Instruction

Walkthrough for Project 2: Data Pagination and Filtering

Project 2 - Data Pagination and Filtering

This Walkthrough is a step by step guide to the different sections in the Instruction tab for this project. It is meant to be a supplemental resource. For best results, you should try to build out the project on your own *before* consulting the Walkthrough. Work through each step in the Instruction tab on your own. If you get stuck, check out the resources in the Review Your Resources section of the Instructions like the [Study Guide](https://teamtreehouse.com/library/study-guide-for-project-2-data-pagination-and-filtering), Project Warm-Ups, and Google. Then if you are still unable to make progress, reference the Walkthrough for additional details.

Before You Start

Building a project can be a challenging task but it is made easier by proper planning and organization. Before you write your first line of code, be sure you have completed all of the steps on the checklist below.

 Complete all Unit 2 curriculum.

 Create a GitHub repository for your project.

 Download the starter files for the project and push them to your GitHub repo. Review these files to familiarize yourself with them.

 Review the steps in the Instruction tab for the project. Consider what each step does and how the different parts of the application work together.

Instruction Walkthrough

1. Review Your Resources

You will find a list of resources on the right side of the project page, just below the “Download files” section. Be sure to review these resources and become familiar with them before beginning the project.

2. Getting Started with the Project Files

If you haven't already, open up the project in your text editor. Now preview the project in Chrome and open up the Dev Tools console. If you don't remember how to open the project in a browser, check out this lesson for a refresher: [Preview Files in a Browser](https://teamtreehouse.com/library/preview-files-in-a-browser).

Now return to your text editor and check out the data.json file. This file contains a data variable, which is an array of objects. Each of these objects contains information on a student. We will be using this data to display students and create pagination buttons.

Now access the script.js file in your text editor. Let's start by using the console.log() method to log out the data variable. This snippet of code should look like console.log(data);. Now return to your browser, refresh the page, and take a look at what logs out to the Dev Tools console. You should see an array containing 42 objects.

Now that you know how to access the student information using the data variable, let's start building our application.

For best results, we are going to take it one small step at a time, testing our code as we go.

3. Display a “page”

We are going to start by defining a function that will display a page of nine students. We will name this function showPage. This function will take two parameters: list, and page. The list parameter will represent the array of student data we are working with and page parameter will be the page number that we want to display.

Below is what your function should look like, including placeholder comments for the functionality that we will be adding later in this step.

function showPage(list, page) {

*// create two variables which will represent the index for the first and last student on the page*

*// select the element with a class of `student-list` and assign it to a variable*

*// set the innerHTML property of the variable you just created to an empty string*

*// loop over the length of the `list` parameter*

*// inside the loop create a conditional to display the proper students*

*// inside the conditional:*

*// create the elements needed to display the student information*

*// insert the above elements*

}

Pro Tip: It is a good idea to regularly test your code as you are writing it. Using console.log() statements is a great way to do this. Before we go any farther, lets test our `showPage` function. First, add console.log(list); and console.log(page); inside of the function. Then call the function by adding showPage(data, 1); just below where you created the function. Now if you open the Dev Tools console you should see an array with 42 objects and the number “1” logged out. Once you confirm that your function is working you can remove the two console.log statements but leave the call of the showPage function as that will come in handy later.

Now create two variables, startIndex and endIndex, which will calculate the index for the first and last student to display on the page. Use the page parameter and some basic math to calculate the value of these variables like so:

* startIndex = (page parameter \* items per page) - items per page
* endIndex = page parameter \* items per page

Pro Tip: Remember, JavaScript is zero-indexed so the index of the first student object in `data` array is 0. Before moving on, it would be a good idea to use console.log() statements to make sure these new variables are working as expected.

Next use the [querySelector method](https://developer.mozilla.org/en-US/docs/Web/API/Document/querySelector" \t "_blank) to select the UL element with a class of student-list and assign it to a new variable named studentList. This is the element we will be adding our student data to.

Now set the [innerHTML property](https://developer.mozilla.org/en-US/docs/Web/API/Element/innerHTML" \t "_blank) of the studentList variable to an empty string. This will remove any existing students that might have been displayed previously.

Next we will create a for loop that will run once for each object in the list parameter. We can do this by using the length property of list.

Inside the loop, create a conditional statement that checks if i is greater than or equal to the startIndex variable and less than the endIndex variable. The student at these indexes are the ones we want to display on the page.

If that condition is met, we will create the DOM elements needed to display the student at that index, which we will assign to a variable named studentItem. We will create these elements using a template literal. Use bracket notation to access the student object at that index and dot notation to access the specific properties of that student object. The end result should be elements that look like this:

<li class="student-item cf">

<div class="student-details">

<img class="avatar" src="https://randomuser.me/api/portraits/women/25.jpg" alt="Profile Picture">

<h3>Ethel Dean</h3>

<span class="email">ethel.dean@example.com</span>

</div>

<div class="joined-details">

<span class="date">Joined 12-15-2005</span>

</div>

</li>

Once the template literal is created, we want to insert it into the DOM on the studentList variable using the [insertAdjacentHTML method](https://developer.mozilla.org/en-US/docs/Web/API/Element/insertAdjacentHTML" \t "_blank) and beforeend position.

