# ICA 14: Single Page Architecture

This ICA has 2 parts. Be sure to read all the way to the end. We have been learning about using GET and POST communication to interact with API endpoints. To do that we have used the $.getJSON() and $.post() jQuery methods.

Those commands have a syntax that follows the pattern shown below:

|  |
| --- |
| // here the API does not require any data in the query string.  $.getJSON( ***url*** , ***function(result){}*** );  // here the API does require data in the query string.  $.getJSON( ***url*** , ***the\_serialized\_data*** , ***function(result){}*** );  $.post( ***url*** , ***the\_serialized\_data*** , ***function(result){}*** ); |

Earlier in the semester, we also saw how we can use the .val() and .html() methods of jQuery to get data out of an input tag and put content into the inner HTML of a tag. These commands, and others like them, all follow the same pattern.

First, we identify the tag or tags that exist in the HTML portion of our code, and then we manipulate the selected tag(s) in some way. We can identify a tag by its id. In CSS and jQuery, the # is used to represent the id of a tag.

For example:

|  |
| --- |
| // in the html body we have:  <div id="firstname"></div>  // in the script tag we have:  $("#firstname").html("fred");  // the above line would put "fred" into the inner html of the div tag |

Let’s briefly review the jQuery methods we already know:

| **jQuery method** | **Notes / Purpose** |
| --- | --- |
| $.getJSON() | Initiate an HTTP GET request |
| $.post() | Initiate an HTTP POST request |
| .html() | Put data into the inner HTML of a tag. (Can also be used to read the inner html of a tag.) |
| .val() | Read the value of an input tag. (Can also be used to set the value of an input tag.) |
| .append() | Append data into the inner HTML of a tag. |
| .serialize() | Retrieve all the named input tags on a form. Use those names and values to create a string of URL Encoded data. |

In this in class activity, we will introduce the following jQuery methods.

|  |  |
| --- | --- |
| **jQuery method** | **Notes / Purpose** |
| .addClass() | Add a CSS style. CSS styles change the way a tag looks. |
| .removeClass() | Remove the CSS styles applied to the tag. |
| .show() | Show a tag. Make it visible. |
| .hide() | Hide a tag. Make it invisible. |

The addClass and removeClass methods allow us to change the visual appearance of HTML tags. Rather than write our own CSS classes, we will use CSS classes that are already written for us.

Bootstap gives us classes of btn and alert. The btn tag makes the HTML look like a button. The alert tag makes the HTML look like an alert. There are additional CSS classes that give the buttons and alerts extra styling. Those additional CSS classes are all named the same, consistent, way using the words: success, danger, warning, information, primary, and secondary.

You can see examples of those kinds of classes here:

<https://www.w3schools.com/bootstrap/bootstrap_alerts.asp>

<https://www.w3schools.com/bootstrap/bootstrap_buttons.asp>

The show and hide methods are important because they allow you to show and hide visual elements on the page. This allows us to create a single html page that changes and adapts with user input. We can combine multiple visual interfaces into one single html file, and show / hide them as needed (as opposed to creating multiple html files).

This technique is fundamental to what is known as a single page architecture (SPA) web application.

Before we explore SPA further, we will examine each of the new jQuery methods.

## Instructions – Part A

1. Download ica14\_puppy.zip and unzip ica14\_puppy into your workspace folder.
2. Your instructor will lead you through the process of adding content to each of the buttons.
3. Your instructor will lead you through the process of using “show” and “hide” to make the picture of the puppy interactive.
4. Upload your ica14\_puppy folder to misdemo, and make sure that you can determine the URL for your work. Use the folder, so you do not have to specify “index.html” as part of your URL.

For example: <https://misdemo.temple.edu/tuz54321/ica14_puppy>

1. You don’t need to turn in this URL, but you should keep it as a study aid for the final exam.

**Part B begins here**

In this second part of the ICA we will use the jQuery show and hide methods to create a simple application. This application will retrieve a dailyquote from an API and show it to a user. The user then has the option to share that quote with a friend via an SMS text message.

