Assignment 1: Callback Piazza

Due Thursday, May 10th 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.

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.

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. 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.

Templates (js/template.js): This file provides the tag function, as implemented in the task list in-class activity, and several template functions to simplify your development. You may need to modify this file for your extension, but not for tasks 1 - 5.

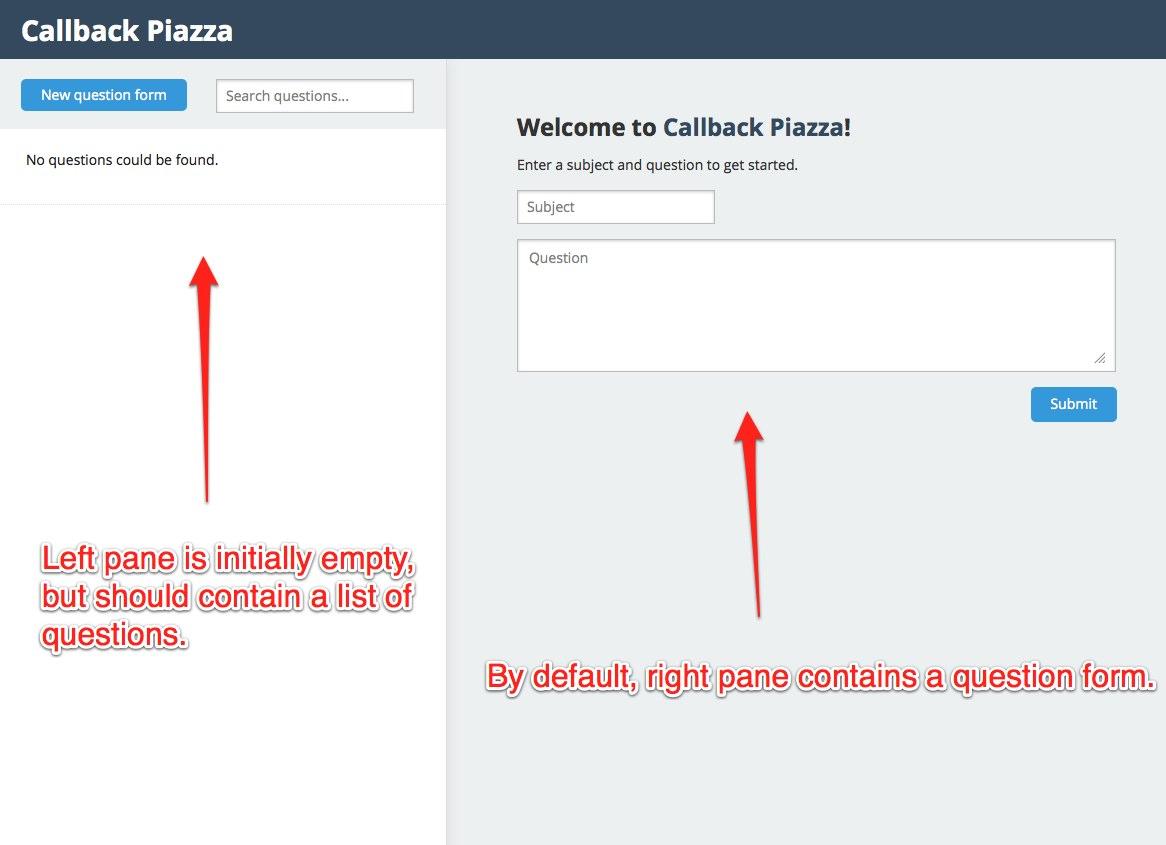
The tag function creates and returns an HTMLElement representing a tag with the provided name, attributes, and contents. It’s a remedy for ugly element.innerHTML = "[large HTML string]"; expressions. See the [templating handout](https://docs.google.com/document/d/1dRq-I9_6OqK9mbQMlB3l5UMtlHUE69-2dSVy2M-75yQ/edit?usp=sharing) if you’d like to know how it works.  
  
For this assignment, however, you shouldn't use the tag function directly to modify the DOM. Instead, you should use the template rendering functions defined in js/templates.js—namely renderQuestionForm, renderQuestions, and renderExpandedQuestion—to construct the necessary HTMLElement to insert into the DOM.

We'll cover the exact usage of these functions later in this handout.

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.

