taught in the tutorials.
2. The upload will start when the user drops the files onto the "*file-type*" input object.
3. The system should show a progress bar representing the upload progress. A suggested progress bar implementation is as follows.

   ```
   <p style="background:red; height:10px; width:0%;"></p>
   ```

   By changing the "`width`" property (0% in the example) from time to time using JavaScript, you can emulate a progress bar.

   Also, you may use the "`<progress>`" tag introduced in HTML5. For details, please visit `http://www.w3schools.com/tags/tag_progress.asp`.
4. By the time the upload has finished, the server should determine if the uploaded file is an image or not, which is the same task in Assignment 1.
5. If the file format is correct, the server side should store the image and the newly-uploaded photo should appear in the album.
6. Else (if the file format is incorrect), an appropriate error message should be shown.

Unlike Assignment 1, according to the above design, there is no way for the user to input the file description. Therefore, the description is **by default empty** until the user inputs one using the "`Edit image`" button introduced earlier.

Important requirement. For the above task, the album should not be reloaded nor leave the current page, meaning that **the layout update should be completed using DOM scripting**.

3.4 Updating without reloading (with multiple browsers)

Again, let us take Gmail as an example in explaining this feature: the inbox of Gmail is updated automatically without reloading the page. The idea is simple and is described as follows.

1. Page has finished loading.
2. Register a periodic timeout event.
3. When the timeout is reached, use XHR to read the status of the album database.
4. Display the up-to-date album if changes are detected.

The update of the album display is required only when the user:

- is not displaying original image;
- is not deleting an existing image;
- is not editing a photo's description;
- is not uploading a photo;
- is not having the mouse over an image;

In other words, the change is required only when the album is "idle". Note that the change to the album database should be done by another browser. Of course, such changes include:

- deleted an existing image;
- edited a photo's description;
- uploaded a new photo;

Last but not least, you have to provide a button (or a link) for user to turn this update feature on or off. Note that this feature is not draw in Figure 1 on page 4.

Note. The update on/off feature is important for you in debugging.

Milestones

New album layout – 30%
Are the new layout correctly displayed with four thumbnails in each row?
Are the image buttons shown correctly?
Can a user delete any photos?
Can a user edit any titles?
Displaying original image – 40%
Are all the layers there and correct?
Position of the background layer
Fit-to-screen requirement #1
Fit-to-screen requirement #2
Drag-and-drop
Drag-and-drop with bound checking
Correct resizing [N, E, S, W]
Correct resizing [NE, SE, SW, NW]
Removing the background layer
Drag-and-drop Upload – 15%
Is the progress bar there?
What if a PDF file is uploaded?
Is the layout updated correctly?
Update without reloading – 15%
Can the update be triggered by deleting image?
Can the update be triggered by uploading an image?
Can the update be triggered by changing an image's description?
Is the layout updated correctly?

Deadline - 23:59, April 11, 2014 (Fri).

Change Log

Time	**Version Update**	**Details**
2014 March 13	NIL	Release version 1.0.
2014 March 13, 8:50pm	1.0 → 1.1	In the second bulletin on Page 2, the two cartridges should be "Perl cartridge with the MySQL database support as well as the PHP cartridge with the MySQL database support".
		In Figure 2 of Page 6, the "Delete Image" and the "Edit Image" should be covered by the "Background Layer".
2014 March 14, 4:40pm	1.1 → 1.2	In the first bulletin on Page 7, the second sentence should be: That means "`Page-Up`" nor "`Page-Down`" button would **<u>not</u>** be pressed by the user.