Assignment 1: Callback Piazza

Due Friday, May 1st, at 11:59 PM

Overview:

Many Stanford students use the student-teacher messaging board Piazza (https://piazza.com) to ask and answer questions pertaining to a course. In this assignment, your job will be to implement a naive version of Piazza. Since this application won't have server interaction, the questions will both be asked and answered by you; you will not see the questions of your peers or be able to answer them.

Your goal will be to provide users the ability to add and answer questions. Once you finish the baseline functionality, you will then extend it in a way of your choosing based on a set of possible extensions. We expect you to apply many of the concepts you've learned in class, including callbacks, events, event listeners, and dynamic manipulation of the DOM.

Important Utilities:

Throughout the assignment, you'll be using the following APIs, which we introduced in the task list application:

`localStorage` - A JavaScript API that allows you to store key-value pairs. These key-value pairs are persistent, meaning they will continue to exist even if the user navigates to a different page or if the browser is refreshed.

Reference handout:
<https://docs.google.com/document/d/1wPFFH_z_OqtzADL_j4jk1HRx_UqsF9rcEKTB9F3-Aoc/edit?usp=sharing>
Further documentation if you're confused: https://developer.mozilla.org/en-US/docs/DOM/Storage#localStorage

We have provided two helper functions for your `localStorage` needs: `getStoredQuestions` and `storeQuestions`.

`storeQuestions`: This function takes in an array and stores it into `localStorage`. Note that any previously-stored array from calling this function earlier is overwritten.

`getStoredQuestions`: This function takes no arguments and retrieves the stored array from `localStorage`. If there is no array present in `localStorage`, the function stores an empty array and returns it.

Handlebars - A JavaScript library that allows you to render complex HTML templates and insert them into the DOM. It's a remedy for ugly `element.innerHTML = "[large HTML string]";` expressions.

Reference handout:
<https://docs.google.com/document/d/1W7DrKMonguxaPk03i81urvuaS2tq22N45HDchFRjeHE/edit?usp=sharing>
Further documentation if you're confused: http://handlebarsjs.com/

Please read our reference handouts thoroughly; they will help you for this assignment. Note that lectures covered fundamental JavaScript and HTML concepts for this project, but omitted full API descriptions that are necessary for this assignment. We've included these in the reference handouts listed above.

Note: You may **not** use jQuery for this assignment.

Overview of Starter Files:

We have provided you various starter files that implement the HTML structure and JavaScript backbone. You can download them at <http://callbackjs.me/downloads/piazza-starter.zip>. A breakdown of the materials is included below:

Basic HTML (index.html): This file describes the HTML structure, which is divided into two panes. It also provides HTML templates that you should use in conjunction with the Handlebars library to implement many of the required tasks (see "Required Tasks" section below). You may freely modify this file for the purposes of implementing an extension. You should **not** add any IDs or classes to elements; all elements in the HTML can be selected with proper use of common selector functions (e.g. `getElementById()`, `querySelector()`, etc.)

Normalize CSS (css/normalize.css): Do not modify this file! It's a common CSS library used to standardize the default styles across browsers.

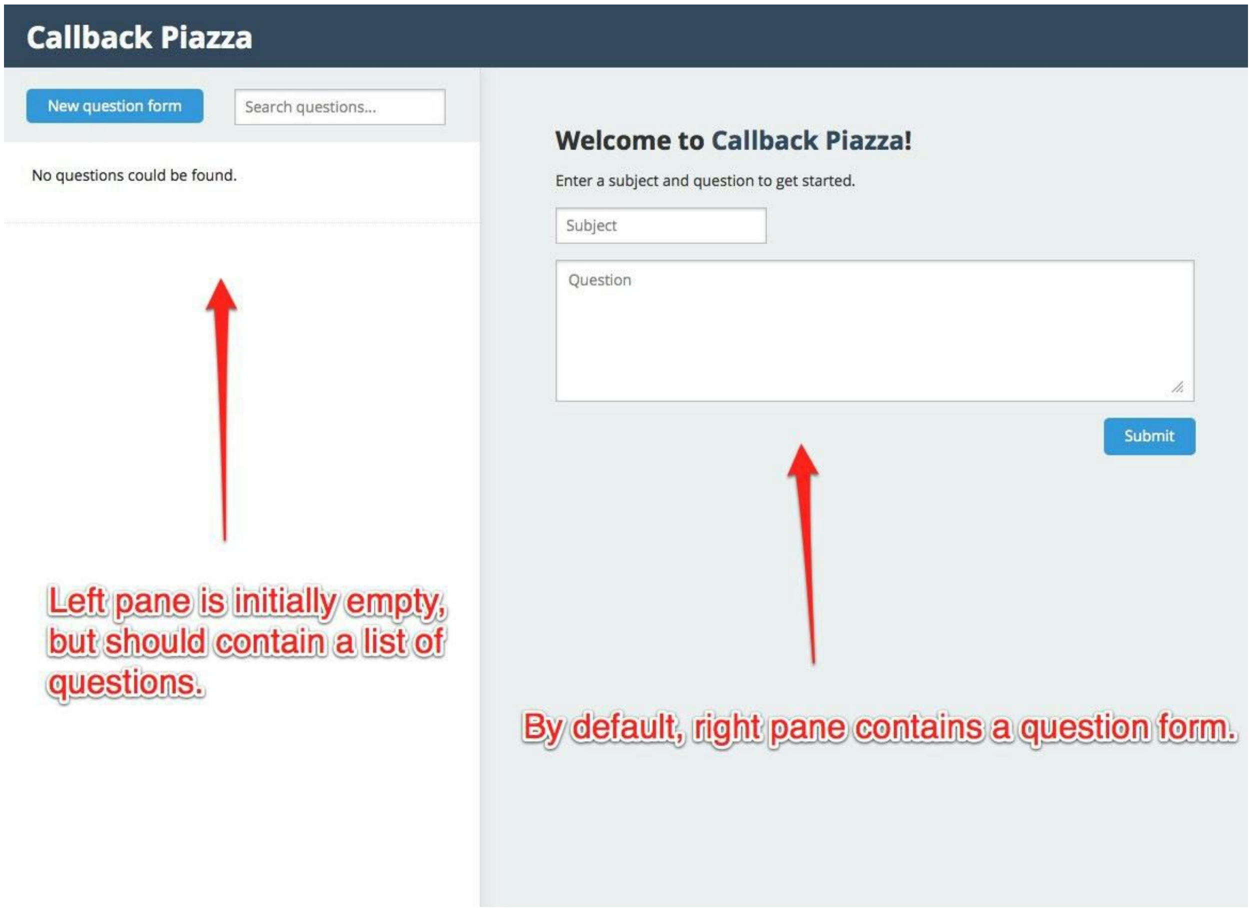
Basic CSS (css/style.css): This CSS file contains the styles for the application. You may modify this file, but you are not required to.

Handlebars (js/handlebars.js): Do not modify this file! Handlebars makes it easy to insert and remove complex HTML into/from documents via JavaScript. You are required to use this library to dynamically add new DOM elements. We have provided you all the templates you'll need in index.html (see index.html section above). Look at the Handlebars reference handout at the end of this document for more information.