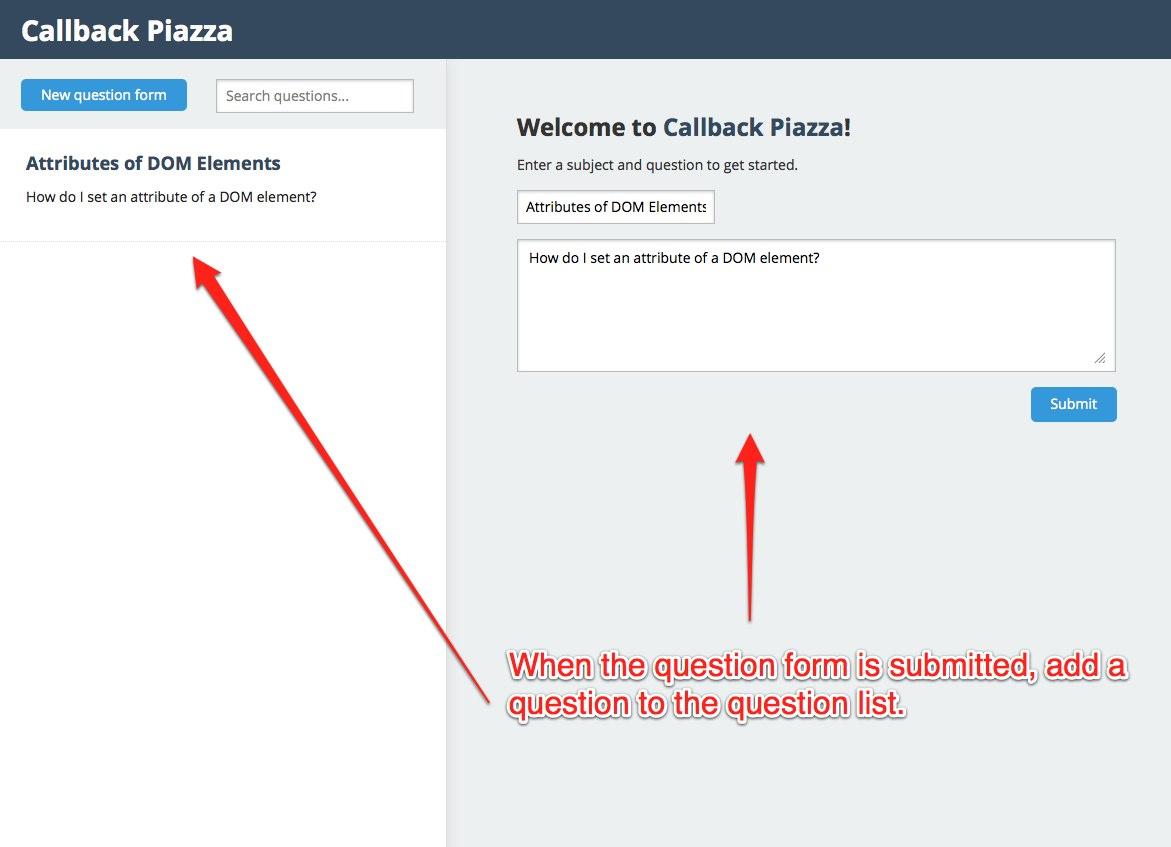
Required Tasks:

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



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.



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. Then, call renderQuestions (defined in js/templates.js) and put the result in the #left-pane tag, emptying any existing contents. 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.

When calling renderQuestions, pass in a questions array of the following format:

