COMP 2132 Final Project

For the final project, we will be building an app that recreates some of the functionality from Instagram: fetching data and displaying posts with images and text, searching and filtering posts, and adding comments and likes. To accomplish this we will use several of the techniques discussed in the course; loops, functions, array methods, data fetching, local storage, dates, and DOM scripting.

The assignment includes some basic page structure and styles. You will provide all functionality described below using JavaScript and submit only your main.js file.

Part 1: Fetching and displaying data

Using the fetch() method, you will retrieve JSON from an API endpoint and use the data to construct the HTML necessary to display posts. Each post will have the following markup, where __USE_DATA__ is a value that you must extract from the fetched data:

```
`<div class="post">
  <header class="post-header">
    <img src="__USE_DATA__" alt="" class="user-icon">
    <p class="user-name">__USE_DATA__</p>
  </header>
  <div class="post-image">
    <img src="__USE_DATA__">
  </div>
  <div class="post-meta">
    <div class="post-meta-actions">
      <img src="images/heart-regular.svg" class="icon icon-like" id="post-meta-like-${post.id}">
      <img src="images/comment-regular.svg" class="icon icon-comment" id="post-meta-comment-${post.id}">
    </div>
    <div class="post-meta-likes">
      Liked by
      <span class="user-name">__USE_DATA__</span>
      and
      <span class="likes-count" id="post-meta-likes-${post.id}">__USE_DATA__ others</span>
    </div>
  </div>
  <div class="post-body">
    <div class="post-body-user">
      <p>
        <span class="user-name">__USE_DATA__</span>
        <span class="post-body-text" id="post-body-${post.id}">__USE_DATA__</span>
      </p>
    </div>
    <div class="post-body-hashtags">__USE_DATA__</div>
  </div>
  <div class="post-comments" id="comments-${post.id}">__USE_DATA__</div>
  <div class="post-date">__USE_DATA__ days ago</div>
  <div class="post-add-comment">
    <input type="text" placeholder="Add a comment..." class="comment-value" id="add-comment-${post.id}">
    <input type="submit" value="Post" class="comment-submit" data-target="comments-${post.id}">
  </div>
</div>`
```

You must also save the fetched data into local storage, so that subsequent page refreshes are not making a network request. This implies that on every page load you must check:

a) is the data in local storage - if so, use the local copy and do not fetch the data

b) if the data is not in local storage, fetch it from the API (https://comp2132.herokuapp.com/posts), then store it in local storage

Once the data is fetched or retrieved from local storage, you must loop through the array of objects, constructing the above markup for each object in the array and appending it to the .innerHTML of the div with class 'posts'.

As you are constructing the markup, you will be required to extract values from the data to fill in the spaces marked with __USE_DATA__. Some of these values can be extracted directly (e.g. image urls, usernames, etc.) and other values will need to be calculated (e.g. the number of days since the post was created, the number of users that have liked the post, etc.)

The values you need to extract are:

- The url of the icon, which will populate the src attribute of the image tag with class 'user-icon'
- The username, which will populate the inner html of the paragraph tag with class 'user-name'
- The url of the main image, which will populate the src attribute of the image tag inside the div with class 'post-image'
- The username of the first like, which will populate the inner html of the span with class 'user-name' inside the div with class 'post-meta-likes'
- The number of likes the post has. This will need to be calculated from the array of likes in the data for each post. The calculated value will populate the inner html of the span with class 'likes-count'. Note that you will also need to concatenate the string ' others' onto the end of this count when the post is created/updated
- The username, as in step 2, which will populate the inner html of the span with class 'user-name' inside the div with class 'post-body-user'

- The body or excerpt of the post, which will populate the inner html of the span with class 'post-body-text'

  For this step you will need to check if the length of the body text is greater than or equal to 70 characters. If it is, use the excerpt instead of the body text and append a span with class 'more' to the end of the excerpt. Example:

  ```
  <span class="post-body-text" id="post-body-1">
    This is the excerpt text... <span class="more">more</span>
  </span>
  ```

  If the body is less than 70 characters, use the body text (ignore the excerpt text) and do not append the span with class 'more'

- The hashtags. For each hashtag in the array of hashtags, you must create the following markup and append it to the inner html of the div with class 'post-body-hashtags'

  ```
  <span class="hashtag">#__hashtag__</span>
  ```

  where `__hashtag__` is the actual hashtag from the data, and is preceded by a # symbol

- The comments. For each comment in the array of comments, you must create the following markup and append it to the inner html of the div with class 'post-comments'

  ```
  <p class="comment">
    <span class="user-name">__username__</span>
    <span class="comment-text">__comment_text__</span>
  </p>
  ```

  where `__username__ and __comment_text__` are the username and comment text from the comment object

- The number of days since the post was created. This numeric value must be calculated from the post's posted date before populating the inner html of the div with class 'post-date'. The string ' days ago' must also be appended to the numeric value

Part 2: Adding functionality via event listeners

Once the content is displaying correctly (there are six posts in total, see screencast for more information) you must add several features to the app:

1. The ability to search by username using the text input in the header. Typing in this input should, in realtime (i.e. no page refreshes, no need to click a button or hit enter), filter the posts such that only posts with matching usernames are showing (see screencast for more info). For this to work correctly, you will need to add a keyup listener to the input with class search that does the following each time the listener is triggered:

   a). Empty out the inner html of the element with class 'posts'

   b). Empty out the inner html of the element with class 'current-hashtag'

   c). Empty out the inner html of the element with class 'current-hashtag-count'

   d). *filter* the default array of posts by the current search value and redisplay the data using the filtered array.

