COMP 3612
Assignment #2: Single-Page App

Version: 1.1 Nov 3, 2024
Due Monday November 18, 2024 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 Single-Page Application to view F1 (Formula 1) race data. You can work in pairs or by yourself. Don't delay finding a partner; if you cannot find a partner to work with, **don't ask me to be in a group of 3**, you will have to work alone. The assignment is very doable by one person, but it will be a more pleasant experience sharing the work with another.

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 (eventually) at:

```
  https://github.com/mru-comp3612-archive/f2024-assign2.git
```

Grading

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

Visual Design	15%
Programming Design and Documentation	10%
Functionality (follows requirements)	75%

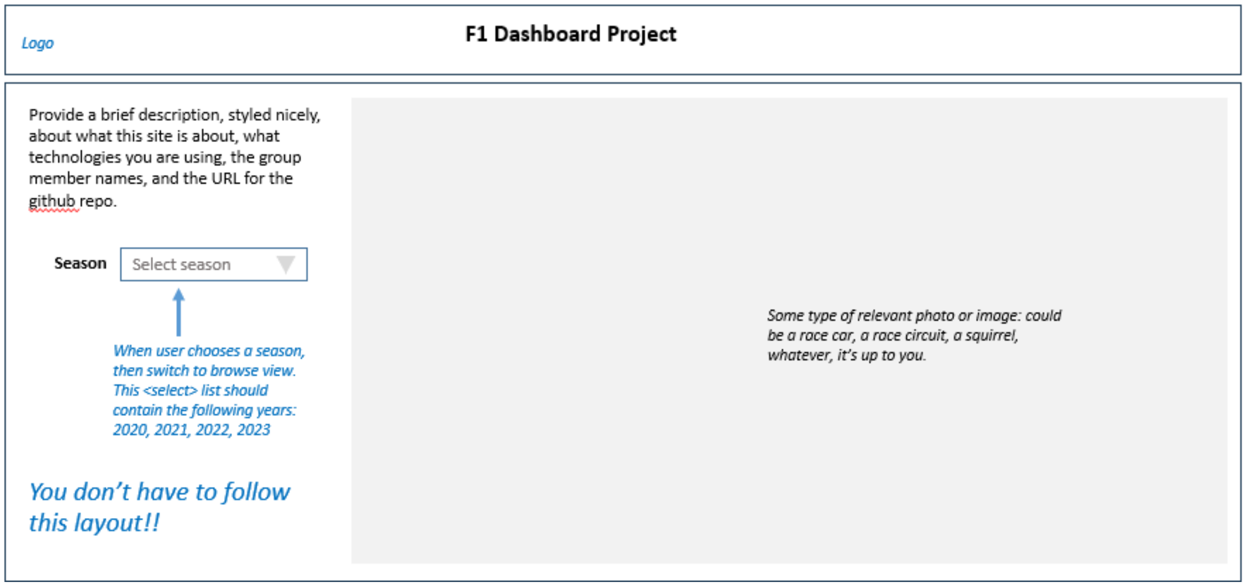
Requirements

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

1. You can use a third-party CSS library/framework (indeed I would encourage you to use one). Tailwind CSS is a great thing to have on your resume, but it does have a bit of a learning curve (Lab7b in your gumroad package can help you quickly learn its essentials). Be careful though that your CSS library is not using/requiring its own JavaScript library as well. Some popular CSS libraries (e.g., Bootstrap) include their own JavaScript libraries to do fancy things with select lists, modals, etc. **You are not allowed to include these** so be sure you don't have any extra <script> tags from your CSS library!
2. Your assignment should have just a single HTML page which **must be** named `index.html`.
3. The assignment should have two main views (**Home** and **Races**) and three dialog/popup views (**Favorites, Driver, Constructor**). When the program first starts, display the **Home** view.
4. The source of data is the provided F1 API. More detail will be provided below.