[

{

subject: "Question 1 subject",

question: "Actual question text",

},

{

subject: "Question 2 subject",

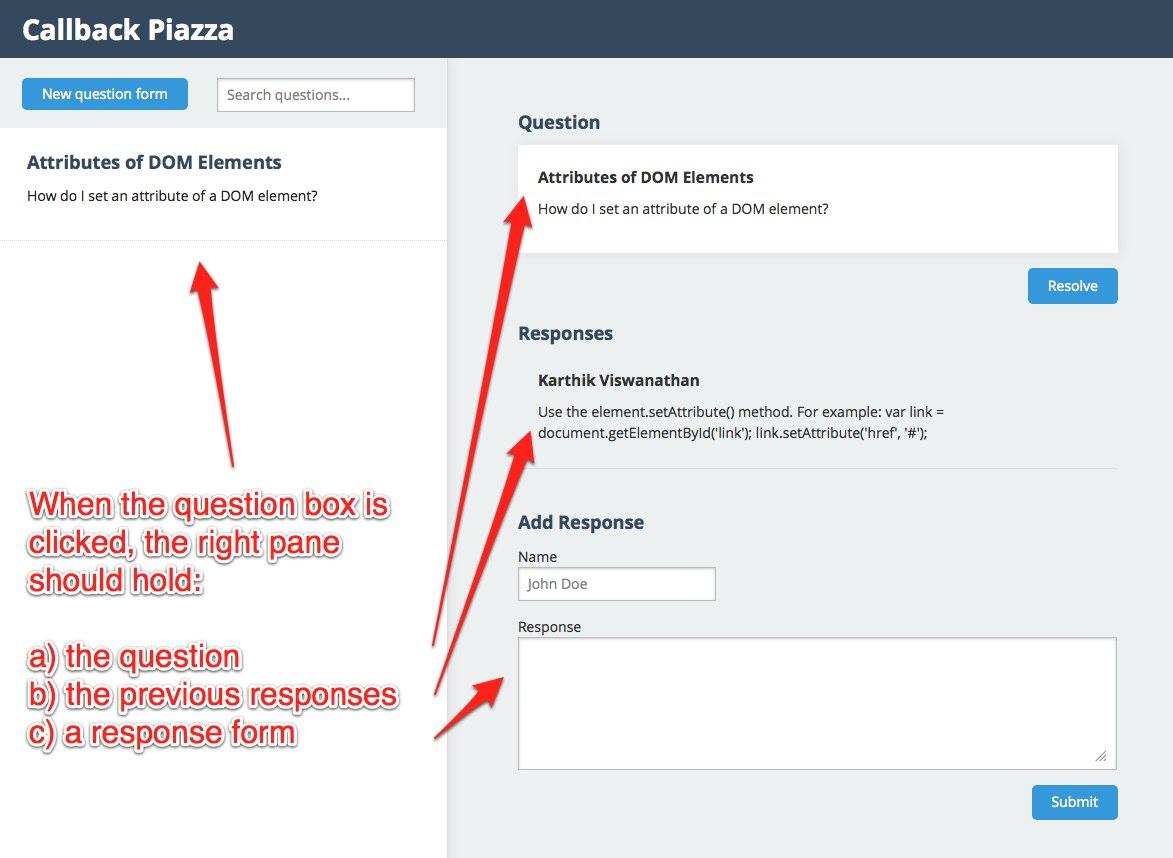
question: "Actual question text 2",

},

]

Each element inside the array is an object with 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.



When a question is added to the #left-panetag, attach a click event listener to the .question-info tag that contains the question information. When this event fires, call renderExpandedQuestion (defined in js/templates.js) and put the result in the #right-pane tag, emptying any existing contents. This should display the question, the previous responses, and a response form.

When calling renderExpandedQuestion, pass in a question object of the following form:

{  
 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.",

},

],

}

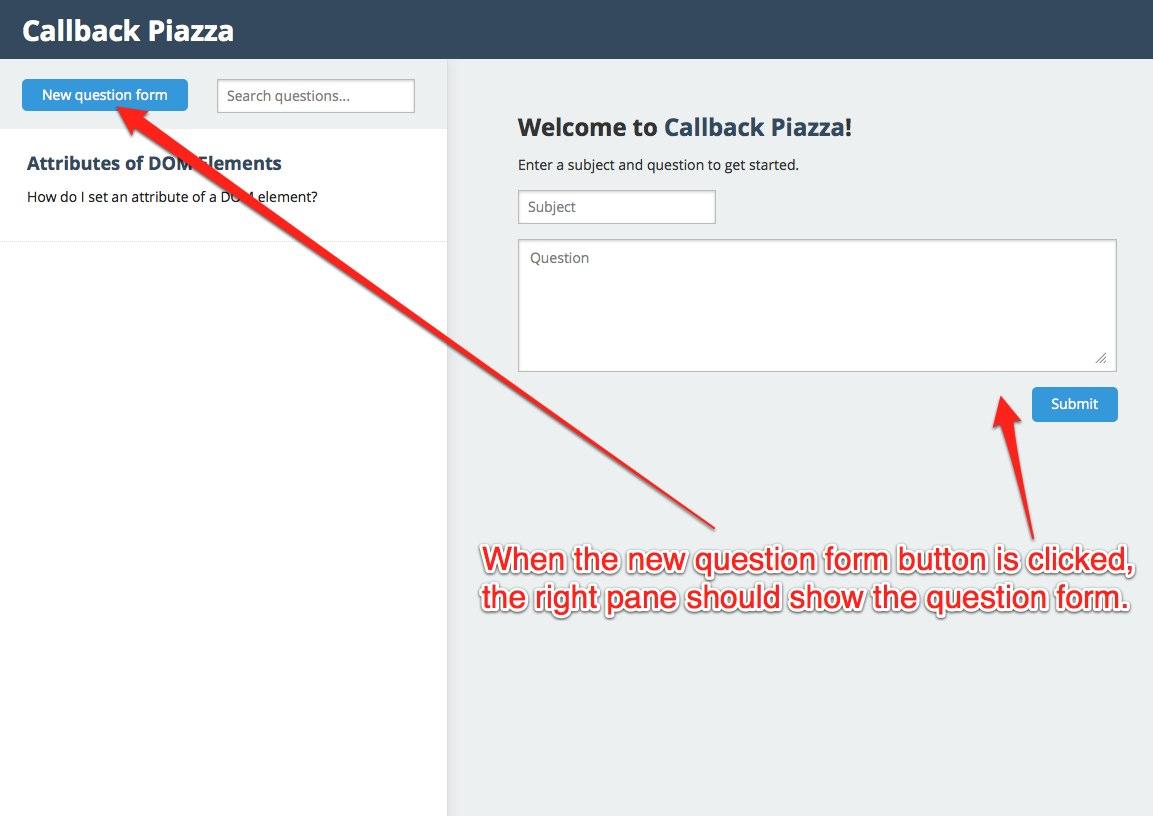
The object should have subject, question, and responses keys. The responses key corresponds to an array, and each element inside the array should be an object with name and response text.

