COMP 3512
Assignment #1: Single-Page App

Version: 1.0 September 29
Due Sunday October 25, 2020 at midnightish

Overview

This assignment provides an opportunity for you to demonstrate your ability to generate a dynamically updateable single page web application using JavaScript. You can work in pairs or by yourself.

Beginning

You will be expanding the final exercise from your last JavaScript 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.

I feel foolish saying this in a third-year university course, but it is your responsibility to read all the assignment instructions thoroughly and carefully. If you are unclear about something, ask me. But before you do, read the material in question again!

Grading

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

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

Data Files

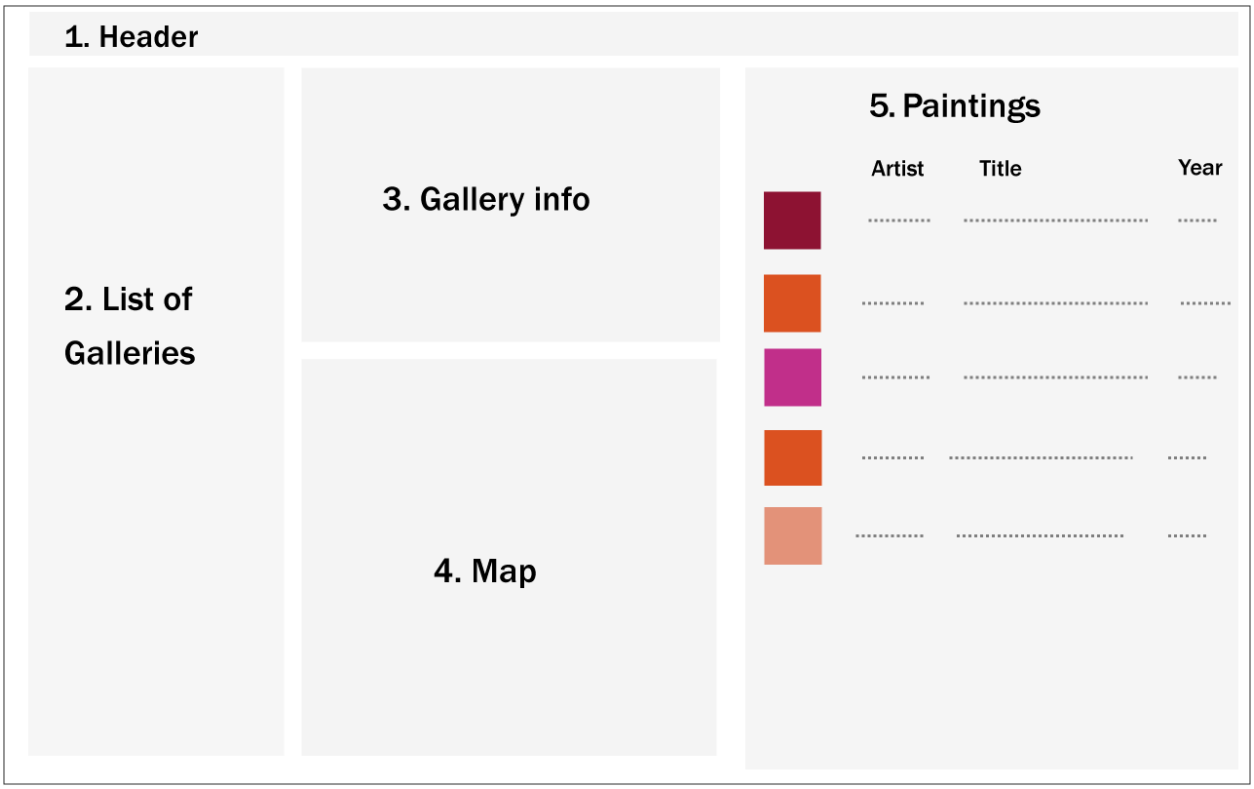
Images for the paintings have been provided.

Requirements

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

1. Your assignment should have just a single 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 could result in a zero grade.** You can make use of a third-party CSS library; if you do, be sure to indicate this in the header. Attractive styling with your own CSS will result in a higher style mark than if you use a third-party library; poor styling with your own CSS will likely result in a lower style mark than if you use a third-party library.

3. You must write your own JavaScript. That is, no jQuery, no Bootstrap, 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 (and I do mean small), then you must provide references (i.e., specify the URL where you found it) via comments within your .js file. **Failure to properly credit other people's work in your JavaScript will result in a zero grade.** Using ancient JavaScript found online will result in lower marks. There is no need to credit code you found in the labs or lectures.
4. Most of the functionality in the app can be found in the two sketches shown on the next few pages. The sketches provided are used to indicate functionality not the layout. You can completely change the layout. Each of these is described in more details below. The first shown below is the **Default View**.



5. **Header**. The page title should be COMP 3512 Assign1. The subtitle should be your name. If you used a third-party CSS indicate this somehow in your header.

6. **List of Galleries**. Just like in last lab, this should display a list of art galleries. Unlike the previous lab, in which you fetched the gallery and painting info within a single hierarchical JSON file, this assignment uses two discrete APIs: one for retrieving gallery information, the other for retrieving painting information. The URL for the gallery API is at present:

   `https://www.randyconnolly.com/funwebdev/3rd/api/art/galleries.php`

   This API returns just information about the galleries (no paintings). You must sort the galleries by the `GalleryName` property.