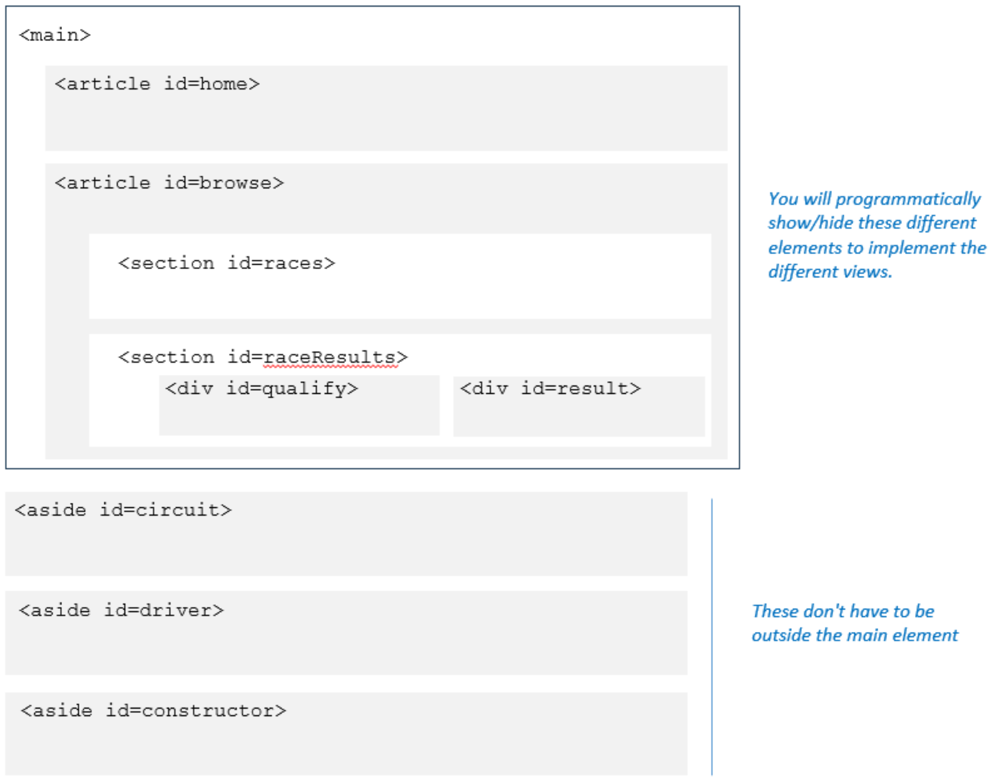
5. **Make sure you are only requesting/fetching the season race data ONCE after the user selects the season!** To improve the performance of your assignment (and reduce the number of requests on my server), you must store the race data in `localstorage` after you fetch it from the API (see Exercise 10.11 in Lab 10). Notice that you have to use `JSON.stringify` to create a JSON string version of the array and then saving that in `localstorage`; similarly, after you retrieve it from `localstorage`, you will need to uses `JSON.parse()` to turn it into an array. 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). Please do this task early on so that you reduce the load on my server! **Failure to implement this functionality will result in a very large loss of marks so don't neglect it.**
6. 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 antediluvian JavaScript found online will also result in lower marks. There is no need to credit code you found in the labs or lectures.
7. **Terminology**: In Formula 1, *races* happen on a *circuit* (e.g., the British Grand Prix is one the Silverstone circuit). There are 10 teams (called *constructors*) in F1; each constructor has two *drivers*. Before the race itself, there is *qualifying*, which determines the starting position. There are three rounds of qualifying, with 5 drivers knocked out in each of the first two rounds. A given race has a set number of laps; the first driver to cross the finish line after completing the set number of laps is the winner. In the race itself, there are points awarded to the driver *and* to the constructor for finishing in the first 10 places. Finishing 1st, 2nd, or 3rd gives the most points. There are other points that can be awarded but you don't care about that. At the season's end, there are two trophies: one for the driver with the most points and one for the constructor with the most points. In this assignment, you are only concerned with viewing races, displaying race and qualifying results, displaying information about circuits, drivers, and constructors, and specifying one's favorite circuits, drivers, and constructors.

8. **Home View**. You will implement a page with similar functionality as that described below. The provided sketch below illustrates the functionality. You are welcome to modify the design as you see fit. When the user selects a season, then switch to **Races View**. The title for this view doesn't have to be "F1 Dashboard Project"; it, and the other bold labels in these views, are there to explain what functionality must be on the view, not necessarily the actual labels that must appear on your page.



Logo
F1 Dashboard Project
Provide a brief description, styled nicely, about what this site is about, what technologies you are using, the group member names, and the URL for the github repo.
Season
Select season
When user chooses a season, then switch to browse view. This <select> list should contain the following years: 2020, 2021, 2022, 2023
You don't have to follow this layout!!
Some type of relevant photo or image: could be a race car, a race circuit, a squirrel, whatever, it's up to you.

