Assignment 4: FlickrStorm

EECS 493: UI Development (Ackerman)
DUE Sunday 12/8, 11:59pm

Spec changes

- (191124) small correction to number of bonus points

We're going to create a browser for flickr images. In your imagination, you can think of yourself creating a small product, a dedicated raspberry pi or a cellphone, attached to your television (or digital picture frame :) that rotates images from flickr every hour or two, similar to chromecast when not casting.

You will put up two different kinds of collections:

- "Interestingness" is a special tag given by flickr.com to some uploaded pictures. There is a new interestingness collection created every day, and you can get to old ones.
- Flickr users create galleries, which are collections of pictures they think are aesthetically similar.

Your flickr browser will have 7 functions:

1. It will start with rotating through an interestingness collection from yesterday, showing the first 10 photos using the flickr api. You will use a bootstrap carousel to do the displays and rotations.

2. The user can "advance" to a new set of 10 interestingness photos by pressing "3". These will be the first interestingness images from a new day. Because the sets are historical, "advance" is actually going to a day earlier.

3. The user can, on any interestingness image, go to a gallery for that image by pressing "5". There may be any number of galleries for an image, including none. You will have to handle the situation where there is no gallery (discussed further below). If the user presses 5 in a gallery view, it is ignored. You will display the first 10 images from the first gallery in the carousel.

4. The user can go back to the beginning interestingness set (ie, for the first day) by pressing "1". It should be the same as (1). If you press 1 while in state (1), it will just restart the set from the first image.

5. The user can return to the interestingness set he/she was browsing when a "5" began looking at a gallery by pressing "2". You can, if you wish, just go back to the first image in that set, rather than go back to the image that was being shown.

6. The user can stop the carousel from rotating by pressing "7".

7. The user can restart the carousel's rotation by pressing "9".

Caveats:

- Any set of photos should be displayed in order. The first json element should be the first item in the carousel.

- You must handle errors by having catch clauses where appropriate. If there is an ajax error, you can display an image of a Michigan M (or rotate among them – your choice). The application must, however, stay active, allowing the user to pick another action. We will inspect your code enough to make sure you did this.

- You must use promises where appropriate. This assignment is mostly an exercise in promises and ajax, reinforcing what you learned in assignment 3. We will inspect your code enough to make sure you did this.

- The carousel can be painful. Our recommendation is that you leave the timer interval at 5 seconds (the default). Shorter times seem to occasionally freeze some versions. Feel free to play with the timer interval while debugging, but caveat emptor.

- The carousel should rotate on all action (except pause obviously). To start the rotation, the web browser window may need to be active (i.e., the user has clicked in it). This is acceptable.

- There will be a label under the carousel, partially to make grading easier. Note it changes according to what is being displayed.

- You can use js, jquery, bootstrap, and/or vue. The example videos were produced with jquery.

- Writing UI code can take a long time (as you may have already surmised). It sometimes takes forever to get a display to look right. UI components can also be quite brittle, requiring considerable effort to figure out the "right" way(s) to use them. If you find yourself struggling in this way, it's not that the frameworks we're using or the API are especially buggy or broken; it's in the nature of UI code and coding.

- GSI/IAs and graders will grade your submission in the latest version of Chrome with online access.

- Submit one .zip file. You can create separate html, css, and js files, or you may combine them. Make sure they're readable in either case. Include a README.txt with any design decisions you made, any external code you used, and the location in your code of where API keys need to be substituted.

Getting started:

- The first thing you need to do is to get an API key. You can do this at:

  https://www.flickr.com/services/api/misc.api_keys.html

  You will need to create an account by giving them an email address. Telling them that you're creating an assignment for a class seems to be okay. Acceptance seems to be immediate, but give yourself some time.

  If you do not provide us with your working key, we will be substituting our key for your key, so in your README.txt file, make sure you tell us all line numbers where a key is used.

- You can find reasonable documentation at https://www.flickr.com/services/api/

  One of the nice things about the flickr API is that for each API call, they have an API Explorer that allows you to see how to form the call and to see real returns. Note that the bottom of the API Explorer page, after you fill out the form, is how the url looks for the call.

- The flickr API appears to be dead stable, so if you're having a problem, it's likely you.

- Do NOT pound the flickr servers, or you will get rate limited. You may wish to get a smaller number of images while debugging. We've not run into rate limiting or rate throttling, but when the entire class starts using the API, mileage may vary. Rate limiting will not be considered an excuse for lateness.

- Your application may vary slightly from the video because the flickr photos and galleries are being updated constantly. Your beginning interestingness set should be yesterday's, whenever yesterday is. (We are requiring yesterday, because the flickr interestingness API tends to return an empty set right after midnight.) It is required that your application not being hardwired for dates and contents.

Starter code/hints:

- You will be using the API calls flickr.interestingness.getList, flickr.galleries.getListForPhoto, and flickr.galleries.getPhotos, as well as needing to create urls for the photos themselves. The urls look like:

```
let src = "https://farm" + photo.farm + ".staticflickr.com/" + photo.server + "/"
+ photo.id + "_" + photo.secret + "_c.jpg";
```

  where you will get the farm, server, id, and secret from the json returned by flickr.interestingness.getList and flickr.galleries.getPhotos. The "src" url can be used in creating the items for the carousel.

- You can change sets in the carousel with:

  ```
  $('.item').remove();
  ```

  and then recreating the items (the images) in the carousel. There is a starter fragment with this.

Grading:

- basic carousel running with label in a responsive (bootstrap) manner (5)
- carousel displaying an initial interestingness set with the correct date and label (15)
- carousel displaying a new interestingness set with the correct date and label (10)
- carousel displaying the correct gallery photos for a photo with correct label (20)
- user can return to the previously seen interestingness set (10)
- user can return to the initial interestingness set (5)
- user can pause and restart (5)
- use of promises (15)
- use of catch clauses in promises and error handling (15)

- extra credit suggestions (max of 5):

- use of a vertical nav bar (5)
- removal of the arrow bars at the side of the carousel (1)
- more aesthetically pleasing bootstrap-style label (3)
- creative use of other flickr API calls (up to 5)