Piazza logic (js/script.js): You will modify this file to implement all the required tasks (see section below). This will contain event listeners and associated application logic, including adding questions, adding answers, and searching.

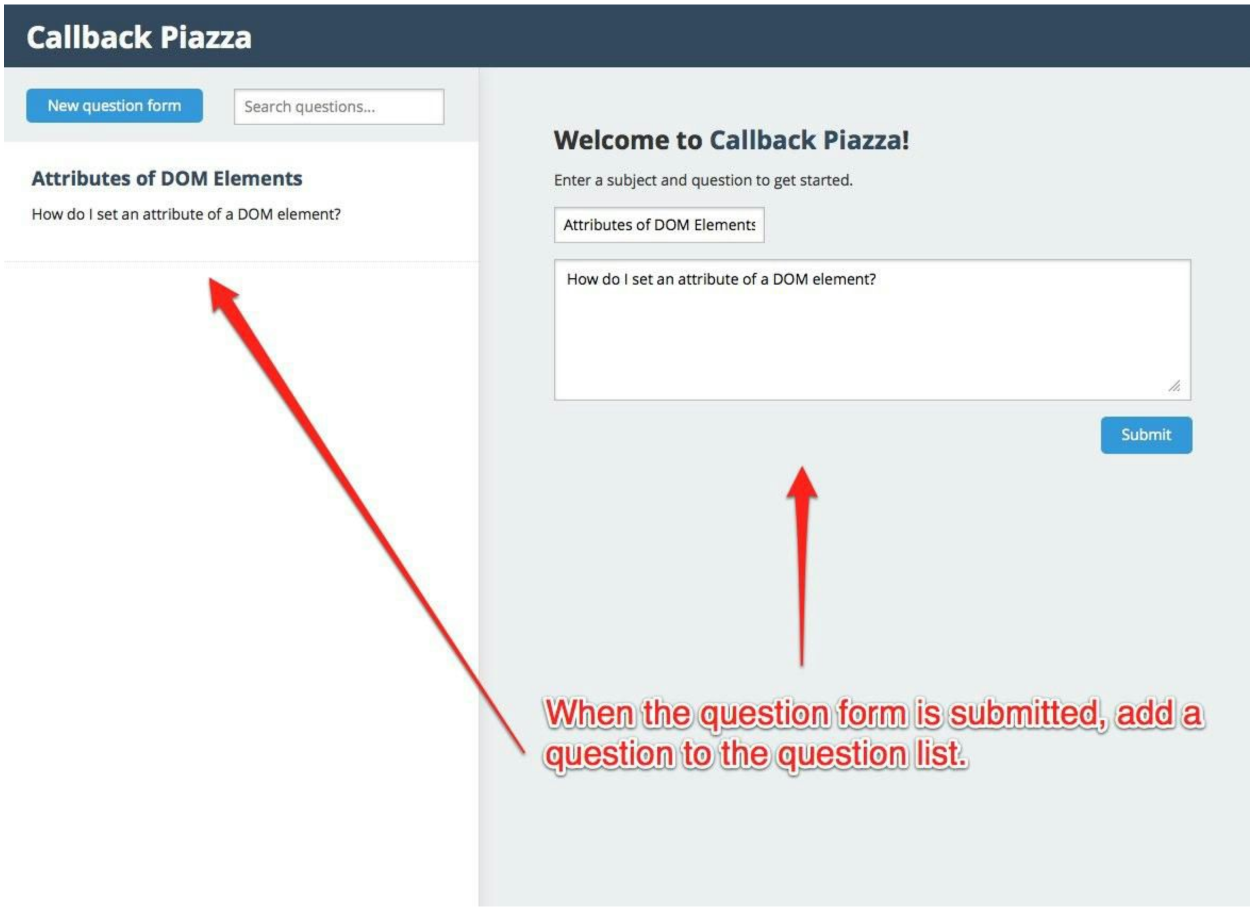
Required Tasks:

If you view the given index.html in your browser, you should see this:


Callback Piazza
New question form
Search questions...
No questions could be found.
Welcome to Callback Piazza!
Enter a subject and question to get started.
Subject
Question
Submit
Left pane is initially empty, but should contain a list of questions.
By default, right pane contains a question form.

The left pane is initially empty, but should contain a list of questions. The right pane, by default, contains a question form. This brings us to the first required piece of interaction:

Task 1: When the question form in the right pane is submitted, add a question to the left pane.


Callback Piazza
New question form
Search questions...
Attributes of DOM Elements
How do I set an attribute of a DOM element?
Welcome to Callback Piazza!
Enter a subject and question to get started.
Attributes of DOM Elements
How do I set an attribute of a DOM element?
Submit
When the question form is submitted, add a question to the question list.

Attach an event listener to the question form tag, `#question-form`, on the right pane. When the form is submitted, prevent the default HTML redirect and use the `#questions-template` Handlebars template to append content to the `#left-pane` tag (see Handlebars reference handout). Then, clear the textbox inputs in the `#question-form`. Make sure to store the question in `localStorage`, and display all stored questions in `#left-pane` when the page is refreshed.

