Project Overview

In this project, you'll use a common web development technique known as “pagination” to enhance the usability of a web page.

You'll start with a file that contains information for 42 students, including name, email, and picture. You'll start by using your JavaScript skills to display the first nine students on the page. Then you will add navigation, also known as pagination buttons, that the user can click to display different pages of students.

Your application will need to work with a list of any number of students, not just with the supplied array of 42 students. This means your solutions need to be flexible enough to handle arrays of student data of various lengths.

For this project, you will want to write your code using plain JavaScript, sometimes known as “vanilla” JavaScript. Avoid using jQuery or any other libraries, plugins, or code snippets for any aspect of this project, including the pagination.

Before You Start

Before starting your project download the project started template from the given link.

- The `index.html` file contains the initial HTML markup. Don't change anything in this file!
- The `css` folder contains the styles for the project. You won't need to change anything in these files.
- The `js` folder contains a `script.js` and a `data.js` file. The `script.js` file is where you will write your code for this project. The `data.js` file contains the data you will use for this project and includes a `data` variable, which you can use in the `script.js` file to access the student data.
- The `img` folder contains a search icon that will be used to search the student directory of this project.
- The `example_output.png` file shows what the completed versions of the project should look like in the browser.

Project Instructions

For this project, you have to do these things -

1. **Display content** :
    - Create a function that will show a “page” of nine students.
    - This function should have two parameters:
        - A `list` parameter to represent student data that will be passed as an argument when the function is called.
        - A `page` parameter to represent the page number that will be passed as an argument when the function is called.
    - Inside the function:
        - Create two variables to store the start index and the end index of the list items to be displayed on the given page. To make this function dynamic, use the `page` parameter and some basic math to calculate the value of these variables like so:
            - Start Index = (`page` parameter * items per page) - items per page
            - End Index = `page` parameter * items per page
        - Select the `UL` element with a class of `student-list` and assign its value to a variable.
        - Use the innerHTML property to set the HTML content of the `student-list` variable you just created to an empty string. This will remove any students that might have previously been displayed.
        - Loop over the `list` parameter.
        - Inside the loop:
            - Write a conditional statement that checks if the current index (`i`) is greater than or equal to the `start index` variable and less than the `end index` variable.

        - Inside that conditional:
            - Create the DOM elements needed to display the information for each matching student as you iterate over the `list` parameter.
            - Insert the elements you have created to the `student-list` variable you created earlier. The insertAdjacentHTML method and `beforeend` option work well for this.

2. **Add Pagination Buttons**
    - Create a function that creates and appends functioning pagination buttons.
    - This function should accept a single `list` parameter to represent student data that will be passed as an argument when the function is called.
    - Inside the function:
        - Create a variable to store the value of the number of pagination buttons needed. You can calculate this using the length of the `list` parameter. Remember, you will want to display nine students per page.
        - Select the `UL` element with a class of `link-list` and assign its value to a variable.
        - Use the innerHTML property to set the HTML content of the `link-list` variable you just created to an empty string. This will remove any pagination buttons that might have previously been displayed.
        - Loop over the variable for the number of pages needed that you created earlier.
        - Inside the loop:
            - Create the DOM elements needed to display the pagination button as you iterate over the number of pages.

        - Insert the elements you have created to the `link-list` variable you created earlier. The insertAdjacentHTML method and `beforeend` option work well for this.
    - Select the first pagination button and give it a class name of `active`.
    - Create an event listener to listen for clicks on the `link-list` variable that you created earlier.
    - Inside this event listener:
        - The click event should only fire when the buttons are clicked. Click event should not fire if a user clicks between or around buttons. So if the click target is a pagination button:
            - Remove the `active` class from any other pagination button.
            - Add the `active` class to the pagination button that was just clicked.
            - Call the `showPage` function passing the `list` parameter and the page number to display as arguments.

3. **Add a Search Component**
    - Dynamically create and add a search bar. Avoid making any changes in the `index.html` file and instead, use your JavaScript skills for this. Below is an example of the format of the search bar elements. The search bar can't be unstyled. If you follow the example below, the provided CSS will style the search bar for you.
4. **Add Functionality to the Search Component**
    - When the "Search" is performed, the student data is filtered so that only students whose name includes the search value are shown. The search should be case insensitive and work for partial matches. For example, if the value `B` or `b` is typed into the search field, students with “Bill” in the name would be shown. Likewise, if `LL` were typed into the search field, students

with the first name "Bill" would appear, as well as students with the last name "Williams".

5. **Add Pagination for Search Results**
    - The pagination buttons should change based on the number of matches to the search. For example: if nine or fewer matches are found, there should be a 0 or 1 pagination button. If 22 matches are found, there should be 3 pagination buttons.
    - Clicking on a pagination button should display the corresponding matching students for that page.
6. **Handle No Search Matches**
    - If no matches are found for a search, display a “No results found” type message on the page.

Before Submitting the Project

Before you submit your project, make sure -

1. Your GitHub repo for this project contains only this project, only files needed to make this project run, and a `README.md` file providing details about your project.
2. You wrote all of my own code for this project. Any code included in your project that you did not write myself is appropriately attributed to its source.
3. You have completed all of the project requirements and believe the project is ready to receive an Expectations grade.

4. You understand that what you submit is what will get reviewed and that if you submit the project after the deadline, your submission won't be seen.

After completing the project, test if everything works perfectly, create a GitHub repository, and upload this project there. Then you have to make the project live using Github Pages. Fill up the submission form before the deadline pass. This project will help you to hone some powerful JavaScript skills and a strong portfolio piece to show off to potential employers.

Happy Coding.

Tahsin Shahriar

Digital Signer:Tahsin Shahriar
DN:C=BD, E=tahsin.ewucse@gmail.com, O=Shikhbe Shobai, CN=Tahsin Shahriar
Date:2021.11.09 16:02:46 +06:00