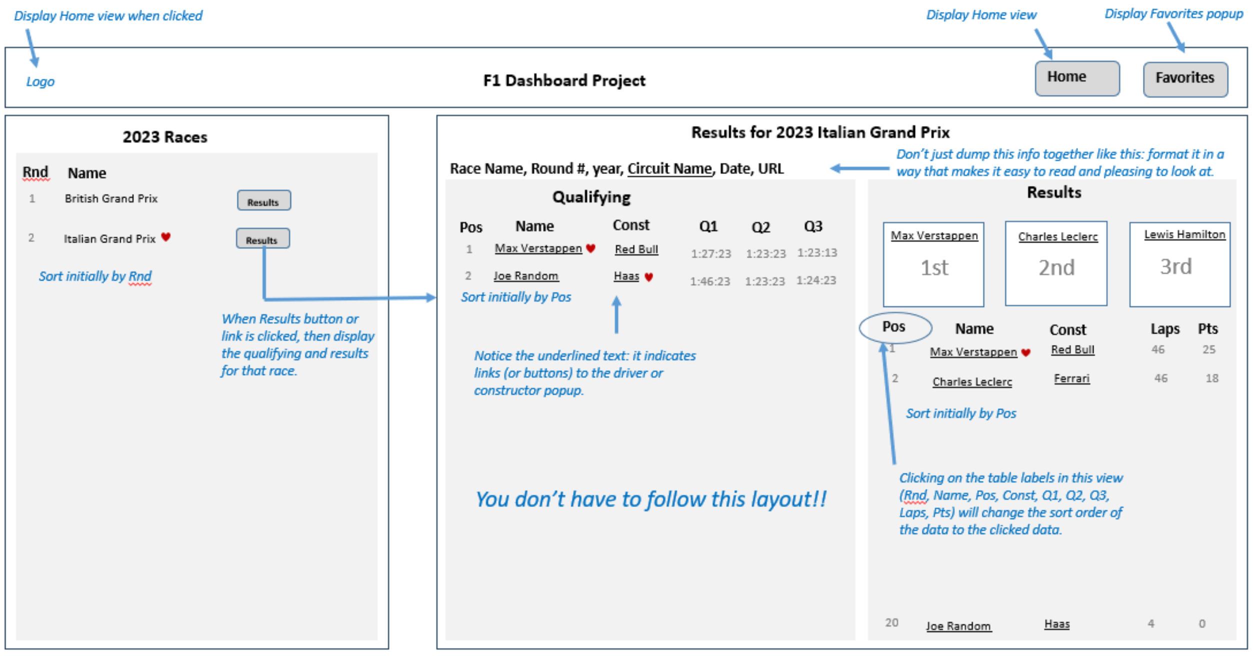
This assignment requires swapping different views. You should implement each view in its own container, e.g.,



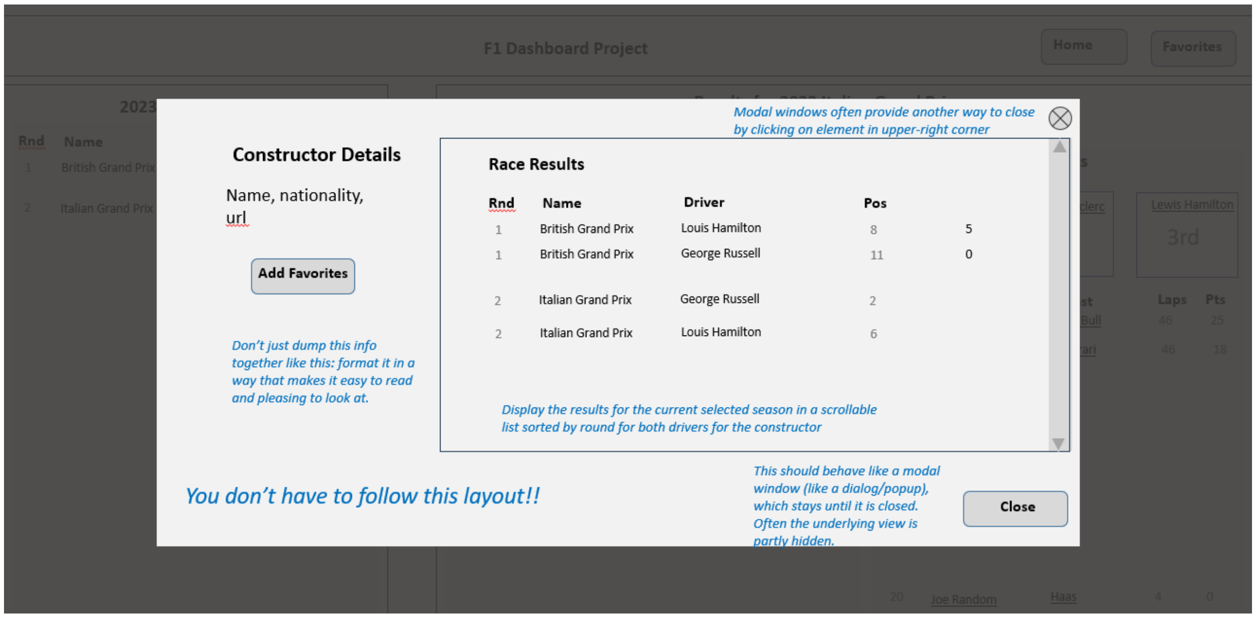
9. **Races/Browse View**. You will display the races (a *race* is on a *circuit*) for the selected season/year sorted by the round. Initially, this view will just display the races (perhaps display a message saying please select a race). When a race is selected, then display the qualifying results and race results. In the race itself, finishing 1st, 2nd, or 3rd gives the most points, so you should display those positions more prominently.

