COMP 3512
Assignment #1: Single-Page App *(version 1.0)*

Due Sunday October 27, 2019 at midnightish
Changes in yellow: last changes October 7, 2019.

Overview

This assignment provides an opportunity for you to demonstrate your ability to generate a dynamically updateable single page web application using JavaScript. You will be creating something similar to the final exercise from your JavaScript 3 lab. That lab made use of a small subset of data in a small JSON file that contained all the information you needed. This lab uses a more realistic API.

Beginning

Starting files will be at: `https://github.com/funwebdev-3rd-ed/case-travel.git`

Submitting

For this assignment, you will simply submit your source files (but not your travel images) to the regular submit drive (I: drive). Name your submit folder as follows: *`username_assign1`*, where *username* is your actual user name. This folder should contain your content. One of the requirements is to host your images in a Google Cloud Storage bucket, so there should be no need to copy all the travel images to the submit drive. If you don't implement the Google Cloud Storage bucket, then you will need to submit the image files, but you will right away annoy me by making me wait while I copy your content from submit to local computer for marking. If you are using the GCP storage, make sure you do NOT submit all the travel images!

Grading

The grade for this assignment will be broken down as follows:

Visual Design	20%
Programming Design	15%
Functionality (follows requirements)	65%

Data Files

Images for the travel images have been provided. Data will be provided in two external APIs.

Requirements

This assignment consists of a single HTML page (hence single-page application or SPA) with the following functionality:

1. Your assignment **must** have just a single HTML page which **must** be named `index.html`.
2. I expect your page will have significantly more additional CSS styling compared to `lab10-test02.html`. If you make use of CSS recipes you found online, you must provide references (i.e., specify the URL where you found it) via comments within your CSS file. **Failure to properly credit other people's work in your CSS will likely result in a zero grade.**
3. Your layout should work at desktop size and at mobile size.

4. You must write your own JavaScript. That is, no jQuery, no React, no other third-party JavaScript libraries. You have all the knowledge you need from the three JavaScript labs to complete this assignment. If you do find yourself, however, making use of some small snippet of code you found online (say more than 4-5 lines), then you must provide references (i.e., specify the URL where you found it) via comments within your code. **Failure to properly credit other people's work in your JavaScript will likely result in a zero grade.** There is no need to credit code you found in the lab exercises.
5. The travel images can be found at the github repo for the 3rd edition of the travel case (`https://github.com/funwebdev-3rd-ed/case-travel.git`). To simplify marking (and also to give you a little experience with it), I would like you host your images in a Google Cloud Storage bucket. Use Standard Storage and set the permission to allUsers for the bucket. Upload your travel image folders (you only need to upload the square and medium folders). The URL for any given uploaded image can be examined in the GCP display: you will make use of this URL later in your `<img>` tags.
6. Most of the functionality in the app can be found in the two sketches shown on the next few pages. Each of these is described in more details below. The sketches provided illustrate a sample layout. You can completely change the layout. The first shown below is the **Default View**.

7. **Header**. The page title should be COMP 3512 Assign1. The subtitle should be your name.
8. **Country List**. Displays a list of countries. The URL for the API is:

   `http://www.randyconnolly.com/funwebdev/3rd/api/travel/countries.php`

   This API takes an optional parameter, which is described as follows:

   - No parameters - returns all countries for which there are images
   - iso=ALL - returns a list of all countries
   - iso=[value] - returns just the country with the specified ISO value
     - e.g., ISO=CA

The country list should always be sorted alphabetically on name. Initially, display all countries.

To improve the performance of your assignment, you must store the list of countries in local storage after you fetch it. Your page should thus check if country data is already saved in local storage: if it is then use it, otherwise fetch it and store it in local storage. This approach improves initial performance by eliminating an early fetch in future uses of the application.

9. **Country Filters**. Allow the user to easily filter the list of countries. User should be able to view countries from a specific continent, view only countries that have images, and allow the user to type in the first few letters of the country name. Also provide way to remove filters (or return to all countries being displayed). For each of these, the list should dynamically update. The filters should be hide-able as well (though initially should be visible). When the user clicks on a country, the city list should be updated. Don't be scared of using icons instead of text. Assume these filters are mutually exclusive (only one can be active).

10. **City List**. Displays a list of cities for the selected country. If the user hasn't selected a country, then display an empty list. The URL for the API is:

    `http://www.randyconnolly.com/funwebdev/3rd/api/travel/cities.php`

    This API takes an optional parameter, which is described as follows:

    - No parameters - returns all cities for which there are images
    - id=ALL - returns a list of all cities
    - id=[value] - returns just the city with the specified ISO value
        - e.g., id=264371
    - iso=[value] - returns just the cities from the specified country
        - e.g., iso=CA

    The city list should always be sorted alphabetically on name. Initially, display all cities in the selected country. Note: to simplify deployment later when we work with PHP, the city table only contains cities from countries for which there are images. Thus, most countries have no cities in this small data set. Your assignment must be able to handle this elegantly.