**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 renderQuestions function (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 js/templates.js). 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.

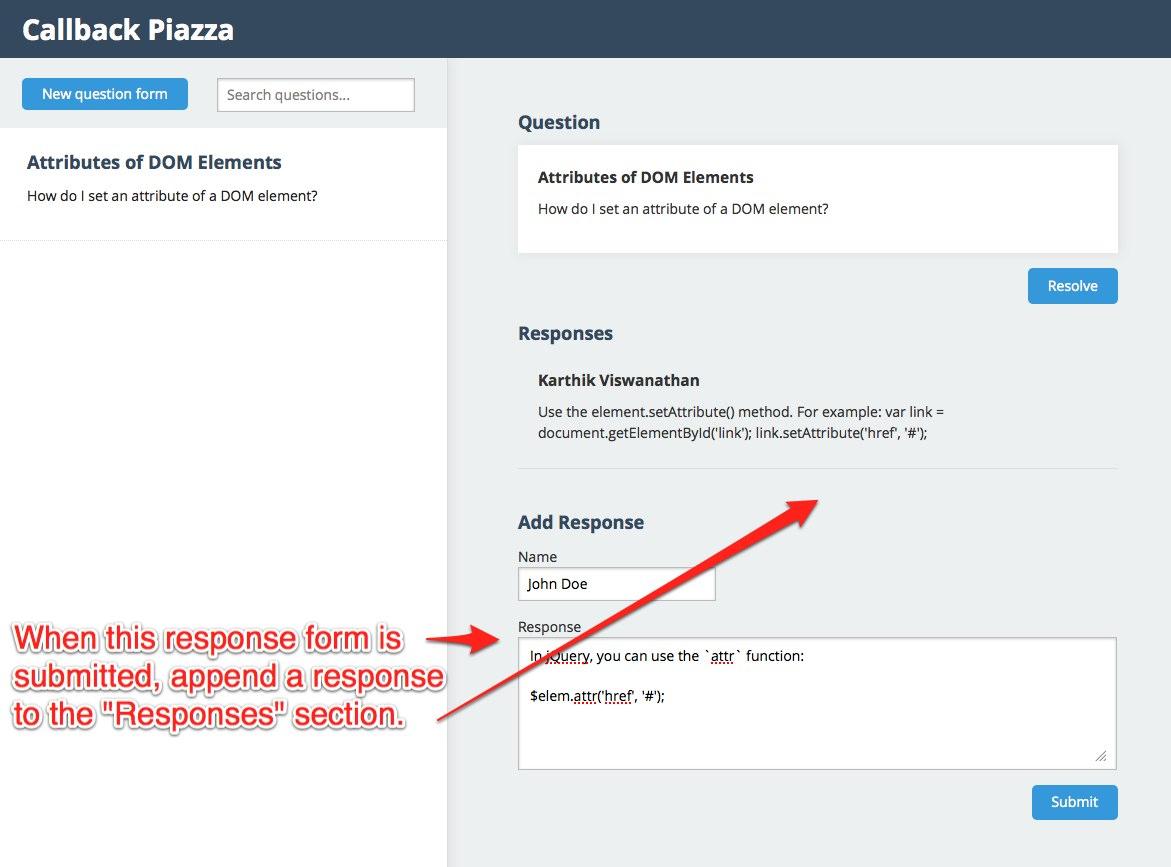
****

When the DOM is first loaded, attach a click event listener to the #interactors .btn button. When this event is triggered, call renderQuestionForm (defined in js/templates.js) and put the result in the #right-pane tag, emptying any existing contents.