Now that we have finished the showPage function we want to test it to make sure everything is working as expected before we move on. To do this, call the showPage function and pass data and 1 as arguments, which you might already be doing if you tested your code previously. Refresh the application in the browser and you should see the first nine students displayed on the page.

4. Add Pagination Buttons

Next, we are going to define a function named addPagination that will create and display our pagination buttons. This function will take one parameter, list, which will represent the array of student data we are working with.

Below is what your function should look like, including placeholder comments for the functionality that we will be adding later in this step.

function addPagination(list) {

*// create a variable to calculate the number of pages needed*

*// select the element with a class of `link-list` and assign it to a variable*

*// set the innerHTML property of the variable you just created to an empty string*

*// loop over the number of pages needed*

*// create the elements needed to display the pagination button*

*// insert the above elements*

*// give the first pagination button a class of "active"*

*// create an event listener on the `link-list` element*

*// if the click target is a button:*

*// remove the "active" class from the previous button*

*// add the active class to the clicked button*

*// call the showPage function passing the `list` parameter and page to display as arguments*

}

Pro Tip: Just like we did earlier with the showPage function, it would be a good idea to test our addPagination function before we go any further. To do that, add console.log(list); inside of the function. Then call the function by adding addPagination(data); just below where you declared the function. Then open the Dev Tools console and you should see an array with 42 objects logged out there. Be sure to regularly test your code as you work through the rest of this walkthrough!

Next create a variable named numOfPages, which will calculate the number of pagination buttons we will need. Use the list parameter, the [Math.ceil() function](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Math/ceil" \t "_blank), and some basic math to calculate this number and round it up like so:

* numOfPages = Math.ceil(list length / items per page)

Then use the querySelector method to select the UL element with a class of link-list and assign it to a new variable named linkList. This is the element we will be adding our pagination buttons to.

Next set the innerHTML property of the linkList variable to an empty string. This will remove any existing pagination buttons that might have been displayed previously.

Now create a for loop using the numOfPages variable we created earlier. Since our pagination buttons should start at 1, set the loop up so that i equals 1 initially. The loop should run until i is less than or equal to numOfPages.

Inside the loop, we will create the DOM elements needed to display the pagination button, which we will assign to a variable named button. We will create this element using a template literal. The end result should look like this:

<li>

<button type="button">1</button>

</li>

Once the template literal is created, we will insert it into the DOM on the linkList variable using the insertAdjacentHTML method and beforeend position.

Next, we'll add the active class to the first button. We can do this by using querySelector to select the first button element and then use the [className property](https://developer.mozilla.org/en-US/docs/Web/API/Element/className" \t "_blank) to set the class to "active".

Pro Tip: Hopefully you have been continuing to test your code as you go but either way, this would be a great point to test your addPagination function again. Be sure you are calling the addPagination and passing data as an argument, which you might already be doing if you have tested your code previously. If everything is working as it should, you should be able to refresh the page and see five pagination buttons on the screen. These buttons won't do anything yet but we will work on that part next!

Now create an [event listener](https://developer.mozilla.org/en-US/docs/Web/API/EventTarget/addEventListener) on linkList that will be called when there is a click event.

Inside this event listener, write a conditional that checks if the tagName of the event target (i.e. the element clicked) is a BUTTON element.

If that condition is met, use querySelector to select the first element with a class of active and then set the className property to an empty string. This will remove the active class from the previous button. We can use querySelector here rather than looping over all of the buttons because there should only ever be one button with the active class.

Still inside of the conditional, lets next add the active class to the button that was clicked (i.e. the click target). Again, we can do this using the className property.

Last but not least, still in the conditional, we will call the showPage function passing the list parameter and the text content of the click target as arguments. We can access the text content of the click target by using the [textContent property](https://developer.mozilla.org/en-US/docs/Web/API/Node/textContent" \t "_blank).

At this point, your addPagination function should be finished. Let's test it to make sure everything works as expected. Be sure that you are calling the addPagination function and passing data as an argument. If you refresh the page, you should see five pagination buttons on the screen. Clicking a button should display the nine students for that “page”.

5. Call Functions

Since you have been testing your code as you go, you have probably already called your showPage and addPagination functions but if you haven't be sure to do so now.

When you call the showPage function you should pass data and 1 as arguments. When you call the addPagination function you should pass data as an argument.

6. Finishing the Project

We are nearing the finish line now! The next step is to review our project and make sure it is in keeping with best practices and that it works properly.

Start by ensuring that your code is properly commented. Good code comments briefly describe your code and what it does.

Next check the formatting of your code make sure that it is in keeping with best practices. This will improve the readability of your code.

From there open your application and preview it in the browser. Test out the application like a user would, looking for bugs or other unexpected behavior. Be sure to keep the Dev Tools console open while you do this!

Last but not least, consider testing out your application in multiple browsers. To pass, your app only needs to work in Chrome but it is common for developers to test their projects in multiple browsers.

7. Before Submitting the Project

Before you submit your project, closely review the “[How you’ll be graded](https://teamtreehouse.com/projects/list-pagination-and-filtering#how-you-ll-be-graded)” section for this project and double-check that you have met all of those requirements.

It is also a good idea to post your project in Slack for others to review before you officially submit it.

[Ok, I got it](https://teamtreehouse.com/instructions/4967/complete)