COMP 3612
Assignment #2: Single-Page App

Version: 1.0 Oct 15, 2021
Due Saturday November 13, 2021 at midnightish

Overview

This assignment provides an opportunity for you to demonstrate your ability to work with JavaScript. The application you will be creating is a text viewer of Shakespeare's plays.

Beginning

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!

Starting files can be found at:

```
https://github.com/mru-comp3612-archive/f2021-assign2.git
```

Grading

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

Visual Design	15%
Programming Design	15%
Functionality (follows requirements)	70%

Data Files

There are two sources of data. The first is a JSON file containing a list of Shakespeare's plays (as well as additional information for each play). The other source of data is an API that will provide the text for a specified play.

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. The assignment should have two main views: **List of Plays** and **Play Text**. When the program first starts, display the List of Plays view.
3. **List Of Plays view**. This data is provided in a JSON file (`plays.json`) containing a list of Shakespeare's plays (as well as additional information for each play). Exercise 8.14 in Lab 8 provides instructions for converting a JSON file into a string and then using `JSON.parse()` to convert it into a JavaScript object. **Make sure you are only parsing this file once!**

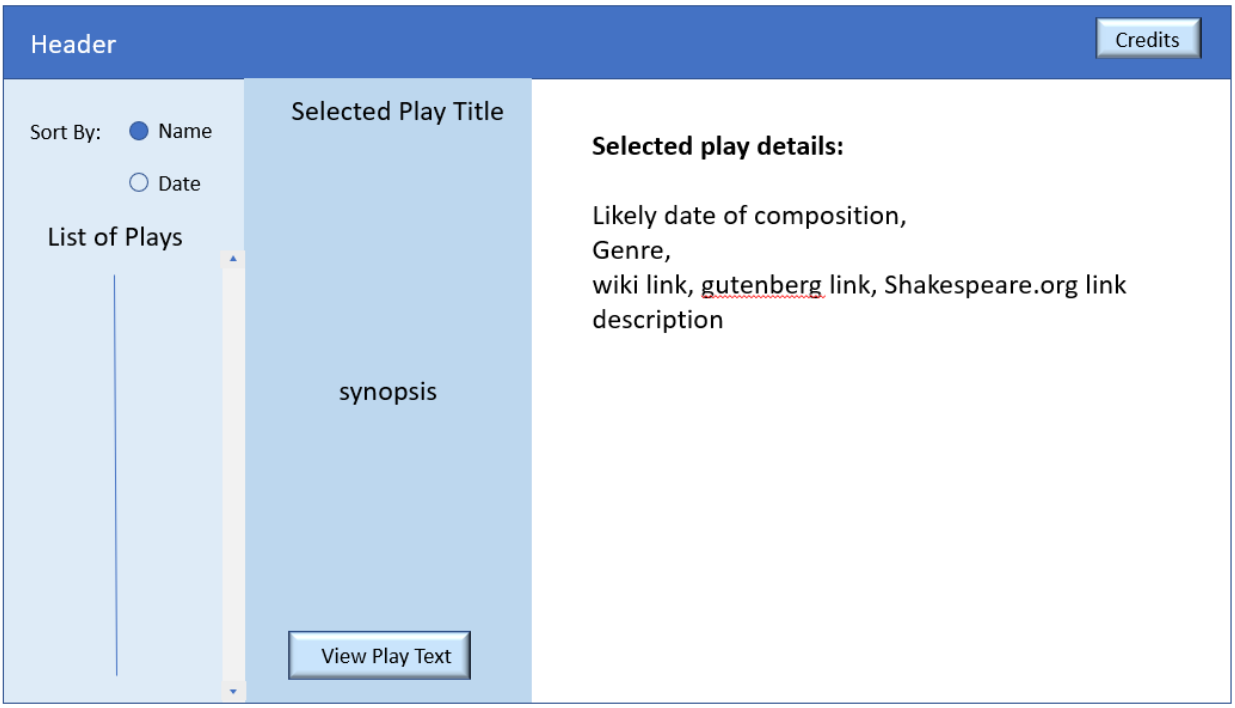
   Display a list of Shakespeare's plays on the left-side of the page based on this data. Initially, display the plays sorted by their name. Provide a sort option for the user so that the user can see the list sorted by name or by the likely date of composition. The list should refresh whenever the user changes the sort option. The list must be an unordered list; the list must be made scrollable by setting the `overflow-y` property of the container to `scroll`. Ideally, add an icon to those plays that have text available (see below).