   People who follow F1 typically have their favorite drivers, constructors, and Circuits. In the **Driver, Constructor, and Circuit Popups** (see below), the user can add a driver, constructor, or circuit to their favorites list. In the Races View please indicate somehow if one of these is favorited. In the diagram shown here, I've used a heart icon but you can do it however you'd like.

   Initially display the races sorted by round, and the qualifying and results sorted by position. If the user clicks on the column headings, sort the list on the clicked field. The list should refresh whenever the user changes the sort option. Provide some visual cue that a sort change has occurred, such as an icon or styling changing in the header. The list must be an unordered list; ideally, make the list scrollable by setting the `overflow-y` property of the container to `scroll`.


Display Home view when clicked
Display Home view
Display Favorites popup
Logo
F1 Dashboard Project
Home
Favorites
2023 Races
Rnd
Name
1
British Grand Prix
Results
2
Italian Grand Prix ♥
Results
Sort initially by Rnd
When Results button or link is clicked, then display the qualifying and results for that race.
Results for 2023 Italian Grand Prix
Race Name, Round #, year, Circuit Name, Date, URL
Don't just dump this info together like this: format it in a way that makes it easy to read and pleasing to look at.
Qualifying
Pos
Name
Const
Q1
Q2
Q3
1
Max Verstappen ♥
Red Bull
1:27:23
1:23:23
1:23:13
2
Joe Random
Haas ♥
1:46:23
1:23:23
1:24:23
Sort initially by Pos
Notice the underlined text: it indicates links (or buttons) to the driver or constructor popup.
You don't have to follow this layout!!
Results
Max Verstappen
1st
Charles Leclerc
2nd
Lewis Hamilton
3rd
Pos
Name
Const
Laps
Pts
1
Max Verstappen ♥
Red Bull
46
25
2
Charles Leclerc
Ferrari
46
18
Sort initially by Pos
Clicking on the table labels in this view (Rnd, Name, Pos, Const, Q1, Q2, Q3, Laps, Pts) will change the sort order of the data to the clicked data.
20
Joe Random
Haas
4
0

10. **Constructor Popup.** Display details on the selected constructor in a modal-style popup. This typically involves displaying some type of element (div/aside/section/etc) that was previously hidden. It will stay displayed until the user close it (that is, it will become hidden again). You don't have to worry about letting the user change the sort order of race data in these popups: simply sort on Rnd (round). The Add Favorites button/link/icon will add the current constructor to the favorites list. Be sure to provide some *temporary* visual notification that an item has been added to the favorites, such as a toaster or even a temporary label. Don't use alert() for this!!



Modal windows often provide another way to close by clicking on element in upper-right corner

Rnd	Name	Driver	Pos	
1	British Grand Prix	Louis Hamilton	8	5
1	British Grand Prix	George Russell	11	0
2	Italian Grand Prix	George Russell	2	
2	Italian Grand Prix	Louis Hamilton	6	

