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

Due Saturday October 20, 2018 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 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.

Beginning

Starting files can be found at: `https://github.com/mru-comp3512-archive/f2018-assign1.git`

Submitting

You will need to share your workspace in c9 with me once it is complete. To do so, click the Share button (it has a gear icon) in the upper-right part of the Cloud9 workspace. Invite me via my email (rconnolly@mtroyal.ca) and be sure to give me RW access (otherwise I can't run anything).

Grading

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

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

Data Files

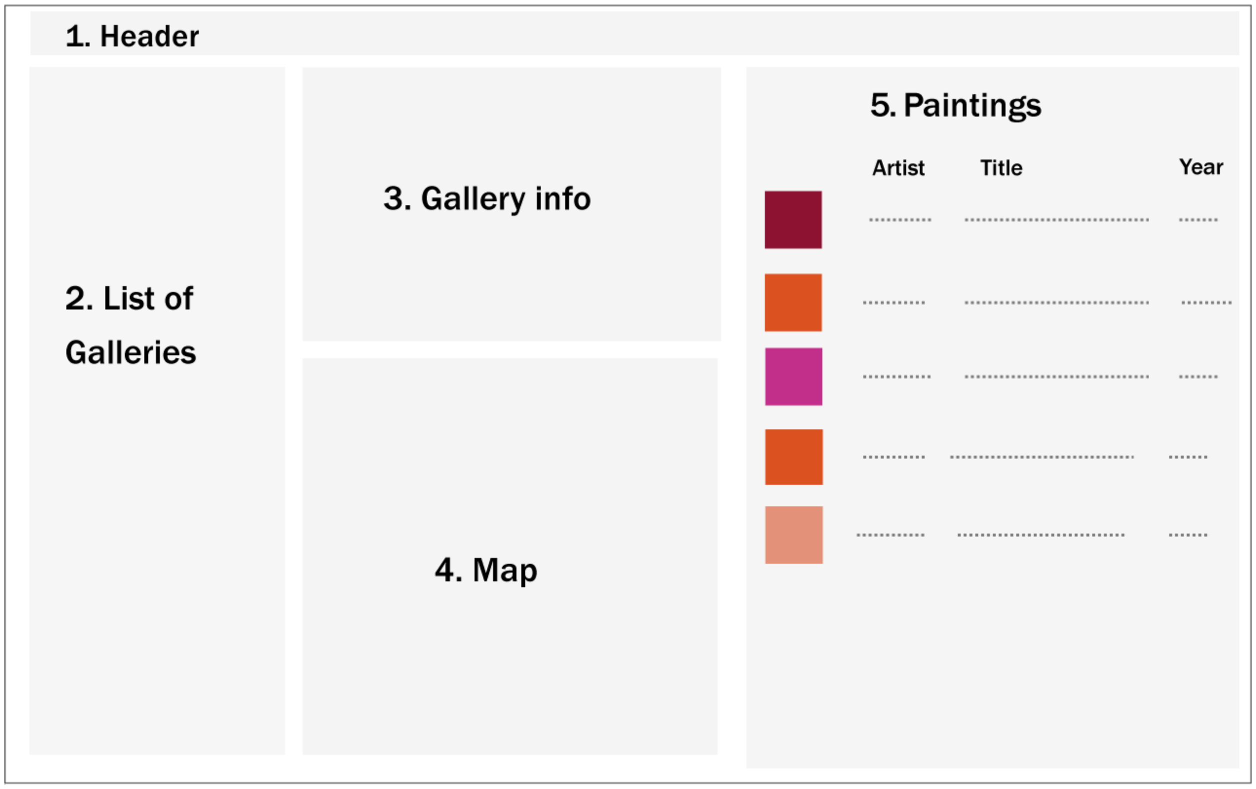
Images for the paintings have been provided. Data will be provided soon in two external APIs.

Requirements

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

1. Your assignment should 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. 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.

4. 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 first shown below is the **Default View**.



5. **Header**. The page title should be COMP 3512 Assign1. The subtitle should be your name.
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 will be:

   `https://www.randyconnolly.com/funwebdev/services/art/galleries.php`

   When using cloud9, the browser will want to use https versions of apis since cloud9 uses https. You can tell the browser to allow an http API via the Allow Unsafe Browsing icon when it appears; alternately, you can wait a few days for the https version to appear.

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

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

   To improve the performance of your assignment, you must store the list of galleries in local storage after you fetch it. Your page should thus check if gallery 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.