   The API will take some time to retrieve this data. Display a loading animation (there are many free animated GIF available) until the data is retrieved.

   When the user clicks on a gallery, then populate Gallery Info, Map, and Paintings sections with the data for the clicked gallery.

   The user should be able to toggle the visibility of the List of Galleries section (though initially it should be visible). Don't be scared of using icons instead of text. When the list is hidden, there should be more space available to display the other information.

7. **Gallery Info**. When the user clicks on a gallery in the list, then your program will need to fetch the paintings from the following API:

   `https://www.randyconnolly.com/funwebdev/3rd/api/art/paintings.php?gallery=xx`

   where xx is the `GalleryID` property for the clicked-on gallery.

   In this area, display the following information from the JSON: `GalleryName`, `GalleryNativeName`, `GalleryCity`, `GalleryAddress`, `GalleryCountry`, and `GalleryWebSite`. The `GalleryWebSite` should be an actual working link.

   The API will take some time to retrieve this data. Display a loading animation (there are many free animated GIF available) until the data is retrieved.

8. **Map**. Display a map using the gallery latitude and longitude properties at the same zoom level as the last lab's exercise.

9. **Paintings**. Display the paintings in the gallery. Display a smaller square version of the painting, artist last name, title, and year of work. The filename for the paintings image is found in the `ImageFileName` field. (See Painting Images below for more information about the images).

   The headings at the top of each column (Artist, Title, Year), should be clickable. When the user clicks on a column heading, the painting list will be redisplayed so that it is sorted on the value just clicked. When first displayed, sort by artist last name.

   The titles should be styled so that they appear as links in some way. When the user clicks on a title, then current content on page will be hidden and replaced with the Single Painting view, described next.

10. **Painting Images.** Painting images are on the Cloudinary CDN. The URL for a given painting is as follows:

    `https://res.cloudinary.com/funwebdev/image/upload/`***`SIZE`***`/art/paintings/`***`FILENAME`***

    e.g.,

    `https://res.cloudinary.com/funwebdev/image/upload/w_200/art/paintings/square/001020.jpg`

    where ***`SIZE`*** is `w_###` or `h_###` where ### is the width or height in pixels; ***`FILENAME`*** is the value of the `ImageFileName` property in the painting object array. If you don't provide a size, then you will get the original image, which is very large (often 2000 pixels wide or more).

    The URL for a square version of the painting is as follows:

    `https://res.cloudinary.com/funwebdev/image/upload/`***`SIZE`***`/art/paintings/square/`***`FILENAME`***

    e.g.,

    `https://res.cloudinary.com/funwebdev/image/upload/w_75/art/paintings/square/001020.jpg`

11. **Single Painting View**. When the user clicks on a painting, sections 2, 3, 4, 5 will be hidden and replaced with just the details view of the single painting:

12. For the Single Painting View, display the large version of the painting (shown above as a red box). Display the following fields: `Title`, `FirstName`, `LastName`, `Title`, `GalleryName`, `GalleryCity`, `MuseumLink` and (working as a link), `CopyrightText`, `YearOfWork`, `Width`, `Height`, `Medium`, and `Description`.

    You will need to display the dominant colors for this painting. This should be displayed as appropriately colored boxes. When the user mouses over one of these boxes, they should see both the color name and the hex for the color in a popup tooltip (simply use the HTML title attribute).

    Clicking on the image should show a larger version of the image (see below).

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

    Note: most of your styling mark will be determined here. I will be expecting that you've taken some care in how you style and position this information.

13. **Large Version of Painting**. When the user clicks on the painting image in Single Painting Vierw, display a larger version as a pop-up modal window. This is sometimes referred to as a lightbox effect. The easiest approach is to simply turn on the visibility of the larger image's div when the smaller version is clicked; when the larger image is clicked, then hide it. This approach makes an image request even if the user doesn't want to see it; a more efficient approach for a real site would dynamically construct the image and div only when the user clicks it. For this assignment, take the easiest approach.

Submitting and Hosting

Your assignment will need to reside on a working public server and your source code on GitHub. Since your site only has static assets, you can make use of any free static file hosting provider. Some possibilities:

- Github Pages
- Netlify
- Firebase Hosting

This step is going to take some time, so don't leave it till the very end. **Be sure to test it to make sure it works.**

When your hosting is working and the assignment is ready to be marked, then send me an email with the following information:

- The URL of the home page of the site on your hosting platform.
- The names of the other people in the group
- The URL of the github repo so that I can mark the source code. If your repo is private, then add me as a collaborator.

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.
2. Remember that JSON and JavaScript is case sensitive. For instance, it is easy to type in `PaintingId` and it won't work because the JSON field is actually `PaintingID`.
3. Your HTML will include the markup for both **Default View** and the **Single Page View**. Initially, the later will initially use CSS to set its `display` to `none`. Your JavaScript code will programmatically change the display value of the relevant markup container for the two views.
4. Most of the 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.
6. Before submitting, test every single bit of functionality and cross-check with this document to make sure it has all the functionality I've requested. Carefully read the specifications for each bit of functionality. Every year, students lose all sorts of marks because they didn't read this document carefully enough!!