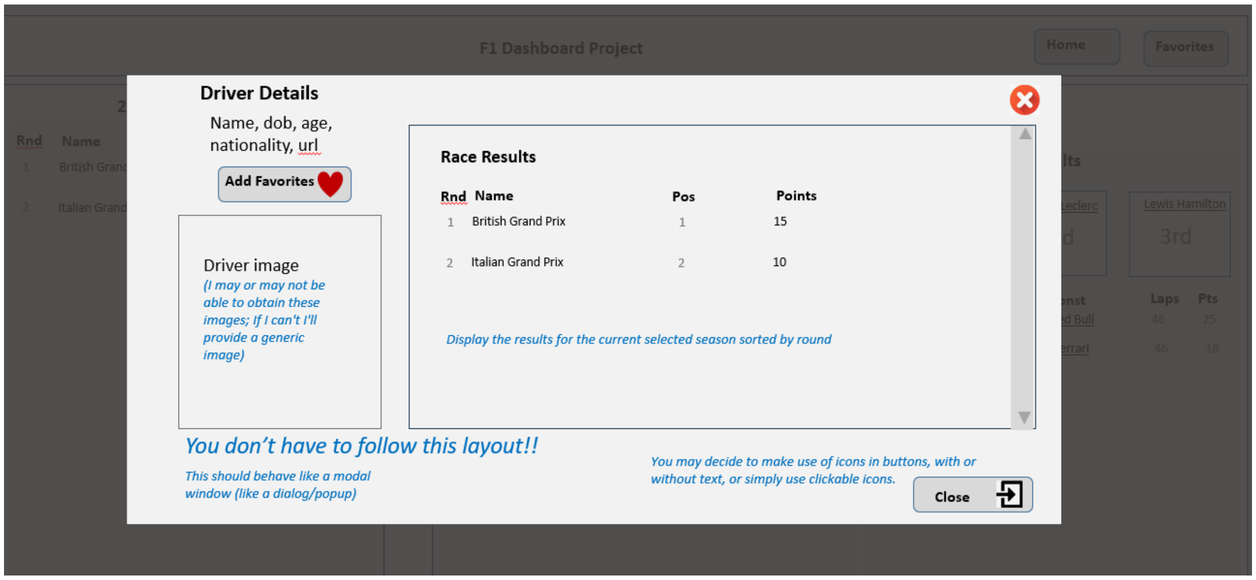
Don't just dump this info together like this: format it in a way that makes it easy to read and pleasing to look at.

Display the results for the current selected season in a scrollable list sorted by round for both drivers for the constructor

You don't have to follow this layout!!

This should behave like a modal window (like a dialog/popup), which stays until it is closed. Often the underlying view is partly hidden.

11. **Driver Popup.** Display details for a driver in a modal-style popup. It works in the same way as the **Constructor Popup**. Here I've included icons to show you that that is a possibility.



(I may or may not be able to obtain these images; If I can't I'll provide a generic image)

Rnd	Name	Pos	Points
1	British Grand Prix	1	15
2	Italian Grand Prix	2	10

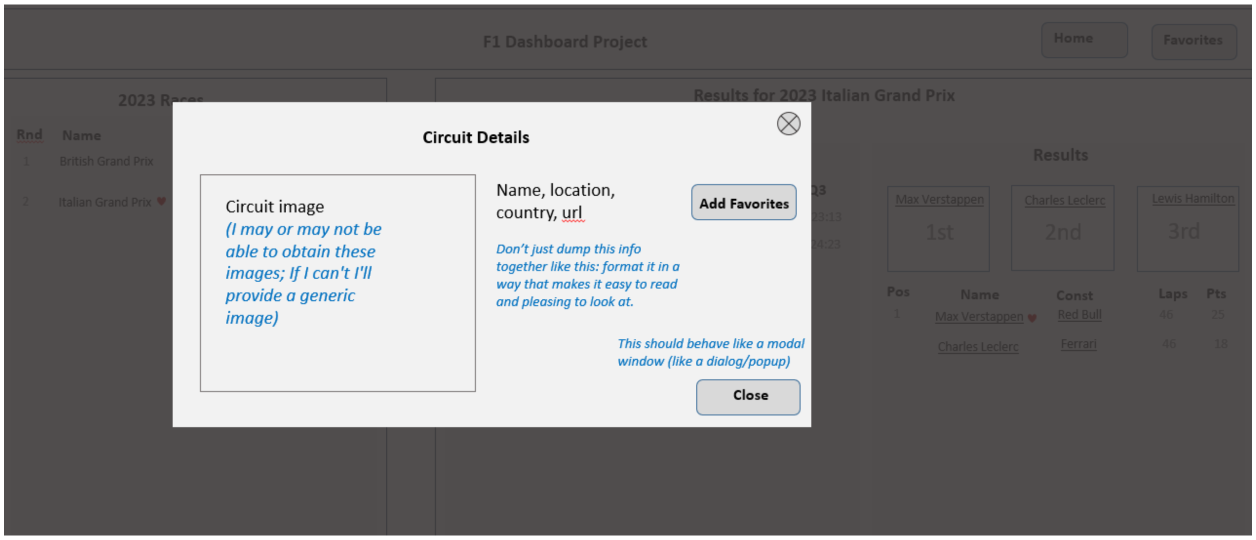
Display the results for the current selected season sorted by round

You don't have to follow this layout!!