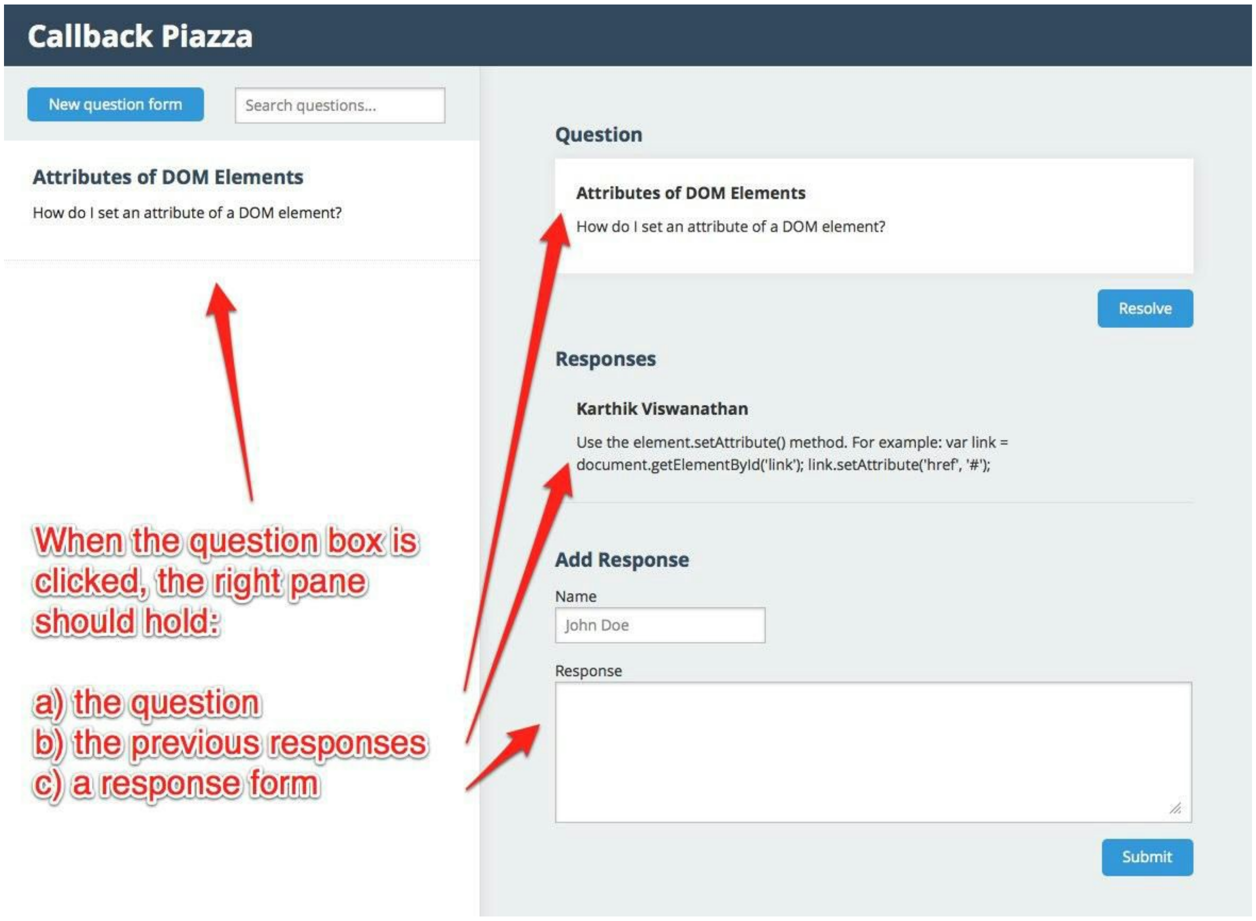
For the `#questions-template` Handlebars template, pass in an object with the following format to the render function:

```
{
      questions: [
            {
                  subject: "Question 1 subject",
                  question: "Actual question text"
            },

            {
                  subject: "Question 2 subject",
                question: "Actual question text 2"
            }
        ]
}
```

The object should have a `questions` key corresponding to an array, and each element inside the array should be an object with the subject and question text.

Task 2: When a question box in the left pane is clicked, display the question, the response form, and the responses in the right pane.


Callback Piazza
New question form
Search questions...
Attributes of DOM Elements
How do I set an attribute of a DOM element?
When the question box is clicked, the right pane should hold:
a) the question
b) the previous responses
c) a response form
Question
Attributes of DOM Elements
How do I set an attribute of a DOM element?
Resolve
Responses
Karthik Viswanathan
Use the element.setAttribute() method. For example: var link = document.getElementById('link'); link.setAttribute('href', '#');
Add Response
Name
John Doe
Response
Submit

When a question is added to the `#left-pane` tag, attach a click event listener to the `.question-info` tag that contains the question information. When this event fires, use the `#expanded-question-template` Handlebars template to display the question, the answer form, and the answers in the `#right-pane` tag (see Handlebars reference handout). The previous answers should be pulled from `localStorage`.

For the `#expanded-question-template` Handlebars template, pass in a question object with the following format to the render function:

```
{
	subject: "Question 1 subject",
	question: "Actual question text",
	responses: [
		{
			name: "Vivek Nair",
			response: "This is an answer."
		},

		{
```

```
                name: "Karthik Viswanathan",
                response: "This is a second answer."
            }
        ]
}
```

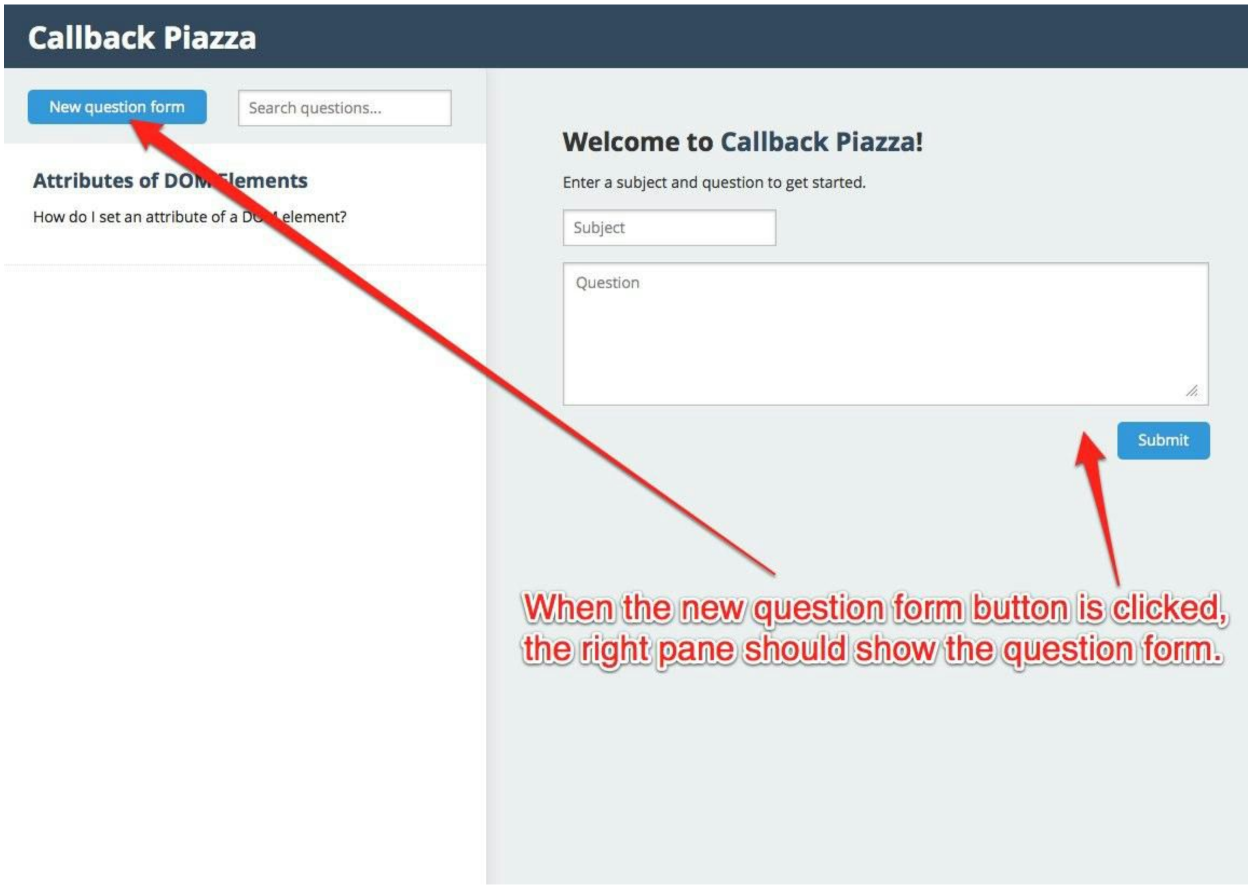
The object should have `subject`, `question`, and `responses` keys. The `responses` key corresponds to an array, and each element inside the array should be an object with the name and response text.