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/services/art/paintings.php?gallery=x`

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

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

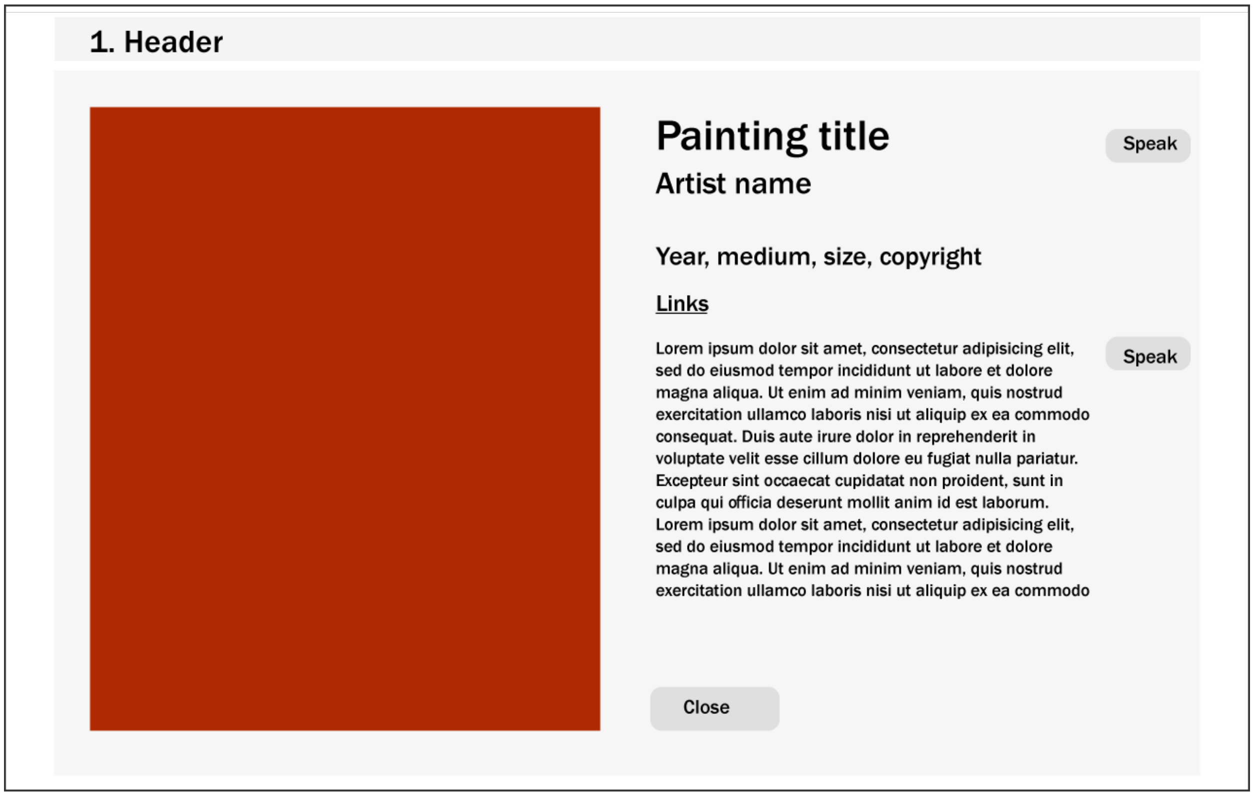
8. **Map**. Display a map using the gallery latitude and longitude properties at the same zoom level as the last lab's exercise.

9. **Paintings**. Display the paintings in the gallery. Display the thumbnails, artist last name, title, and year of work. The filename for the square-thumbnail image is found in the `ImageFileName` field (you will have to add the correct folder name and extension).

   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. To make your page more usable, you may decide to make the entire row clickable.

10. **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:



Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum. Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo

11. 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`, `MuseumLink` and `WikiLink` (working as links), `CopyrightText`, `YearOfWork`, `Width`, `Height`, `Medium`, and `Description`. I expect this to be nicely formatted and laid out sensibly.

    Hide the link if the property is empty/null. As well, since some of the URLs are lengthly, don't display the URL as the link label (use instead something like 'Museum Link' etc instead). Since some of these galleries may have modified their sites, it's possible that some links will be 404 errors.

    Some paintings have no description nor excerpt data.

    The Speak buttons will use the speech synthesizer to say either the `Title` or the `Excerpt`.

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

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 are 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 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. We didn't get a chance to discuss this yet in class, but when constructing single-page applications in JavaScript, you often 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 painting identifier of the painting that was clicked on in the list of paintings. You can do this by using the `setAttribute()` method in JavaScript to set, for instance, an attribute named `data-id` whose value is the `PaintingID` field when dynamically generating the list of paintings. Then, in the click event handler for each painting, you can determine the unique identifier for painting was clicked (and thus later retrieve the painting 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).