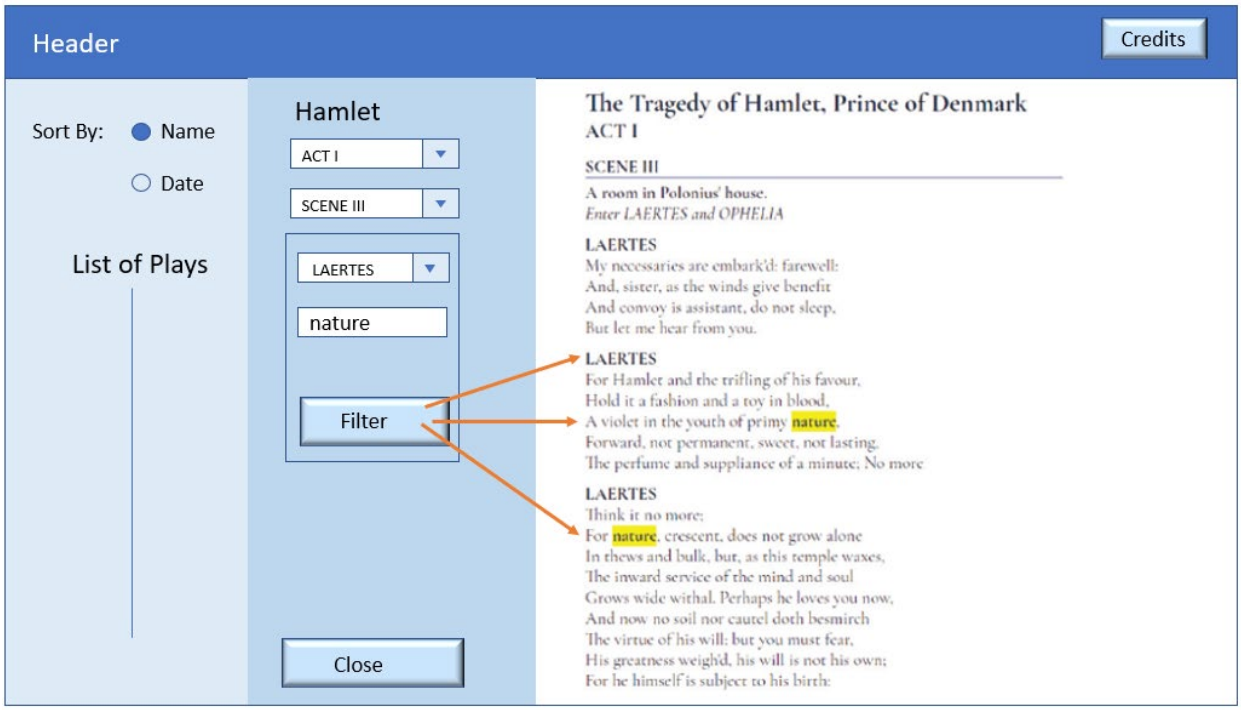
Add the play's `id` property to the `<li>` item by using the `data-id` attribute (e.g., `<li data-id="hamlet">Hamlet</li>`). Your event handler for the click event will need this `data-id` attribute value when it constructs its API fetch request.

When the user selects a play from the list (by clicking on it), display the selected play's title, date of composition, genre, wiki link, Gutenberg link, Shakespeare.org link, synopsis, and the description. The three links must be working hyperlinks. As well, when a play is selected, change its appearance in the list so that the user has some visual feedback (use `classList.add()` to add a class to the element).