Note: the questions array you store in `localStorage` should contain question objects with exactly this format, as this will simplify rendering the template. We recommend storing the responses along with the question, as shown here, as opposed to storing them separately.

Depending on how you structure your code, you may need to associate an id with each question so you can identify it later on. Feel free to include an id attribute in your question objects. You can generate a unique id using `Math.random()`.

The `#questions-template` Handlebars template (which you used to render the left pane in task 1) puts each question's id into the id attribute of a questions-info div (see index.html). This id attribute can come in handy while trying to identify which question object is associated with a given questions-info div.

Task 3: When the new question form button is clicked, display the question form in the right pane.

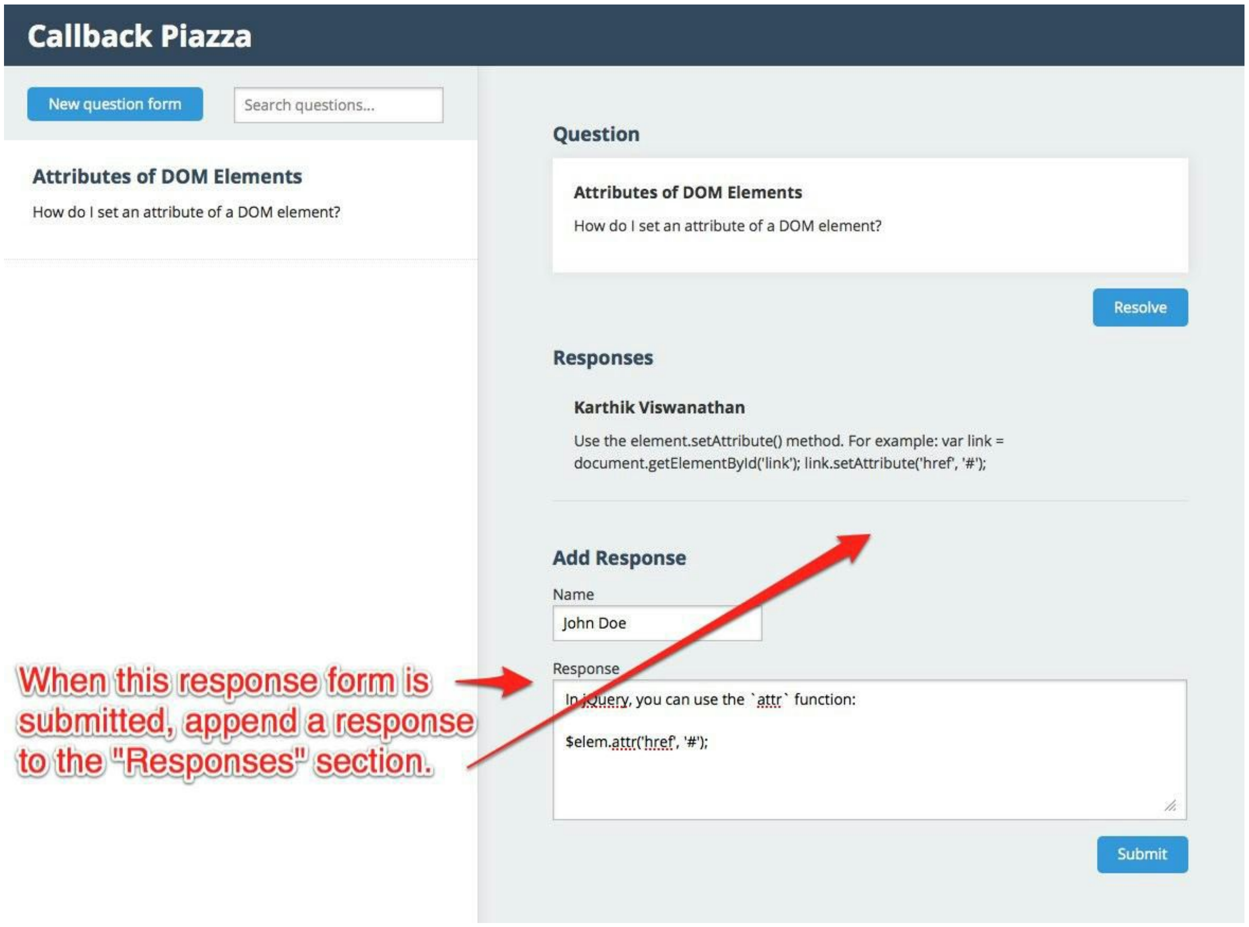

Callback Piazza
New question form
Search questions...
Attributes of DOM Elements
How do I set an attribute of a DOM element?
Welcome to Callback Piazza!
Enter a subject and question to get started.
Subject
Question
Submit
When the new question form button is clicked, the right pane should show the question form.

When the DOM is first loaded, attach a click event listener to the `#interactors .btn` button. When this event is triggered, fill the `#right-pane` tag with the rendered `#question-form-template` Handlebars template (see Handlebars reference handout).

You don't need to pass anything to the render function of the `#question-form-template` Handlebars template.

<u>Task 4:</u> When the answer form is submitted, append an answer to the "Previous Answers" section.

Before submitting:



Attributes of DOM Elements

How do I set an attribute of a DOM element?

Question

Attributes of DOM Elements

How do I set an attribute of a DOM element?

Responses

Karthik Viswanathan

Use the element.setAttribute() method. For example: var link = document.getElementById('link'); link.setAttribute('href', '#');