This should behave like a modal window (like a dialog/popup)

You may decide to make use of icons in buttons, with or without text, or simply use clickable icons.

12. Circuit Popup. Display details for a circuit in a modal-style popup. It works in the same way as the **Constructor Popup**.

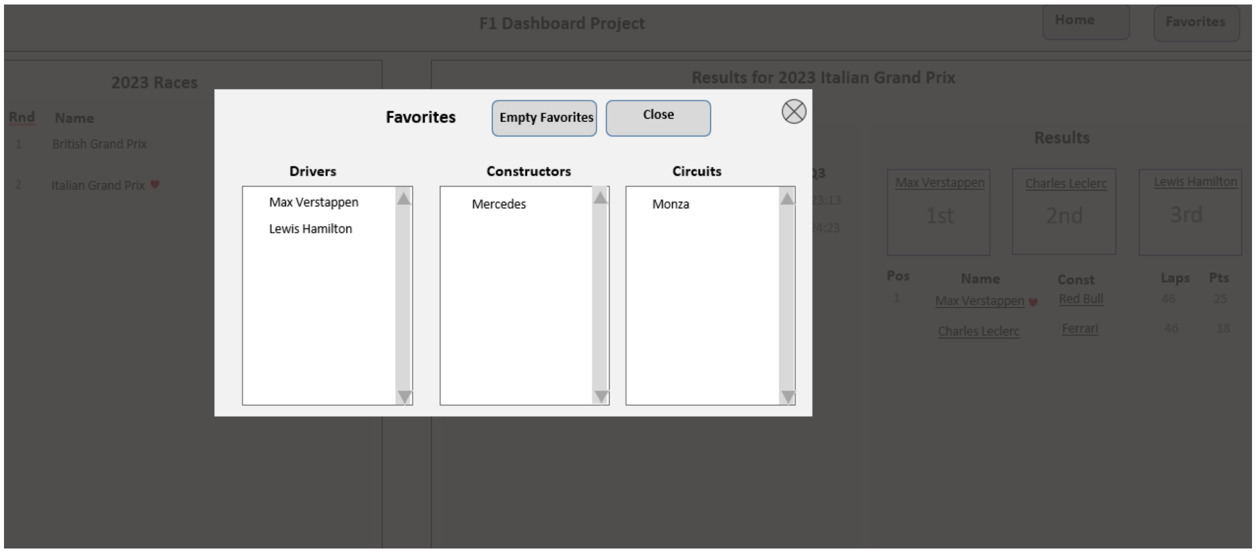


(I may or may not be able to obtain these images; If I can't I'll provide a generic image)

Don't just dump this info together like this: format it in a way that makes it easy to read and pleasing to look at.

This should behave like a modal window (like a dialog/popup)

13. Favorites Popup. Display the favorites in a modal-style popup. It works in the same way as the **Constructor Popup**. It should also provide a way to empty all the favorites. The favorites should be preserved even when the user exits the website. This will require saving the favorites in localStorage every time an item is added to favorites (or is emptied). This also means they should be loaded from localStorage when the page loads.



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.

API

I will be providing the API calls on my own server. **I will revise this document and provide details for the URLs and functionality of this API when they are available.**

Until then, I have provided csv files and a database diagram that illustrates how the different data tables are related.

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. Early on, try consuming the API in the browser. Simply copy and paste the URL 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. 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 `Name` and it won't work because the JSON field is actually `name`.
3. Your `index.html` file will include the markup for all your views. Your JavaScript code will programmatically hide/unhide (i.e., change the `display` value) the relevant markup container for these views.
4. Most of your visual design mark will be determined by how much effort you took to make the 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 a lot of code duplication (you shouldn't … if you are copy+pasting code, that should tell you that you are doing things wrong from a design standpoint)? 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, var, `XmlHttpRequest`, etc)? Is the code excessively reliant on found code from Stack Overflow, etc?