Homework #2

Data Collection and JavaScript Explorations

Due: Wednesday 2/5/2020 at 6pm

In this assignment you will explore using JavaScript to collect both static and dynamic user data in a web browser. Your use of JavaScript will be fairly basic but will include exploration of functions, event handling, timers, DOM manipulation and various browser APIs. Subsequent assignments will have you take a portion of this assignment and its collected data for reporting and visualization.

Write a script file called reporter.js that you will include on the various pages of the sample site you made in Homework #1. You will also add to this site another page reportertest.html that will allow you test to make sure that your data collector is working well and to explore what has been collected. Details of what the reporertest.html will contain comes later in the description.

Note: While you will use the pages from the previous assignment you must make a copy of the HW1 site and put any new work in this new copy to insure that grading of previous work is not tainted. Again do not start adding content to your HW1 directly where it is currently hosted, duplicate it and put it at another URL where you will add this new work.

reporter.js

This vanilla JavaScript (vanilla = no frameworks) file will collect both static and dynamic user data. You should be able to include this script in any page on your static site with something like `<script src="reporter.js"></script>` and it will do its collecting.

Static collected data includes:

- user-agent string (string)
- user language (string)
- cookies being on (boolean)
- javascript being on (boolean)
- images being on (boolean)
- CSS being on (boolean)

- available screen dimensions (max ability of the monitor - numbers: height and width)
- window size (height and width upon load - numbers: height and width)
- effective connection type (string)

Tip: Do not over think the boolean examples or immediately ask for help. There may not be a property from some of them but some simple logic or way of using the DOM could easily allow you to determine such values.

These static values should only be collected once unless they changed.

Every page load you should also collect as much performance data as you can including

- page loading information - use the `PerformanceTiming` object to find this information. We will not enumerate all the values it will be your job to understand what is being provided.
- initial start loading times
- end loading time
- total time taken

Once the page is loaded you then listen for and record dynamic data in the form of user events including

- clicks
- mouse movements
- keystrokes
- scrolling (`window`)
- page unloading (`beforeunload`)

- idle time

  Details: You should record idle time greater than 2 seconds - what this means is that if a user stops doing one of the events above for more than 2 seconds you record the amount of idle resting time before the next event starts. If you want to think of this as an event call it "Idle". Thinking of it as an event could be helpful for reasoning.

  Given that there is a 0.Xms - 1999ms zone of uncertainty if it is useful you may think of that time as the event "Pause". A pause time is not big enough to suggest that the user is not attentive. Tracking pauses may help you record things or reason about the event flow more easily.

All the information that you collect should be stored in the `localStorage` of your browser. You should consider using an array of objects and will be forced to explore the use of JSON (hint). The uniqueness of an entry should be based upon the URL recorded and the Date Time. In subsequent assignments we will discuss sessionization of the data and transmission to an end point. Given that those are requirements of the next assignment you may start thinking about code organization now.

Reportertest.html

Your reporter script should be put on all the various pages in the test site, but a special page called reportertest.html should allow you to explore what you have collected. Like the previous example this page should use valid HTML and CSS. You are allowed to use CSS frameworks as you like to beautfiy the page, but note that it may add complexity you don't want. You may not however, use JS frameworks though as we want vanilla JavaScript only here.

Note: Students hoping to play with JS frameworks will have some ability to do that on their backend reporting pages (HW3 and beyond) if they so desire.

The reporting page must have a list of all the data capture entries. You may choose to do some form of pagination or scrolling to address the large amount of entries you may record. Each entry should have a link that if you click on it that it brings up the information recorded on the page. The information should be presented as follows:

- static information - put it in an HTML table
- performance information - put it in an HTML table but make sure to summarize the most relevant information (start, stop, load) at the top followed by the more specific data
- event information - present the events in a scrolling region (<section> or with scrolling) as a list. Clearly label each type of event and use some icon or color code to indicate the type of event. Each event should be a link that if you click it will reveal the full event object in a dialog or elsewhere on the page.

Because entries may get messed up add a special button "Purge" that will remove the entry you are looking at from `localStorage` as well a button "Purge All" that will remove all entries from storage. Both buttons when pressed should require you to agree to the action so as to not accidentally remove data.

Note: We will not draw this page out nor tell you how this page looks exactly. As this is a diagnostic page the exact look is not a massive consideration as thus is left up somewhat to your discretion. However, note that if the graders are confused and can't explore it properly points will obviously lost. Further note that while aesthetics are not a major aspect of the diagnostic page they are encouraged. Very messy interfaces will loose points so at least make what you do orderly. For optimal or even extra point consideration try to make it look professional or even pleasing.

Tips

- Your event handlers and overall collection code should assume that other code is in the page. In other words do not bash variables and avoid polluting the global window space.
- The idle mechanism is likely the most challenging part of the assignment. There are many ways to do this, but timers are likely involved and some simple logic as well since you will have time stamps on events as well. This can devolve into a convoluted mess without planning. Also do not assume you can do this easily by calculating everything at the end as the page unloads as that is an inappropriate solution that will fail outside base cases.
- Do the static properties first, then the performance and then the events. Do the basic click event and then keyboard before tackling the most voluminous event information.

- Build your reporter page as you go along to see what you are doing and to get your handy purge mechanism ready as soon as you can.

- Use your dev tools in the browser to explore properties and storage before you code things.
- The amount of code won't be large but writing out the reporting page might be a bit messy with the DOM. Make sure you consider using template strings (``) to make data output easy. The tables in particular can be hard or easy depending on how you do this.
- Understand that if you reach for a 3rd party resource for tables or CSS it may actually cost you more time as you have to learn it and it may have bugs or challenges of its own.
- Sketch out the structure of your reporting script and especially reporting page before you code it. A little planning will saves you potentially hours or rework and coding.
- Give yourself enough time to do the assignment. The work ramp up here is significant.

Advanced Variations

Students looking for more challenge given their previous experience in my classes (112, 134) or outside web dev background can consider the following modifications for additional points. Additions should happen in the order shown from top to bottom.

- Add in error tracking - collect JavaScript errors that happen in a page and save those to your collected data
- Data size reduction - the natural inclination will be to collect the raw data of events and objects in their natural form. Given the amount of data collect storage and transmission of the sniffed data could become a problem. Explore a minimal form with a modified data format, compression or both to improve the algorithm. Try not to throw away needed data though.
- Replay mechanism for the events - provide a mechanism using an iframe to load a URL and replay the events that are collected. Note: You can dispatch events but the movement may be difficult to simulate unless you think of moving a "fake cursor" image around that matches the mouse position collected.

- Find and adopt browser finger printing algorithm and see if you can correct a unique signature for a browser for client-side sessionization

Logistics Note

A Gradescope entry with logistics will be provided with any details on submission. Again remember you should fork your project from HW1 to avoid grading problems.