Add Response

Name: John Doe

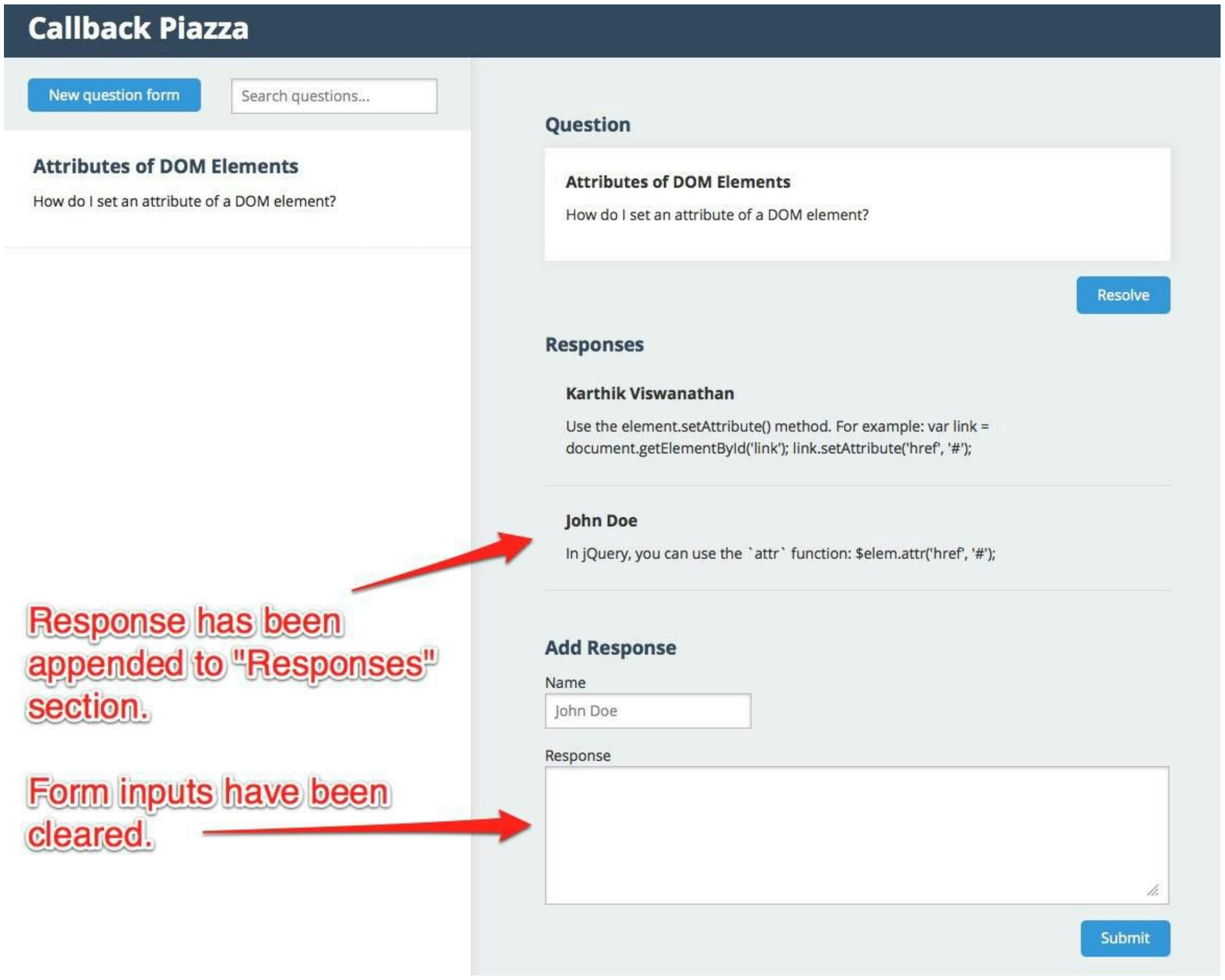
Response:

```
In jQuery, you can use the `attr` function:

$elem.attr('href', '#');
```

When this response form is submitted, append a response to the "Responses" section.

After submitting:


Callback Piazza
New question form
Search questions...
Response has been appended to "Responses" section.
Form inputs have been cleared.
Resolve
Submit

Attributes of DOM Elements

How do I set an attribute of a DOM element?

Question

Attributes of DOM Elements

How do I set an attribute of a DOM element?

Responses

Karthik Viswanathan

Use the element.setAttribute() method. For example: var link = document.getElementById('link'); link.setAttribute('href', '#');

John Doe

In jQuery, you can use the `attr` function: $elem.attr('href', '#');

Add Response

Name

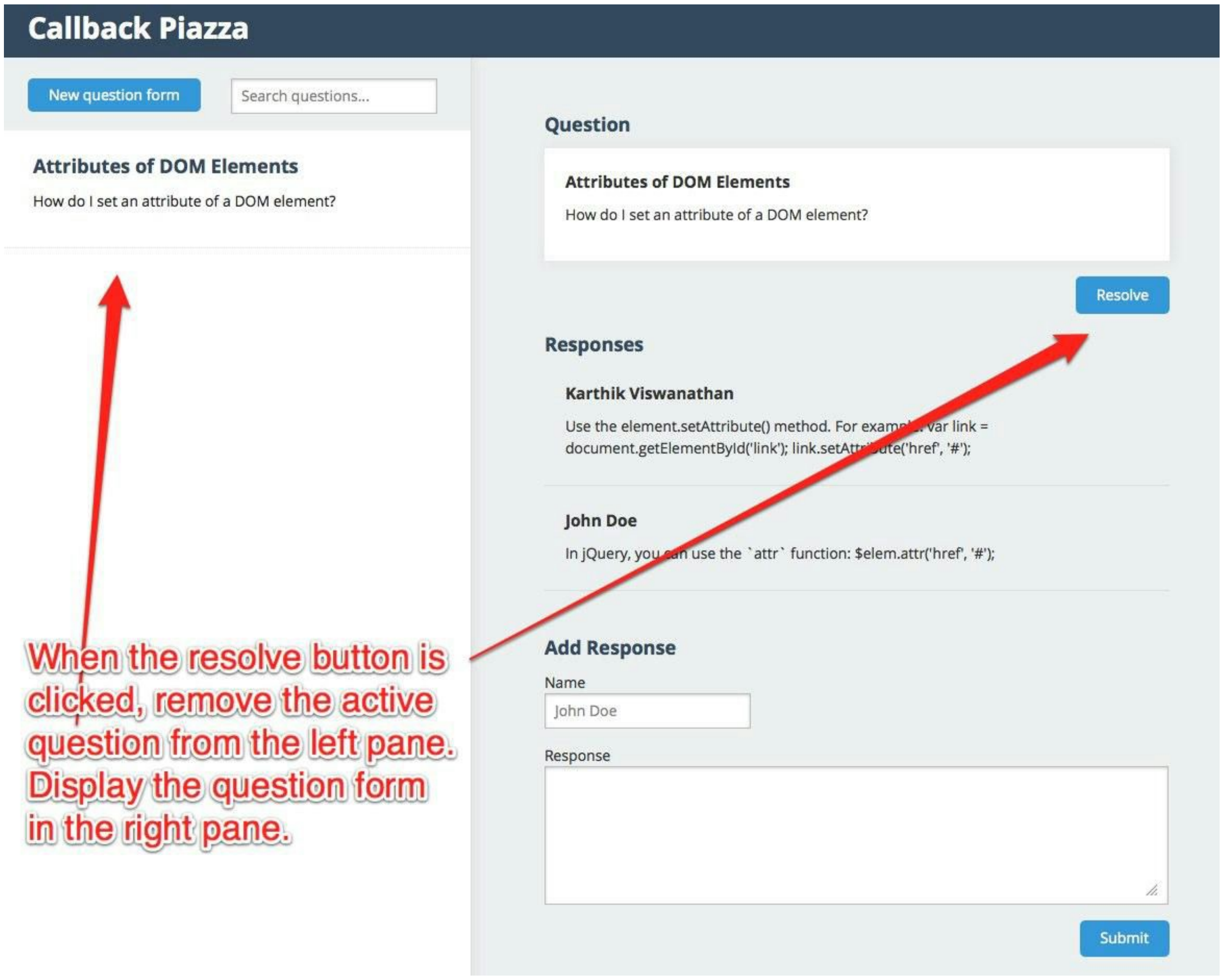
John Doe

Response

After the expanded question information is displayed in the right pane, attach a submit event listener to the response form tag, `#response-form`. When the form is submitted, prevent the default HTML redirect. Then, re-render the `#expanded-form-template` Handlebars template (see task two for more details about this template) so the latest response appears (see Handlebars reference handout). Make sure to store the response in `localStorage`. Clear the textbox inputs in `#response-form`.

