Final Project Discussion

In this document, I want to provide some discussion for each of the 21 functions you will create for this project. The functions are individually unit tested just as your weekly practice exercises have been.

Before discussing the functions, here is some **mandatory information:**

1. Do not modify the provided HTML, CSS or other files. The code you write should only go in the main.js file.

2. I wanted to have you use modules with imports, but automated testing gets much more complicated this way… so all of your code goes in one file for this project.

3. 21 functions may sound like a lot but… when you test code, you want to limit how much one function does. This is a good habit overall. Keep each function as simple as possible. Ideally, each function would only do one thing, but working with the DOM is not always that simple.

4. I decided to automate much of this project like your practice exercises for a few reasons:
    a. You get immediate feedback.
    b. You are already used to this format.
    c. The number of students enrolled. Last year, it took me 2 weeks to grade nearly 50 final projects. This year there are more like 70. I will still have some manual tasks, but this will be easier to manage.
    d. I have received overall positive feedback with this format.
    e. This will allow me to provide a much later due date because I will not have weeks of grading to do - bonus for you!

5. I have spent a lot of time constructing and testing this, but this is still the first time I am using it. Please reach out with any questions / screenshots / etc.

6. I know this stifles the creative factor compared to just saying “build a project of your liking”.... But the benefits mentioned above won out in that regard. I do hope to add some CSS testing in future semesters.

7. All feedback is welcome! In fact, encouraged! I want to constantly improve this process.

8. **Give yourself plenty of time!** 21 functions over approximately a month really means you can write around 5 per week which is a similar pace to what your practice assignments have been.

9. Use the required function names. Complete the functions in the order provided.

10. Reference my project cheatsheet. I’ll likely be adding to it as we go, too.

Individual functions discussed on the next page.

Functions:

1. createElemWithText
   a. Receives up to 3 parameters
   b. 1st parameter is the HTML element string name to be created (h1, p, button, etc)
   c. Set a default value for the 1st parameter to "p"
   d. 2nd parameter is the textContent of the element to be created
   e. Default value of the 2nd parameter is ""
   f. 3rd parameter is a className if one is to be applied (optional)
   g. Use document.createElement() to create the requested HTML element
   h. Set the other desired element attributes.
   i. Return the created element.

2. createSelectOptions
   a. Test users JSON data available here: https://jsonplaceholder.typicode.com/users
   b. For testing (not in function) you may want to define users with the test data.
   c. Receives users JSON data as a parameter
   d. Returns undefined if no parameter received
   e. Loops through the users data
   f. Creates an option element for each user with document.createElement()
   g. Assigns the user.id to the option.value
   h. Assigns the user.name to the option.textContent
   i. Return an array of options elements

3. toggleCommentSection
   a. Receives a postId as the parameter
   b. Selects the section element with the data-post-id attribute equal to the postId received as a parameter
   c. Use code to verify the section exists before attempting to access the classList property
   d. At this point in your code, the section will not exist. You can create one to test if desired.
   e. Toggles the class 'hide' on the section element
   f. Return the section element

4. toggleCommentButton
   a. Receives a postId as the parameter
   b. Selects the button with the data-post-id attribute equal to the postId received as a parameter
   c. If the button textContent is 'Show Comments' switch textContent to 'Hide Comments'
   d. If the button textContent is 'Hide Comments' switch textContent to 'Show Comments'
   e. Suggestion (not required) for above: try a ternary statement
   f. Return the button element

5. deleteChildElements
    a. Receives a parentElement as a parameter
    b. Define a child variable as parentElement.lastElementChild
    c. While the child exists…(use a while loop)
    d. Use parentElement.removeChild to remove the child in the loop
    e. Reassign child to parentElement.lastElementChild in the loop
    f. Return the parentElement

NOTE: The above functions had no dependency on other functions. They were very self-contained which is ideal. That is not always possible though. We will try to limit dependencies as we go. The next several functions have small dependencies.