11. **City Filters**. Allow the user to easily filter the list of cities. User should be able to view only cities that have images, and allow the user to type in the first few letters of the city name. Also provide way to remove filters (or return to all cities in the country being displayed). For each of these, the list should dynamically update. The filters should be hide-able as well (though initially should be visible). Assume these filters are mutually exclusive (only one can be active).

12. **Country + City Details.** When the user clicks on a country in the country list, display its information (name, area, population, capital name, currency, domain, languages, neighbouring countries, and description). I expect this to be nicely formatted and laid out sensibly. Every year students lose easy marks because they put no effort into the layout here.

    The languages field is a comma-delimited list of ISO 639-2 codes. You must instead display the full name (e.g., English instead of en). When your application first starts up, use the API at `http://www.randyconnolly.com/funwebdev/3rd/api/travel/languages.php` to retrieve a list of 135 language codes and names. You **must** store this in local storage to improve performance. Some languages in the comma-delimited list have a hyphen and regional information (e.g., `en-CA` for Canadian English eh). Ignore the hyphen and the region and just display the main language name. Some languages don't appear in the languages file: you can ignore those. Use the JS `Find` method.

The neighbours field is a comma-delimited list of ISO codes. You must display the full name of the neighbouring countries (United States and not US). Since you will have already stored the country list in local storage, you can simply search this array using the JS `Find` method.

13. **City Details.** When the user clicks on a city in the city list, display its information (name, population, elevation, timezone). I expect this to be nicely formatted and laid out sensibly.
14. **Country + City Map.** When the user clicks on a country in the country list, display a static map of the country (use a zoom level between 6-9) using Google Maps Static API. This is just an `<img>` tag but with the `src` attribute a link to Google's API URL. This will require using your API key.

    When the user clicks on a city in the city list, two things will happen: First, the static country map will be replaced with a new static country map that includes a marker for the city (again, you will use the Google Maps Static API). Second, you will display a static road map of the city (use a zoom level between 9-14).
15. **Travel Photos**. When the user clicks on a country or a city in the two lists, then your program will need to fetch the travel photos for the city or country from the following API:

    `http://www.randyconnolly.com/funwebdev/3rd/api/travel/images.php`

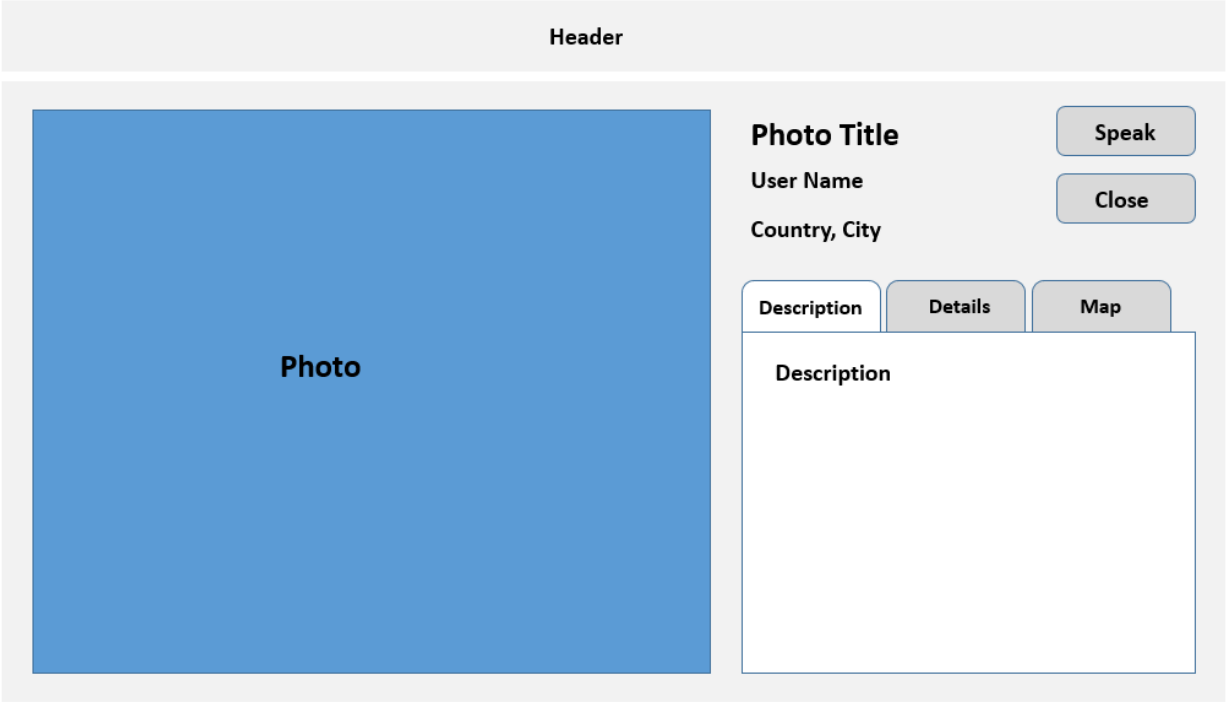
    This API takes an optional parameter, which is described as follows:

    - No parameters - returns all images
    - id=ALL - returns all images
    - id=[value] - returns just the single image with the specified id value
        - e.g., id=108
    - iso=[value] - returns just the images from the specified country
        - e.g., iso=CA
    - city=[value] - returns just the images from the specified country
        - e.g., city=5913490
16. Display thumbnails of the images, using either the `square75` or `square150` versions. You **must** use the `square75` version for mobile sizes, and the `square150` for desktop sizes. How? Use the `<picture>` element.