<u>Task 5:</u> When the resolve button is clicked, remove the associated question from the question list. Make the right pane display the new question form.

Before resolving:



Attributes of DOM Elements

How do I set an attribute of a DOM element?

Karthik Viswanathan

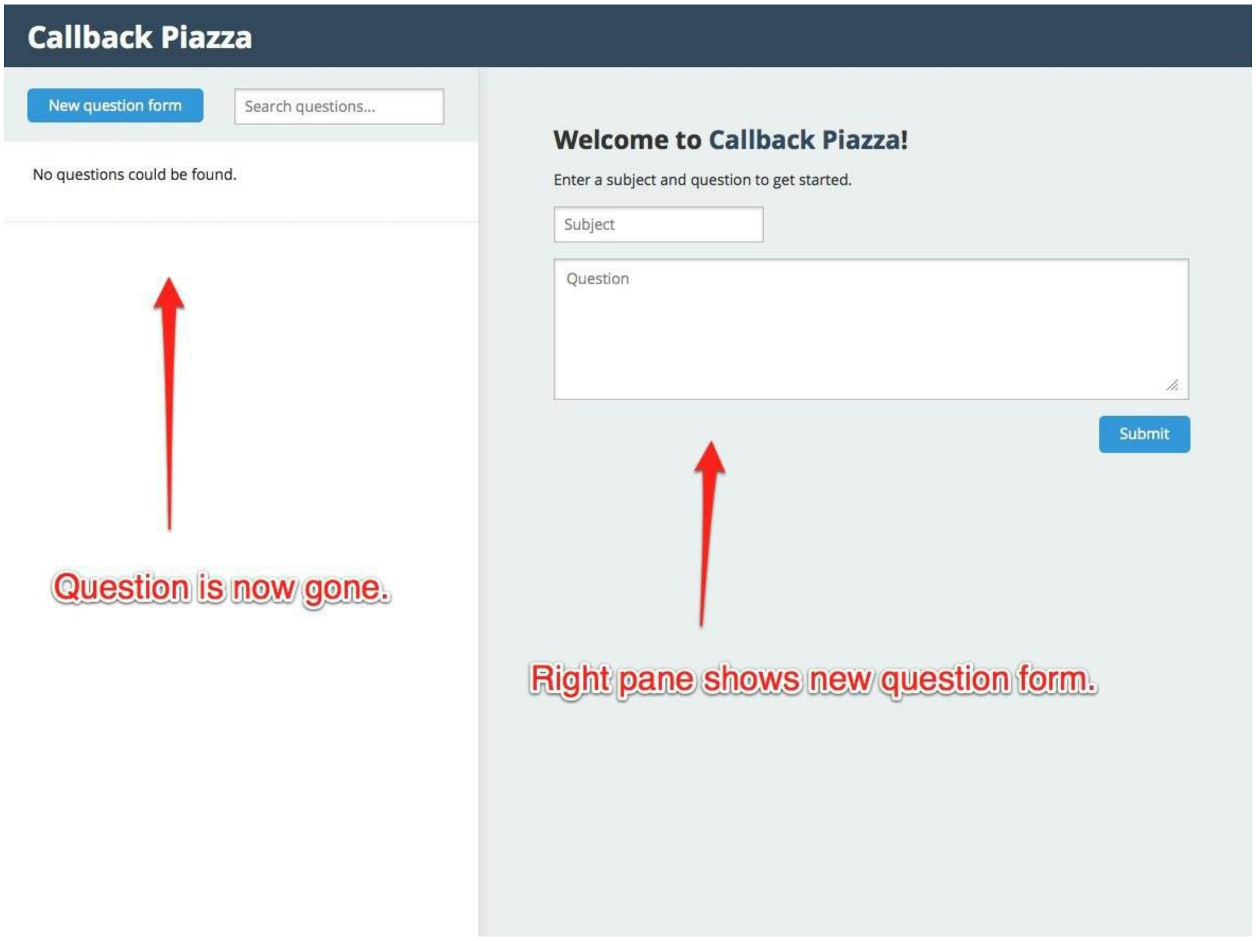
Use the element.setAttribute() method. For example: var link = document.getElementById('link'); link.setAttribute('href', '#');

John Doe

In jQuery, you can use the `attr` function: $elem.attr('href', '#');

When the resolve button is clicked, remove the active question from the left pane. Display the question form in the right pane.

After resolving:


Callback Piazza
New question form
Search questions...
No questions could be found.
Welcome to Callback Piazza!
Enter a subject and question to get started.
Subject
Question
Submit
Question is now gone.
Right pane shows new question form.

When an expanded question is displayed in the right pane, attach a click event listener to the resolve button, `.resolve`. When this event is fired, remove the associated question from `localStorage` and re-render the `#left-pane` using the `#questions-template` Handlebars template (see task one for more details about this template). Re-render the `#right-pane` tag with the `#question-form-template` Handlebars template.

You don't need to pass anything to the render function of the `#question-form-template` Handlebars template.

Keep in mind that all of these features are dynamically implemented in JavaScript. There should be no redirecting throughout the user's interaction with the application, even when panels change!

In order to have persistent storage of data across browser refreshes, we require that you directly use the `localStorage` API. After all, what sort of messaging board would you have if your question disappeared after you refresh?

Extensions:

Finally, **we require that you implement at least one of the extensions** described below or an extension of your choosing. If you decide to do something not listed on this handout, please get it approved by us.

For each assignment, we will give the student with the best assignment a certificate and prize. The assignment will also be featured in-class. Implementing more advanced extensions increases your chances of earning the prize. Of course, we stress quality over quantity; if you implement one extension cleanly and concisely, you will have a higher chance of winning than if you implement many extensions poorly.