6. addButtonListeners
    a. Selects all buttons nested inside the main element
    b. If buttons exist:
    c. Loop through the NodeList of buttons
    d. Gets the postId from button.dataset.id
    e. Adds a click event listener to each button (reference addEventListener)
    f. The listener calls an anonymous function (see cheatsheet)
    g. Inside the anonymous function: the function toggleComments is called with the event and postId as parameters
    h. Return the button elements which were selected
    i. *You may want to define an empty toggleComments function for now.* Not all tests will pass for addButtonListeners until toggleComments exists. I recommend waiting on the logic inside the toggleComments function until we get there.

7. removeButtonListeners
    a. Selects all buttons nested inside the main element
    b. Loops through the NodeList of buttons
    c. Gets the postId from button.dataset.id
    d. Removes the click event listener from each button (reference removeEventListener)
    e. Refer to the addButtonListeners function as this should be nearly identical
    f. Return the button elements which were selected

8. createComments
    a. *Depends on the createElemWithText function we created*
    b. Receives JSON comments data as a parameter
    c. Creates a fragment element with document.createDocumentFragment()
    d. Loop through the comments
    e. For each comment do the following:
    f. Create an article element with document.createElement()
    g. Create an h3 element with createElemWithText('h3', comment.name)
    h. Create an paragraph element with createElemWithText('p', comment.body)
    i. Create an paragraph element with createElemWithText('p', `From: ${comment.email}`)
    j. Append the h3 and paragraphs to the article element (see cheatsheet)
    k. Append the article element to the fragment
    l. Return the fragment element

9. populateSelectMenu
    a. *Depends on the createSelectOptions function we created*
    b. Receives the users JSON data as a parameter
    c. Selects the #selectMenu element by id
    d. Passes the users JSON data to createSelectOptions()
    e. Receives an array of option elements from createSelectOptions
    f. Loops through the options elements and appends each option element to the select menu
    g. Return the selectMenu element

NOTE: The next functions use Async / Await to request data from an API. We cover this in Week 13. I do not recommend proceeding beyond this point until you have completed the learning module for Week 13.

10. getUsers
    a. Fetches users data from: https://jsonplaceholder.typicode.com/ (look at Resources section)
    b. Should be an async function
    c. Should utilize a try / catch block
    d. Uses the fetch API to request all users
    e. Await the users data response
    f. Return the JSON data

11. getUserPosts
    a. Receives a user id as a parameter
    b. Fetches post data for a specific user id from: https://jsonplaceholder.typicode.com/ (look at Routes section)
    c. Should be an async function
    d. Should utilize a try / catch block
    e. Uses the fetch API to request all users
    f. Await the users data response
    g. Return the JSON data

12. getUser
    a. Receives a user id as a parameter
    b. Fetches data for a specific user id from: https://jsonplaceholder.typicode.com/ (look at Routes section)
    c. Should be an async function
    d. Should utilize a try / catch block
    e. Uses the fetch API to request the user
    f. Await the user data response
    g. Return the JSON data

13. getPostComments
    a. Receives a post id as a parameter
    b. Fetches comments for a specific post id from: https://jsonplaceholder.typicode.com/ (look at Routes section)
    c. Should be an async function
    d. Should utilize a try / catch block
    e. Uses the fetch API to request all users
    f. Await the users data response
    g. Return the JSON data

NOTE: The next functions will depend on the async API data functions we just created. Therefore, these functions will also need to be async. When they call the API functions, they will need to await data from those functions.

14. displayComments
    a. Dependencies: getPostComments, createComments
    b. Is an async function
    c. Receives a postId as a parameter
    d. Creates a section element with document.createElement()
    e. Sets an attribute on the section element with section.dataset.postId
    f. Adds the classes 'comments' and 'hide' to the section element
    g. Creates a variable comments equal to the result of await getPostComments(postId);
    h. Creates a variable named fragment equal to createComments(comments)
    i. Append the fragment to the section
    j. Return the section element

15. createPosts
    a. Dependencies: createElemWithText, getUser, displayComments
    b. Is an async function
    c. Receives posts JSON data as a parameter
    d. Create a fragment element with document.createDocumentFragment()
    e. Loops through the posts data
    f. For each post do the following:
    g. Create an article element with document.createElement()
    h. Create an h2 element with the post title
    i. Create an p element with the post body
    j. Create another p element with text of `Post ID: ${post.id}`
    k. Define an author variable equal to the result of await getUser(post.userId)
    l. Create another p element with text of `Author: ${author.name} with ${author.company.name}`
    m. Create another p element with the author’s company catch phrase.
    n. Create a button with the text 'Show Comments'
    o. Set an attribute on the button with button.dataset.postId = post.id
    p. Append the h2, paragraphs, button, and section elements you have created to the article element.
    q. Create a variable named section equal to the result of await displayComments(post.id);
    r. Append the section element to the article element

    s. After the loop completes, append the article element to the fragment
    t. Return the fragment element