    Most countries and most cities have no images, so you need to be able to handle this with a message. These images are clickable, so be sure to change the cursor to indicate this.

    Your click handler here **must** use event delegation!

17. **Single Photo View**. When the user clicks on a photo thumbnail, everything in the default view (other than the header) will be hidden and replaced with just the Single Photo view of the single photo.



Display the `medium640` or `medium800` version of the photo (shown above as a blue box). You **must** use the `medium640` version for mobile sizes, and the `medium800` for desktop sizes.

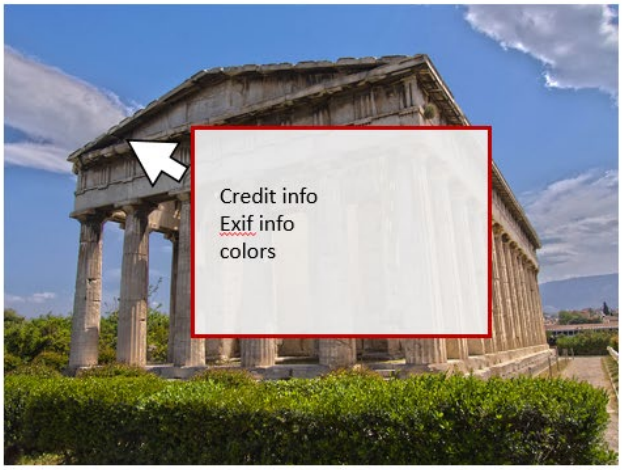
Display the following fields: user name, city, country, and photo title. I expect this to be nicely formatted, laid out sensibly, and works with either landscape or portrait photos.

The Speak buttons will use the speech synthesizer to speak the title. Don't be scared of using icons instead of text.

The Close button will hide this view and show instead the **Default View**.

18. **Description, Details, and Map**. In the sketch above, Description, Details, and Map are shown as tabs, though you could use hide / show boxes as well. The description tab (it should be showing by default) will show the photo description field. The map tab should show an interactive JavaScript Google Map API showing a map with the photo's lat+long location indicated via a marker. The details tab should show the exif information (model, exposure, aperture, focal length, iso), credit information, and colors information. The colors information is an array of five hex colors, corresponding to the five dominant colors in the image. You should display them as color boxes and also display the hex values as well.

19. **Hover Photo**. When user hovers over the image, display in a box (that has a little bit of transparency), the credit info, the exif info (model, exposure, aperture, focal length, iso), and the five dominant colors (shown as five colored boxes). When the mouse is moved outside the image, this box should disappear.



When user hovers over the image, display in a box (that has a little bit of transparency), the credit info, the exif info, and the five dominant colors (shown as five colored boxes).

Hints

1. Test the APIs out first in the browser. Simply copy and paste the URLs into the browser address bar. The JSON can be hard to understand because it doesn't have white space, so use an online JSON formatter (such as https://jsonformatter.curiousconcept.com) via copy and paste to see the JSON structure in an easier to understand format (or add a JSON formatter extension to your browser).
2. Remember that JSON and JavaScript are case sensitive. For instance, it is easy to type in `Id` and it won't work because the JSON field is actually `id`.
3. Your HTML will include the markup for both **Default View** and the **Single Photo View**. Initially, the later will initially use CSS to set its `display` to `none`. Your JavaScript code will programmatically hide/unhide (i.e., change the `display` value) the relevant markup container for the two views.
4. Most of your visual design mark will be determined by how much effort you took to make the two views look well designed. Simply throwing up the data with basic formatting will earn you minimal marks for this component.
5. Most of your programming design mark will be based on my assessment of your code using typical code review criteria. For instance, did you modularize your code using functions? Are your functions too long? Is the code documented? Do you have code duplication (you shouldn't)? Did you make use of object-oriented techniques? Are variables and functions sensibly named? Is your code inefficient (e.g., fetching the same data repeatedly)? Are you using outdated JavaScript techniques (e.g., inline coding, `XmlHttpRequest`, etc)? Is the code excessively reliant on found code from Stack Overflow, etc?

6. When constructing single-page applications in JavaScript, you may need to "insert" data into dynamic HTML elements that you modify/create in JavaScript. In HTML5, this is supported via the `data-X` attribute.

   For instance, in this assignment, you may need a way to determine the photo identifier of the photo that was clicked on in the list of photos. You can do this by using the `setAttribute()` method in JavaScript to set, for instance, an attribute named `data-id` whose value is the `id` field when dynamically generating the list of photos. Then, in the click event handler for each photo in the list, you can determine the unique identifier for the photo that was clicked (and thus later retrieve the photo object for that id) by using the `getAttribute()` method.

7. For the Speak buttons and the Close button, be sure to only add click event handlers for these buttons only once when the page loads (and not every time you display a painting).