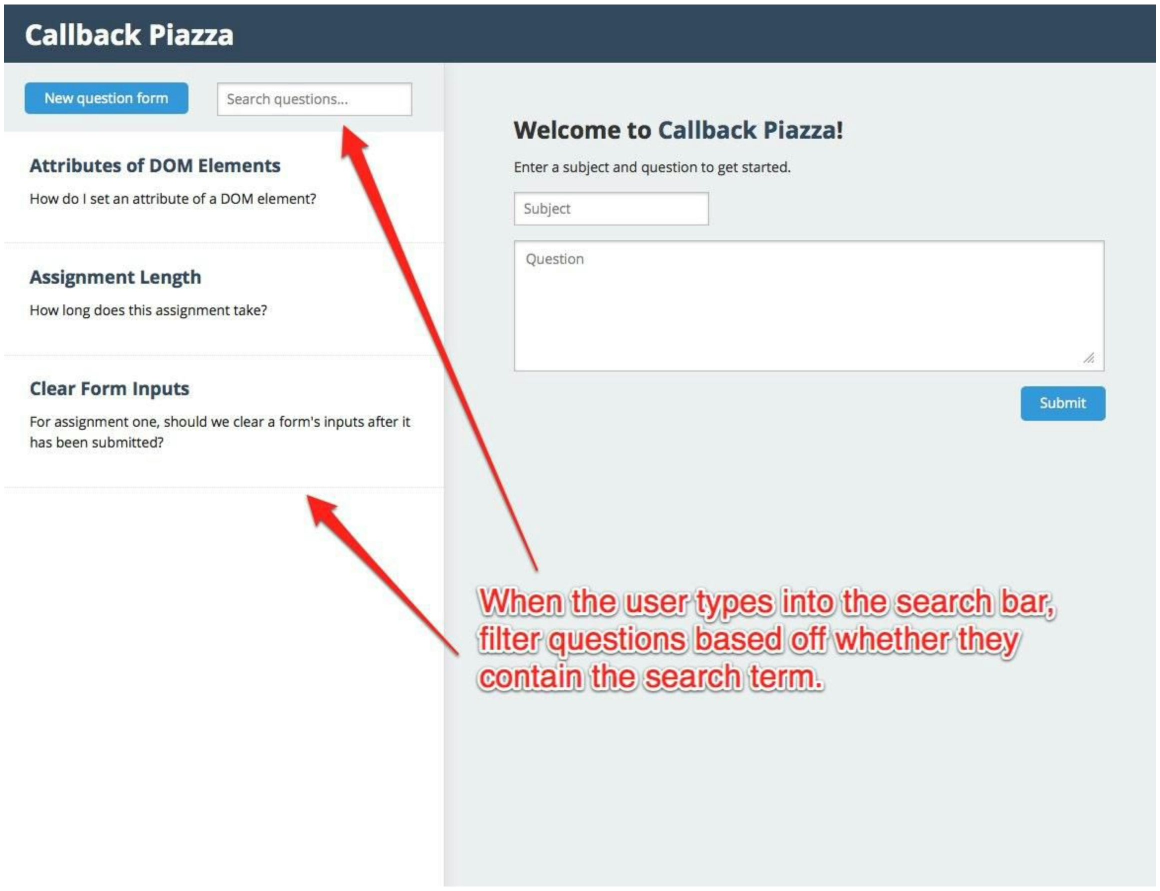
Here is a list of extension ideas, ordered by difficulty. Easier ideas are listed earlier, and harder ideas are listed later. You might find that some of these ideas are more difficult than they seem.

1. **Form Validation**
   Add basic validation to all forms in the application. Ensure the user inputs non-empty text for all required fields. Impose the restriction that the user must enter both a first and last name for answers, and check for this. Add the limit that question descriptions must have at most 500 characters and at least 50. Do the same for answer responses.

   If inputted information does not validate, you should append a descriptive error message in HTML. For example, if a name is not long enough, the error message should explicitly state this. Place the error message near the field that it corresponds to. You should not use the `alert` function in your implementation.

2. **Search through questions**
   When the search input tag, `#search`, is edited, filter the questions in the left pane to only those that match the search terms. Remove all filtering if the search input becomes empty.

   Before search:



Callback Piazza

Attributes of DOM Elements

How do I set an attribute of a DOM element?

Assignment Length

How long does this assignment take?

Clear Form Inputs

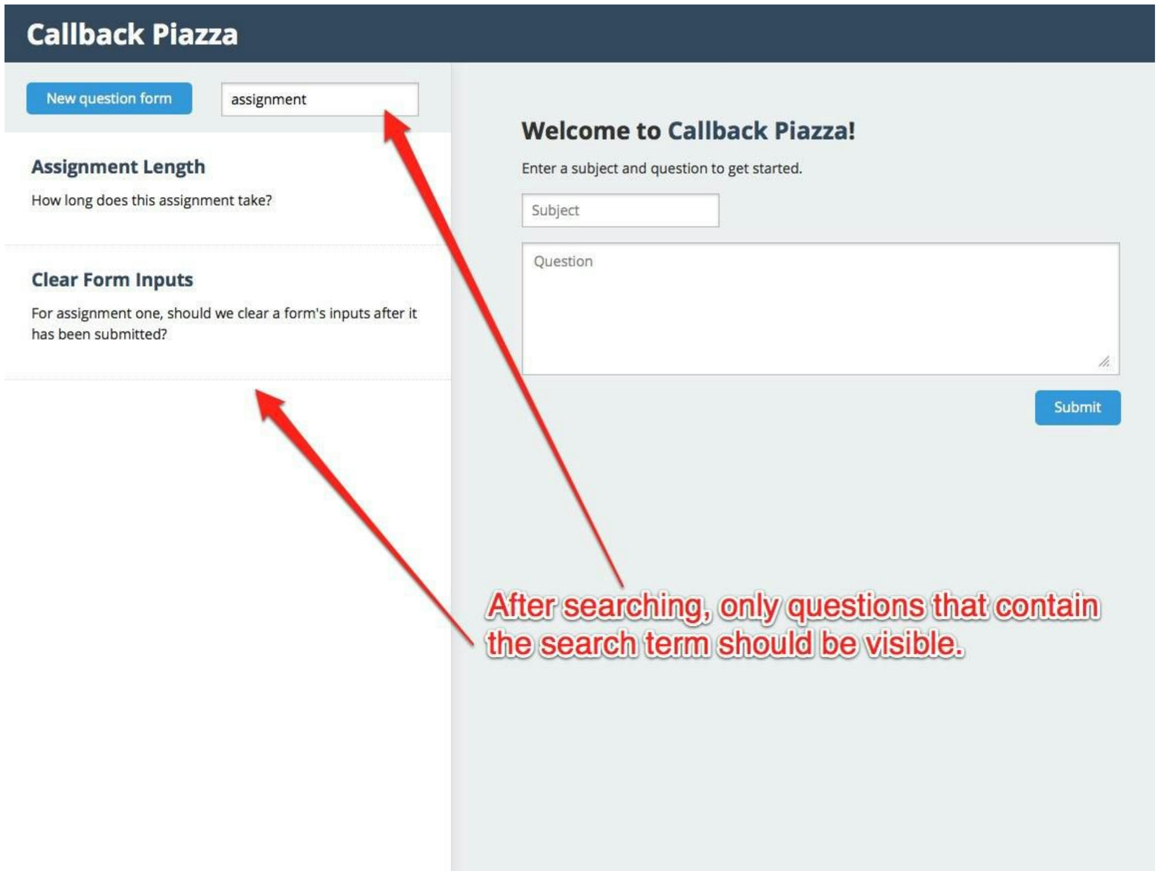
For assignment one, should we clear a form's inputs after it has been submitted?

Welcome to Callback Piazza!

Enter a subject and question to get started.

When the user types into the search bar, filter questions based off whether they contain the search term.

After search:


Callback Piazza
New question form
assignment
Assignment Length
How long does this assignment take?
Clear Form Inputs
For assignment one, should we clear a form's inputs after it has been submitted?
Welcome to Callback Piazza!
Enter a subject and question to get started.
Subject
Question
Submit
After searching, only questions that contain the search term should be visible.