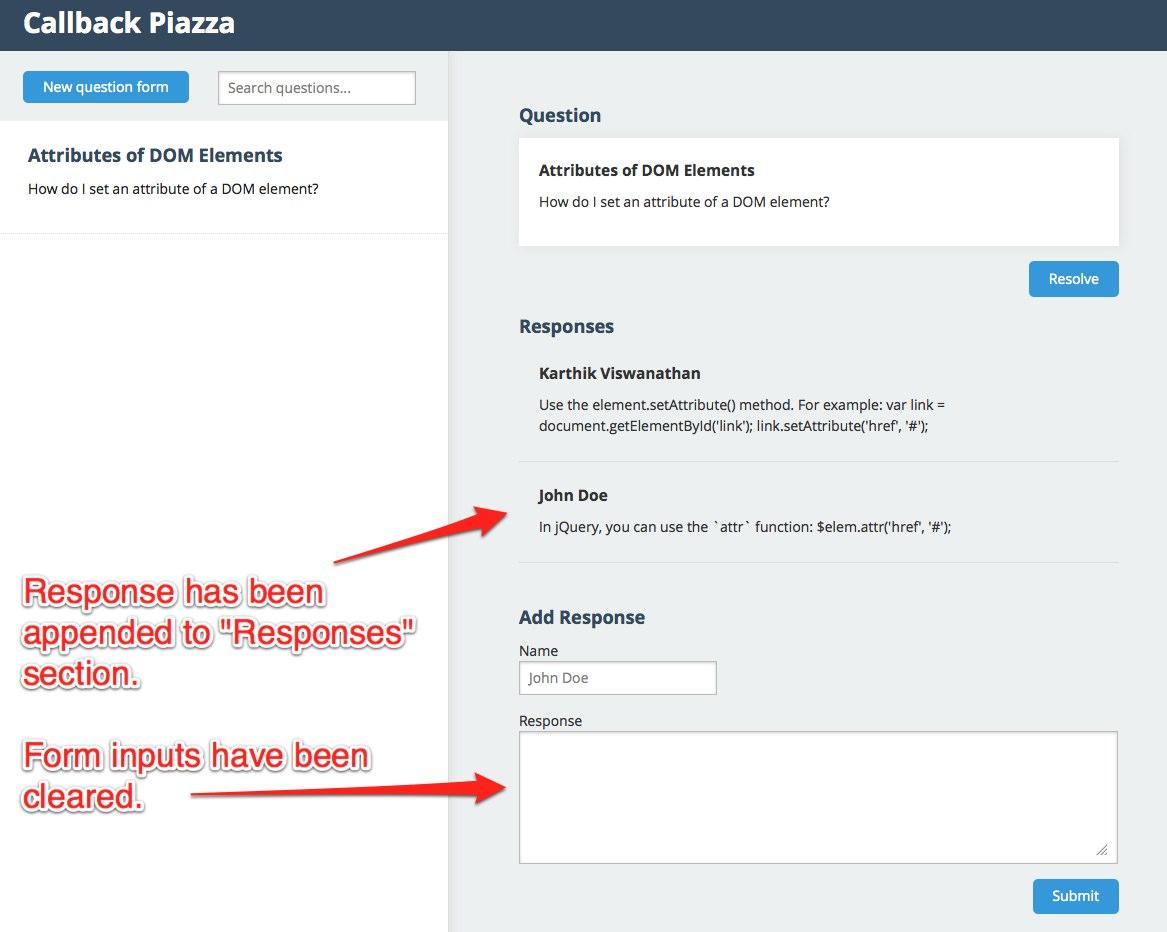
You don’t need to pass any parameters to the renderQuestionForm template function.

**Task 4:** When the answer form is submitted, append an answer to the “Previous Answers” section.

Before submitting:

****

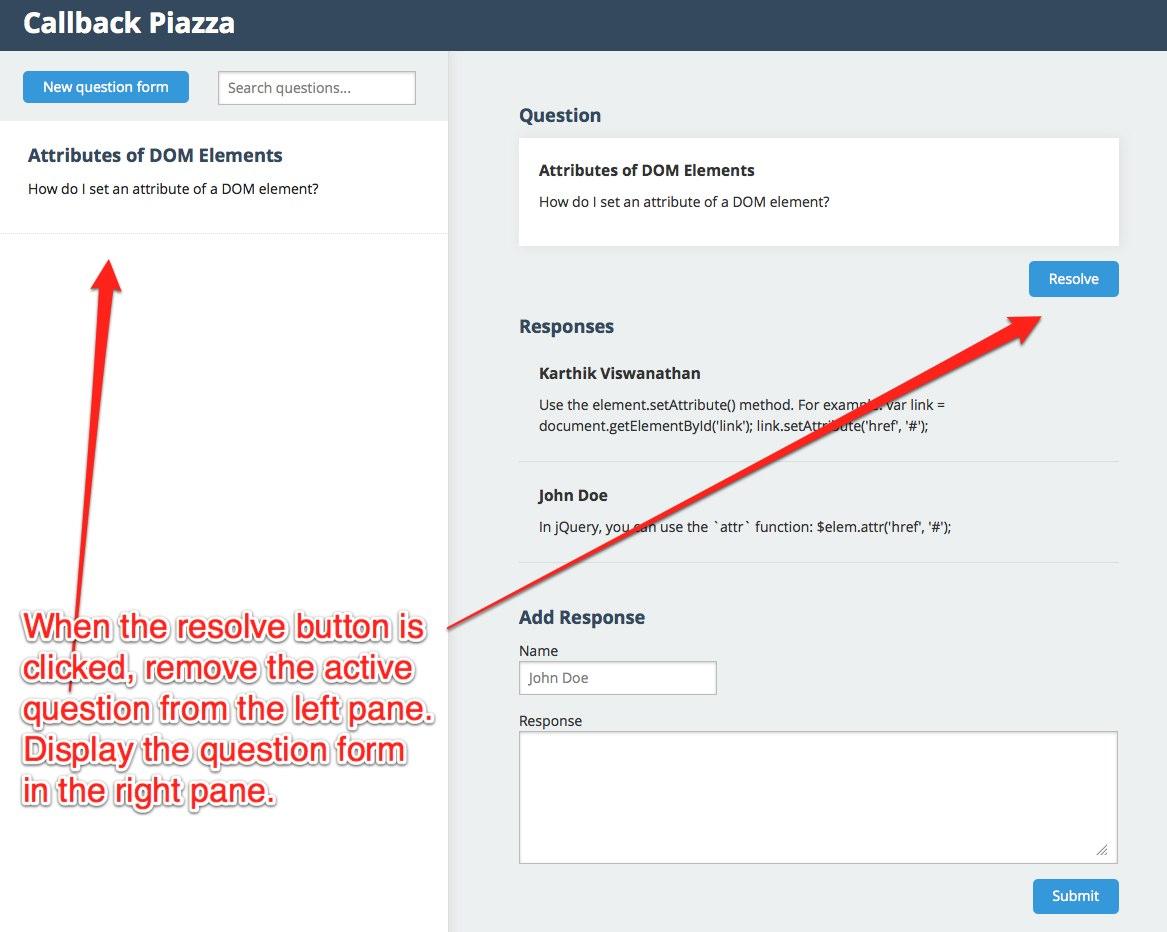
After submitting:

****

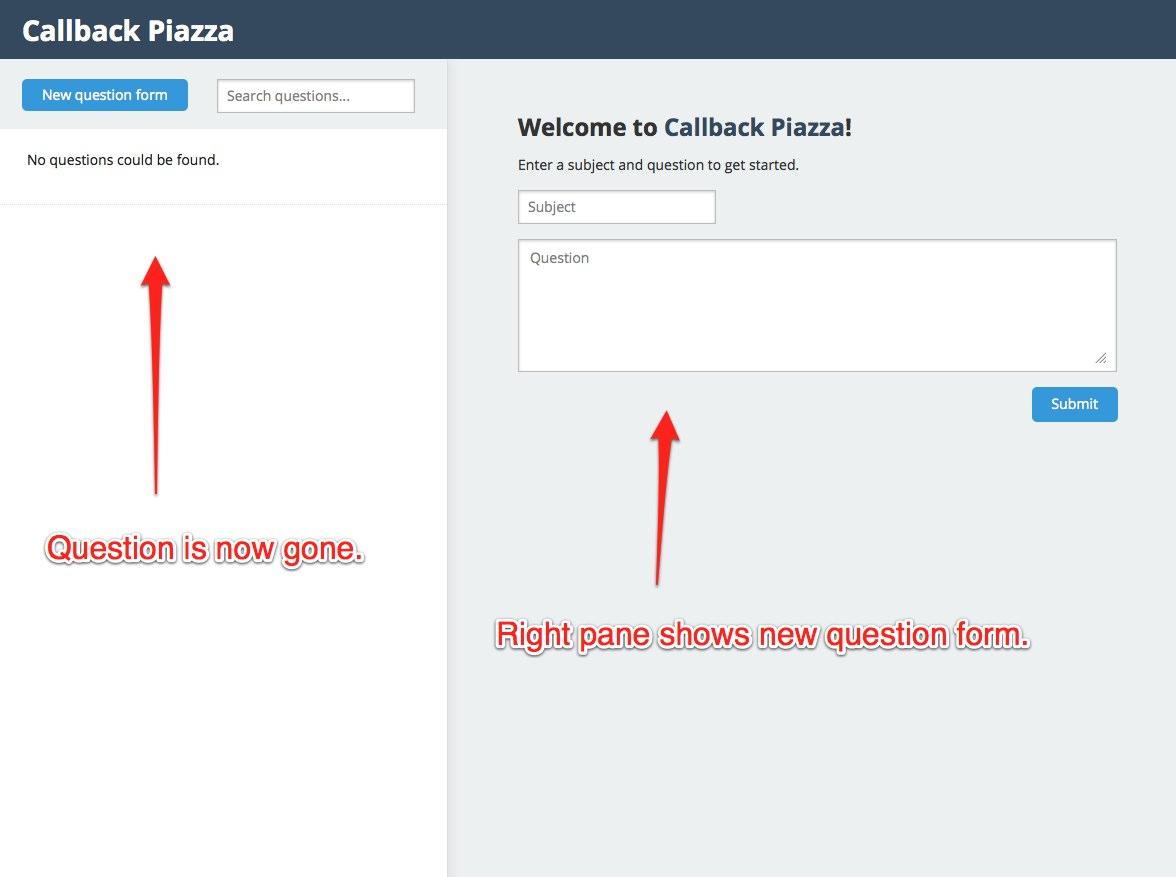
After the expanded question information is displayed in the right pane, attach a submit event listener to the response form tag, #response-form. When the form is submitted, prevent the default HTML redirect. Then, re-render the #right-pane using renderExpandedQuestion, just as you did in task 2, so the latest response appears. Make sure to store the response in localStorage. Clear the text inputs (name and response) in #response-form.