HINT! Because you will need to completely redisplay the posts, and the array of posts will change, it may be a good idea to encapsulate the outputting of a post into a function with parameter...

2. The ability to click any of the hashtags on the page and filter the posts so that only posts with a matching hashtag are displayed. For this to work correctly, you will need to a click listener to the document that does the following each time the listener is triggered:

   a). Check if the classList of the event target contains the class 'hashtag'

   b). If the classList does contain 'hashtag', complete steps b through f starting with getting the inner html of the event target and extracting the tag from it. Extracting the tag can be accomplished by removing the first character (the # symbol). You may want to use the slice() method of the string class to perform this extraction

   c). Empty out the inner html of the element with class 'posts'

   d). *filter* the default array of posts by the tag value from step b and redisplay the data using the filtered array

   e). Fill the inner html of the element with class 'current-hashtag' with the following markup:

   ```
   #__TAG__ <img src="images/times-circle-regular.svg" class="remove icon">
   ```

   where `__TAG__` is the value retrieved from step b

   f). Fill the inner html of the element with the number of matching posts followed by the word 'post' or 'posts' if the number of matching posts is greater than 1

3. The ability to remove the hashtag filter if present. For this to work correctly, add another conditional statement inside the document's click listener you created in step 2 and do the following:

   a). Check if the classList of the event target contains the class 'remove'

   b). If the classList does contain 'remove', complete steps b through e starting with emptying out the inner html of the element with class 'current-hashtag'

   c). Empty out the inner html of the element with class 'current-hashtag-count'

   d). Empty out the inner html of the element with class 'posts'

   e). Redisplay the default array of posts. You may want to store a copy of this default array locally for this reason (instead of re-fetching from local storage)

4. The ability to submit new comments to a post. For this to work correctly, add another conditional statement inside the document's click listener you created in step 2 and do the following:

   a). Check if the classList of the event target contains the class 'comment-submit'

   b). If the classList does contain 'comment-submit', complete steps b through e starting with ensuring that the value of the event target's previous element sibling is not empty (i.e. empty comments are not allowed)

   c). Get a reference to the input element on the page whose id matches the data-target value of the event target. To retrieve the data-target value, use .getAttribute('data-target')

   d). Append the following markup to the inner html of the element retrieved in step c:

   ```
   <p class="comment">
     <span class="user-name">you</span>
     <span class="comment-text">\__VALUE__</span>
   </p>
   ```

   where `__VALUE__` is the non-empty value of the event target's previous element sibling from step b

   e). Empty out the inner html of the event target's previous element sibling

5. The ability to expand excerpts, if an excerpt is present in a post. For this to work correctly, add another conditional statement inside the document's click listener you created in step 2 that does the following:

   a). Check if the classList of the event target contains the class 'more'

   b). If the classList does contain 'more', complete steps b through d starting with getting the id of the post. To get the id of the post, extract the last character of the id of the event target's parent node. HINT: using slice with -1 as the argument will return the last character of a string

   c). Index into the array of default posts using the value from step b and extract the body text

   d). Store the extracted body text in the inner html of the event target's parent node

6. The ability to like a post, increasing the total likes. For this to work correctly, add another conditional statement inside the document's click listener you created in step 2 and do the following:

   a). Check if the classList of the event target contains the class 'icon-like'

   b). If the classList does contain 'more', complete steps b through e starting with getting the id of the post. To get the id of the post, extract the last character as per 5 b), but use the id of the event target, NOT the id of the event target's parent node.

   c). Get a reference to the element on the page whose id matches `post-meta-likes-__ID__` where __ID__ is the id you extracted in step b (e.g. #post-meta-likes-1)

   d). Parse the integer out of the inner html of the element you retrieved in step c and store it in a variable

   e). Increment the value of your variable by one, and use this new value to overwrite the inner html of the element you retrieved in step c. NOTE: you will need to append ' others' to the numeric value, i.e. after clicking, the initial value

   `Liked by aber_madi and 3 others`

   should update to

   `Liked by aber_madi and 4 others`

7. The ability to focus the comment input via the comment icon. For this to work correctly, add (yet) another conditional statement inside the document's click listener you created in step 2 and do the following:

   a). Check if the classList of the event target contains the class 'icon-comment'

   b). If the classList does contain 'icon-comment', complete steps b through d starting with getting the id of the post. To get the id, use the method from 6 b) (i.e. use the event target's id)

   c). Get a reference to the element on the page whose id matches `add-comment-__ID__` where __ID__ is the id you extracted in step b (e.g. #add-comment-2)

   d). Focus the element you retrieved in step c by using the focus() method