16. displayPosts
    a. Dependencies: createPosts, createElemWithText
    b. Is an async function
    c. Receives posts data as a parameter
    d. Selects the main element
    e. Defines a variable named element that is equal to:
        i. IF posts exist: the element returned from await createPosts(posts)
        ii. IF post data does not exist: create a paragraph element that is identical to the default paragraph found in the html file.
        iii. Optional suggestion: use a ternary for this conditional
    f. Appends the element to the main element
    g. Returns the element variable

NOTE: This is the last group of functions. I call them “procedural functions” because they exist to pull the other functions together in an order that allows the web app to function as it should. This means their sole purpose is to call dependencies with the correct data in the proper order.

17. toggleComments
    a. Dependencies: toggleCommentSection, toggleCommentButton
    b. Receives 2 parameters: (*see addButtonListeners function description*)
        i. The event from the click event listener is the 1st param
        ii. Receives a postId as the 2nd parameter
    c. Sets event.target.listener = true (I need this for testing to be accurate)
    d. Passes the postId parameter to toggleCommentSection()
    e. toggleCommentSection result is a section element
    f. Passes the postId parameter to toggleCommentButton()
    g. toggleCommentButton result is a button
    h. Return an array containing the section element returned from toggleCommentSection and the button element returned from toggleCommentButton: [section, button]

18. refreshPosts
    a. Dependencies: removeButtonListeners, deleteChildElements, displayPosts, addButtonListeners
    b. Is an async function
    c. Receives posts JSON data as a parameter
    d. Call removeButtonListeners
    e. Result of removeButtonListeners is the buttons returned from this function
    f. Call deleteChildElements with the main element passed in as the parameter
    g. Result of deleteChildElements is the return of the main element
    h. Passes posts JSON data to displayPosts and awaits completion
    i. Result of displayPosts is a document fragment
    j. Call addButtonListeners
    k. Result of addButtonListeners is the buttons returned from this function
    l. Return an array of the results from the functions called: [removeButtons, main, fragment, addButtons]

19. selectMenuChangeEventHandler
    a. Dependencies: getUserPosts, refreshPosts
    b. Should be an async function
    c. Automatically receives the event as a parameter (see cheatsheet)
    d. Defines userId = event.target.value || 1; (see cheatsheet)
    e. Passes the userId parameter to await getUserPosts
    f. Result is the posts JSON data
    g. Passes the posts JSON data to await refreshPosts
    h. Result is the refreshPostsArray
    i. Return an array with the userId, posts and the array returned from refreshPosts: [userId, posts, refreshPostsArray]

20. initPage
    a. Dependencies: getUsers, populateSelectMenu
    b. Should be an async function
    c. No parameters.
    d. Call await getUsers
    e. Result is the users JSON data
    f. Passes the users JSON data to the populateSelectMenu function
    g. Result is the select element returned from populateSelectMenu
    h. Return an array with users JSON data from getUsers and the select element result from populateSelectMenu: [users, select]

21. initApp
    a. Dependencies: initPage, selectMenuChangeEventHandler
    b. Call the initPage() function.
    c. Select the #selectMenu element by id
    d. Add an event listener to the #selectMenu for the “change” event
    e. The event listener should call selectMenuChangeEventHandler when the change event fires for the #selectMenu
    f. NOTE: All of the above needs to be correct for you app to function correctly. However, I can only test if the initApp function exists. It does not return anything.

NOTE: There is one last step to get your app to function correctly. I cannot test for this, but you must apply it to call the script into action.

*** This must be underneath the definition of initApp in your file.

1. Add an event listener to the document.
2. Listen for the “DOMContentLoaded” event.
3. Put initApp in the listener as the event handler function.
4. This will call initApp after the DOM content has loaded and your app will be started.