Add a button that allows the user to view the play's text, if available. Only a few plays have text available. If the `filename` property of the play is empty, then there is no play text available; in such a case, simply hide the button (change its `display` property to `none`).

The image below roughly illustrates what the List of Plays view should look like. Be sure to style the play details nicely using contrast and spacing. When no plays have been selected yet (i.e., when page is first viewed), don't display anything in the second column (except a background color in the second column), and in the third column display "Select a play to see details". This message will disappear once a play is selected.





Examine `assign2.js` in the editor of your choice. In it, you will see the URL for the external API that will provide the play data. Examine this URL in the browser in order to see the structure of the data. A Shakespeare play contains multiple acts; each act contains multiple scenes. (To reduce the size of the downloaded files, not all acts and scenes have been included).

When the user selects a play, fetch the play data by adding the `data-id` attribute of the `<li>` for the play as a query string, as shown in the comments in `assign2.js`. When the fetched play is retrieved, populate the three other `<select>` elements from this data. Also populate the `<section id="playHere">`, `<article id="actHere">`, and `<div id="sceneHere">` elements with the first scene from the first act of the selected play.

Add event handlers to the `<select>` elements. They will change what part of the play is displayed.

The filter button will highlight all occurrences of the user-entered text in the play and only show the speeches from the specified player. This will require wrapping the text in `<b>` elements. You might find using `RegExp` helpful here for the text-matching.

If you examine `index.html` in the browser you will see that you have been provided with quite a bit of the styling and markup already for this view. Notice the containers for the fetched data in the `<aside>` and `<section>` elements. Notice the sample markup for the play data. This will be eventually commented out and replaced with JavaScript code that programmatically generates this markup.

When the user clicks the close button (or selects another play), the view will return to the **List of Plays** view.

5. Header. Add an icon/button/label to the header named Credits. When the user **mouses over** this icon, display your name(s) and course within a box. This message should be formatted uniquely and should disappear after 5 seconds (use `setTimeout`).
6. To improve the performance of your assignment (and reduce the number of requests on my server), you must store each retrieved play text data in local storage after you fetch it from the API (see Exercise 10.11 in Lab 10). Your page should thus check if this play data is already saved in local storage: if it is then use local data, otherwise fetch it and store it in local storage. This approach improves initial performance by eliminating this first fetch in future uses of the application. Be sure to test that your application works when local storage is empty before submitting (you can empty local storage in Chrome via the Application tab in DevTools).
7. 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 lectures and 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.

Testing

Every year students lose many easy marks because they didn't read the requirements carefully enough. When your assignment is getting close to done, I would recommend going through each requirement step and carefully evaluate whether your assignment satisfies each specified requirement.

Submitting and Hosting

Your assignment source code must reside on GitHub and reside on a working public server (in this case GitHub Pages).

GitHub Pages works in conjunction with git and github. You push your html/css/images to github repo; and then push to the host. The instructions for doings so can be found at:

`https://docs.github.com/en/pages/getting-started-with-github-pages/creating-a-github-pages-site`

It is up to you whether you want your pages to be public (available to the world) or private (available to only those with access to your github repo).

If you are using a private repo, you must add me as a collaborator!

I would strongly recommend getting your hosting to work a few days before the due date. It's okay if your assignment is still not complete at that point: the idea here is to make sure hosting works ahead of time!

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 github pages.
- 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. Begin by testing the API out first in the browser. Simply copy and paste the URLs into the browser address bar and add the relevant query string. 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. Alternately, install a json viewer extension in Chrome.
2. Remember that JSON and JavaScript are case sensitive. For instance, it is easy to type in `stagedirection` and it won't work because the JSON field is actually `stageDirection`.
3. Your `index.html` file will include the markup for both **List Of Plays view** and the **Play Text view**. 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 design marks.
5. Most years, students tend to get low marks on the design side of the assignment. I would recommend looking at other sites on the internet and examine how they present data. Notice the use of contrast (weight, color, etc) and spacing.
6. 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?