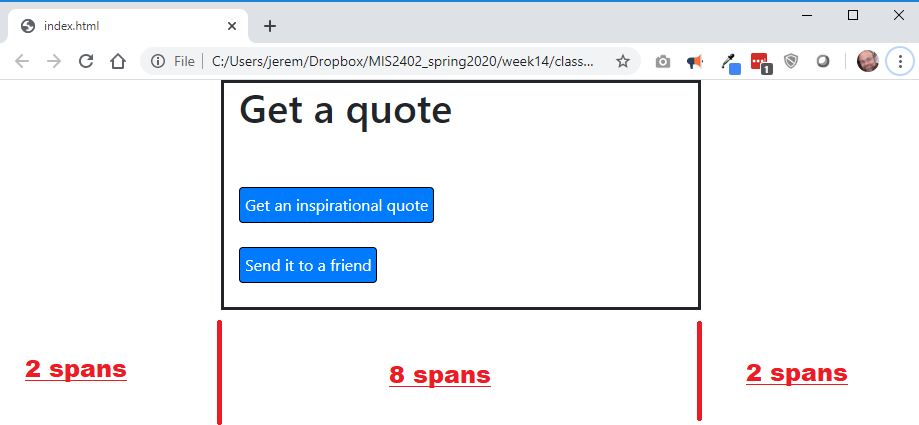
We don’t show the quote and the SMS options all at once. Instead, we have buttons that allow the user to navigate from one interface to the other.

Make note of the following URLs.

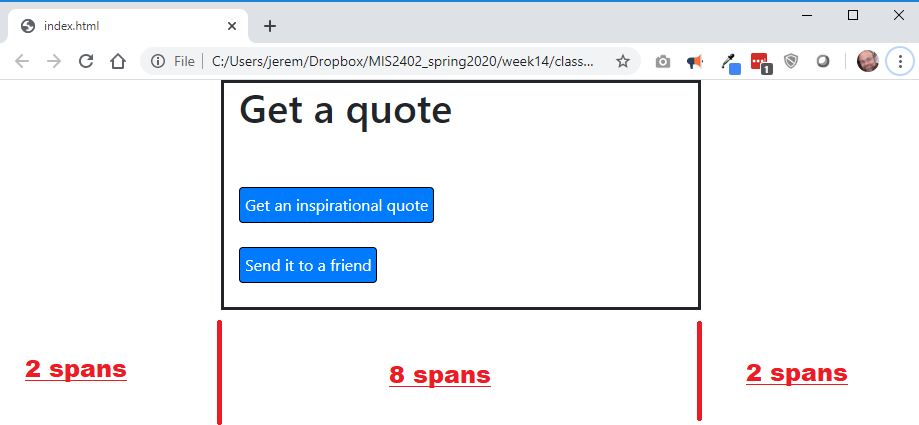
|  |  |
| --- | --- |
| **URL** | **Notes** |
| <https://misdemo.temple.edu/dailyquote> | GET from this URL to get an JSON response containing a fun and/or inspirational quote. Each GET requests results in a new/different quote. |
| <https://misdemo.temple.edu/textbelt> | POST to this URL to send a text message. You must pass “misdemousername”, “misdemopassword” and “message” and “phone” as URL encoded data. This API feature returns confirmation data in a JSON array. |

## Instructions – Part B

1. Start with ica14\_quotes.zip. Set up your project in VS code.
2. Notice that the HTML portion of this page has a Bootstrap container div, a Bootstrap row div, and several Bootstrap column div tags. Specifically, there is:
   1. A 2 span column on the left
   2. A 2-span column on the right
   3. In between the left and right columns there are two 8-span div tags identified by “div-quote” and “div-sms”. For the sake of clarity, a CSS style has been added to these div tags to give them a visible border.
3. Down in the <script> tag there are hide commands that are commented out. Uncomment them and see what happens.
4. Comment and/or uncomment the hide commands, and experiment until you see an image like what is show below:



1. Note that the width of each portion of the page is measured in bootstrap “spans”. The full width of the page equals 12 “spans”. The red text added to the following screenshot illustrates this.



1. By using hide and show you can control which of the 8-span div tags is displayed in response to user driven events (like button clicks.)

In the <script> tag, edit the click event handlers for # btn\_show\_form and # btn\_show\_quote to show / hide the 8-span div tags. This gives the illusion of moving from “page” to “page”. But really, everything is in a **single** page, index.html. That’s why this is called a **Single Page Architecture**.

Before you move on to the next step, satisfy yourself that this is working. Test it!

1. Now complete the supporting function get\_the\_quote(). This function should make an GET request, using getJSON, against <https://misdemo.temple.edu/dailyquote>

* Be sure to console.log the data you get back from the API.
* Looking at the data you get back, you should be able to put the text of the quote into ***two*** places.
  + The div tag “quote”
  + The input tag “message”

1. Now complete the supporting function send\_the\_text(). This function should serialize the form that holds the text message tags, and send it to the textbelt web service.
2. Test your work.
3. As before, upload your ica14\_quotes folder to misdemo, and make sure that you can determine the URL for your work. Use the folder, so you do not have to specify “index.html” as part of your URL.

For example: https://misdemo.temple.edu/tuz54321/ica14\_quotes

1. On canvas, turn in the URL to ica14\_quotes.