When the DOM is loaded, attach a keyup event listener to the search input. In the associated event handler, retrieve the inputted text. Then:

a) If there is no inputted text, remove all filtering and display all questions.
b) Otherwise, filter the questions array to only those elements that contain the search term. Then, set the contents of the `#left-pane` tag to the rendered Handlebars `#questions-template`, passing in the filtered list of questions. The `Array.filter()` <https://developer.mozilla.org/en-US/docs/JavaScript/Reference/Global_Objects/Array/filter> and `String.indexOf()` <https://developer.mozilla.org/en-US/docs/JavaScript/Reference/Global_Objects/String/indexOf> methods will come in handy here.

If a question that has been filtered out is being viewed in the right pane, remove that question and display the new question form instead (just like required task 3).

3. **Favoriting questions or responses**
Add a favorite button associated with each question or each response. Display questions that are favorited before other questions in the question listing. Favorited questions should have a star (or some other symbol), indicating that it has been

favorited.

4. **Up/down voting of questions or responses**
   Add a count associated with each question or each response that represents how many up-votes it has (negative numbers correspond to down votes). Add buttons that allow the user to up and down vote questions or responses. Display questions/responses with more up votes earlier than questions/responses with fewer up votes.

5. **Highlighting of terms as the user searches for questions**
   When the user searches for a question, highlight the search term in all questions that appear within the search results. For example, if there is a question with the subject `"DOM modifications"`, and the user searches for `"mod"`, highlight the `"mod"` in `"modifications"`.

   To accomplish this, first display the questions that match the search term normally via the Handlebars template. Then, go through each tag that either contains the subject or description of a question. If you see the search term within one of these tags (check its `innerHTML`), wrap the search term within a `<span>` tag. Give this `<span>` tag a CSS class, and add the appropriate styling (set the CSS `background-color` property to some light yellow color) to make the search term highlighted.

   For example, in the previous example, `"DOM modifications"` would be changed to `"DOM <span class="highlight">mod</span>ifications"`. The `.highlight` class in CSS would then contain a `background-color: #FFFFED;` property (`#FFFFED` corresponds to light yellow).

6. **Organizing questions by category**
   Piazza currently has tags for questions, which are used to place questions into different categories. Implement such a tagging system.

   Use javascript 'split' function for strings

   First, add a "tags" input to the question form. The user should be able to associate multiple tags with a question via this input. For example, a user might type "CSS selectors, JavaScript, HTML" into this input, meaning that he/she wants the question to be tagged with three keywords: "CSS selectors", "JavaScript", and "HTML".

   Display the list of tags underneath the description of each question. Do this both for the question listing in the left pane and the expanded question in the right pane. To accomplish this, you'll need to modify the Handlebars templates we provided.

   Allow users to filter by tag. Maintain a global list of all tags that have ever been entered, and display this list somewhere in the application. If the user clicks on one of these tags, he/she should only see questions that have that tag. You must provide the user a way to stop filtering by tag and return to the normal question listing.

You have creative license in how these features are implemented. Don't hesitate to modify the existing HTML or the CSS stylesheet. Think about what features you'd like to see; that is, what have you always wanted Piazza to have?

Grading:

We will grade your application based on two criteria: the functionality of your application and proper JavaScript coding style.

Functionality (60%):
You must implement all the required tasks and at least one extension. Given that you have done so and tested for robustness (e.g. if all questions are deleted, your local storage shouldn't get corrupted), you should receive full marks. We will deduct points for incomplete/omitted features and lack of robustness.

Style (40%):
Since we have concentrated on the basics of JavaScript during lectures and have not extensively covered proper design patterns, we will not focus on proper design for this assignment. Style grading will in part be based on JavaScript syntax (e.g. no inline JavaScript in HTML), event instantiation, and other topics covered in lecture. Given that asynchronous JavaScript code can become unwieldy, we will expect you to comment your code extensively.

Additionally, your style will largely be graded on consistency. Follow the style already present in the starter files. If you go on to be a JavaScript engineer in industry, the team you'll be working with will likely have a baseline style established. You should always match this style. Consistency is everything.

Hints

Assignment Length / Line Count:
The reference solution, which implements the search extension, is approximately 150 lines of code (lines are those with semicolons, colons, or opening/closing braces; not including comments). Good solutions will likely range from 100-250 lines of code. Exactly how long the assignment will take depends on the person; our estimate is 8-15 hours, but some people may take more, and others may take less.

Click and submit events:
By default, link click and form submit events will navigate to another page. For this assignment, however, we don't want this functionality; the entire application should work without any new page loads or browser refreshes. In turn, you'll need to prevent the default click/submit functionality. To do so, simply call `event.preventDefault()` in the appropriate click/submit handler (assuming `event` is the event object passed to the handler).

Event listeners on removed elements:

If you attach an event listener to a DOM element, and that element is removed, the event listener will be removed with it. This occurs regardless of whether a similar element is re-inserted into the DOM. Consider the following code sample:

```
var element = document.getElementById("element");

// log when the element is clicked
element.addEventListener("click", function() {
      console.log("Clicked the element.");
});

// remove the element
var parentNode = element.parentNode;
parentNode.removeChild(element);

// create an element with same ID and add it to the parent node
var newElement = document.createElement("div");
newElement.id = "element";
parentNode.appendChild(newElement);

// clicking the the #element div will not log anything now;
// when `element` was removed via `parent.removeChild(element)`,
// `element`'s event listeners went with it
```

You will run into this issue often. Each time you change the right pane to display question details (the question itself, the answer form, and answers), you'll need to reattach an event listener to the answer form. In like manner, when the user clicks the new button to add a new question, you'll need to reattach an event listener to the new question form.

The above strategy is the most straightforward way to handle the problem of removed event listeners. There is, however, one other methodology where you don't need to reattach listeners at all. It's known as event delegation, and is a cleaner solution to the above problem (see section below for more details). You're free to use either strategy, however.

Event Delegation
The removed event listeners problem mentioned in the prior section can be solved by using event delegation, which we introduced while creating the task list application. You are not required to use event delegation, but it can make your code much simpler. We've created an event delegation handout in case you want to employ it for this assignment. It's available on the course website (<https://docs.google.com/document/d/1-7e5Rbgw-y1tIntUEWL-sB8gs2FzaE1HbhtEz4mb_-U/edit?usp=sharing>).

Handlebars

Much of this assignment revolves around rendering Handlebars templates. We've compiled a handout on Handlebars for your reference, which is available on the course website (<https://docs.google.com/document/d/1W7DrKMonguxaPk03i81urvuaS2tq22N45HDchFRjeHE/edit?usp=sharing>).

Additionally, in the starter code for this assignment, we have given you an example usage of a Handlebars template that you may find useful throughout your development. We've also given you all the templates you'll need to finish the assignment.

The code we've provided compiles the `#question-form-template` Handlebars template and, when the DOM loads, renders it into the `#right-pane` tag. This adds the question form to the right pane when the application starts.

Submission:

Visit <https://web.stanford.edu/class/cs42/cgi-bin/piazza.php> and submit a zip file of your entire piazza assignment directory (all starter files should be included in the zip). You may submit as many times as you'd like before the deadline; only your last submission will be retained.

Honor Code:

All of your code for this assignment should be your own original work. Do not copy other students' work. If you referenced online sources, cite them as a comment in your code.