**Task 5:** 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:

****

After resolving:

****

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 renderQuestions function (see task 1). Re-render the #right-pane tag with the renderQuestionForm function (see task 3).

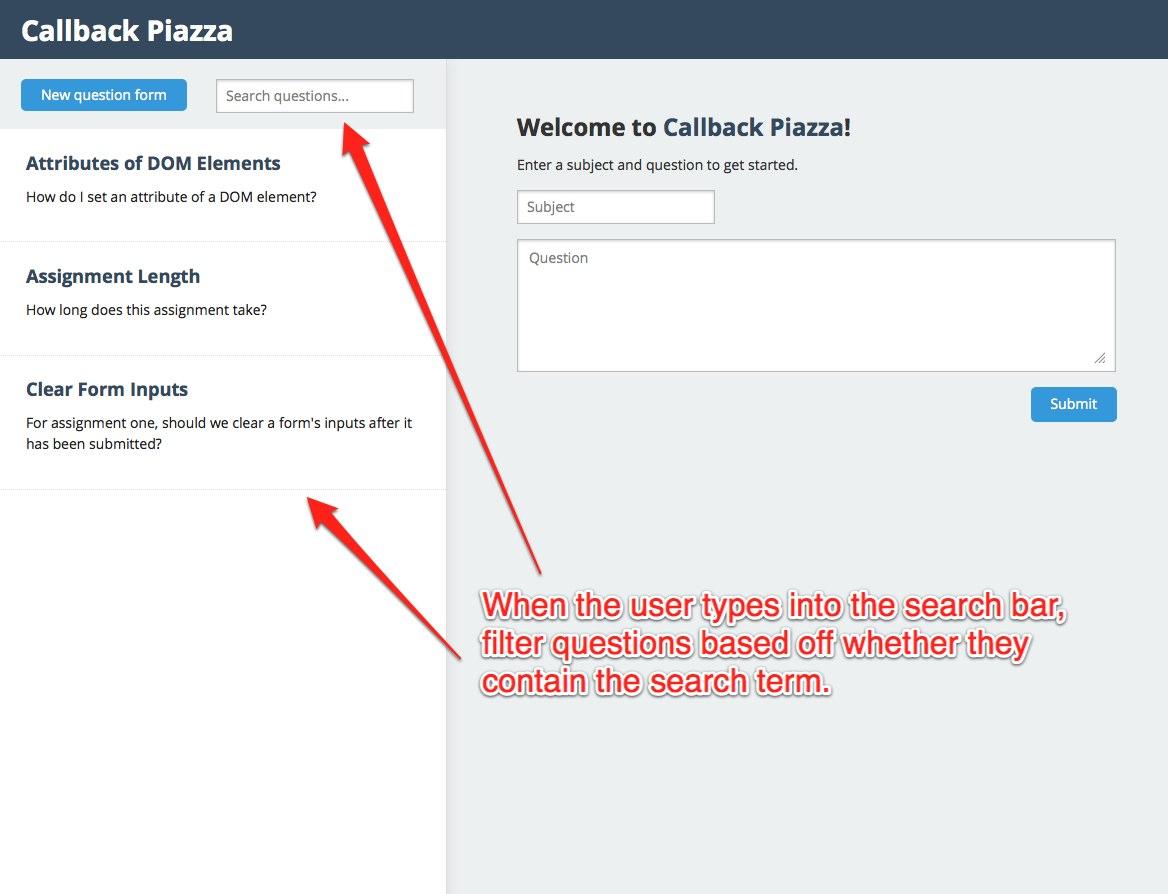
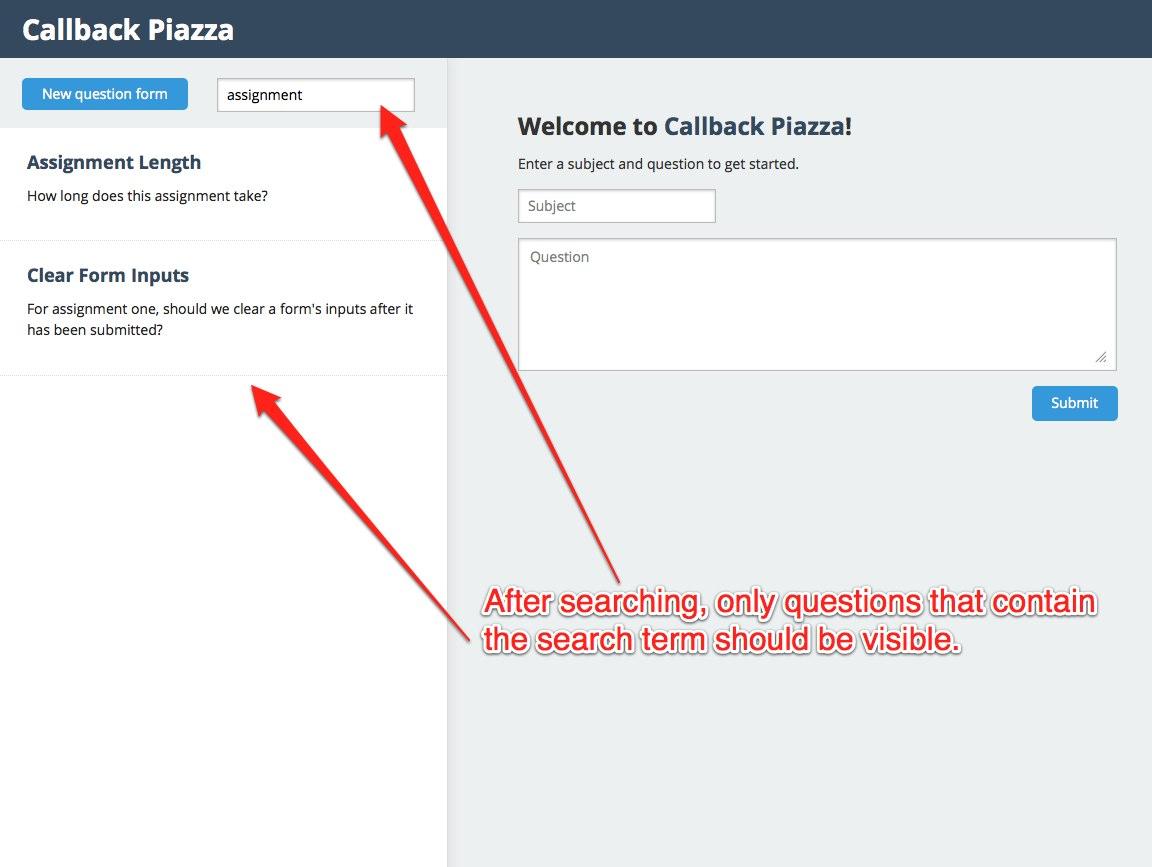
Keep in mind that all of these features are dynamically implemented in JavaScript. There should be no redirecting throughout the user’s interaction with the application, even when panels change!

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 refreshed?

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.

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. Add the limit that question descriptions must have at most 500 characters and at least 1. 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. The best way to do this is to modify js/templates.js, adding tags to contain error messages.
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:  
   After search:  
     
     
   When the DOM [is loaded, attach a keyup event listener to the](http://www.w3schools.com/jsref/jsref_indexof.asp) 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-paneusing the renderQuestions function, passing in a 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 they have 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 renderQuestions function. 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.  
     
   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 templates we provided in js/templates.js.  
     
   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:**

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:**

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 125 lines of code (not including comments or blank lines). 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. 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>>).

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.

Submission:

See email with subject “CS42: Callback Piazza Assignment Submission Instructions” for details.