**WebTraining3: Javascript Exercise**

Now we will add some functionality to our app.

**Tweeting**

After clicking the tweet publish button, the text box should be erased of it's content, and a new DOM element with the text should be appended to the form a list of your tweets.

Until we'll have a server side, we'll load some data from a local JSON array for our tweets list, instead of the static HTML code, for example:

[

{username: 'Bobo', text: 'hello followers!'},

{username: 'Elvis', text: 'this exercise is really easy!'},

{username: 'Mimi', text: 'I want to go to sleep'}

]

Use this data for the list of tweets of the people you follow on the main page. Make sure that your new tweet is pushed to the JSON array. You may hardcode your own made-up username.

**Users page**

Implement the functionality of the 'Follow' button:  
The button has 2 states: 'Follow' and 'Unfollow'. Clicking the button toggles between these 2 states, and changes the color of the button too.

Also, add a list of people you follow (in the users page). So when you toggle between states, their avatar and name is added / removed from the list.

Save the users list and the folowees list in a local JSON array, similar to the tweeting functionality.

The name filter text box should actually filter the users list while typing. That means that if you begin typing in the box, and you wrote 'El' - only users with names containing 'El' should be visible on the users list (like Elvis or Elai). When you add 'v' (resulting in 'Elv') only users with names containing 'Elv' are visible in the users list (like Elvis. Elai is filtered out).

**OfekTestFramework**

In the very near future, we will want to change our code (add functionality, refactor it or incorporate a new framework that will simplify our code). But we also want to keep our functionality intact. In other words, we want to **change our code**, but make it **work all the same**.

This process is far less scary if we have something that validates our code and check that it behaves as it should. This is called **testing,** and it's a **big deal**.

So we want to test our Javascript code. Many frameworks exist exactly for this purpose (Mocha, Jasmine, Karma etc), but since we are in training and want to get our hands dirty, we will implement our own mini testing framework, called OfekTestFramework.

Testing should be available through the following API:

assert(value, name) -

tests the value and outputs the result on the page.

 value - a boolean value that determins the test result (true -

         passed, false - failed)

 name - the name of the test

test\_group(name, test\_group\_function) -

runs a multiple test function and outputs on the page whether the  test group passed. If any of the asserts in test\_group\_function

fails - the whole test group fails.

 name - test group name

 test\_group\_function - a function that runs tests with assert()

Here is a simple test (this is actually the tests that run in the previous sample):

test\_group('first test group', function() {

assert(true, "simple successful test");

assert(true, "simple successful test 2");

assert(false, "simple unsuccessful test");

});

test\_group('second test group', function() {

assert(true, "simple successful test");

assert(true, "simple unsuccessful test 2");

assert(true, "simple unsuccessful test 3");

});

**Notes**

* The purpose of this exercise is to implement the 2 fore-mentioned functions and to make use of them for some trivial tests.
* Closures are helpful here.

**Testing our new functionality**

Now that we have a testing framework, it's time to use it.  
Write some tests for the tweeting functionality and the users page. Make sure that when buttons get clicked - the appropriate data is added to the data array.

**OfekQuery**

So far, we've added some Javascript functionality, wrote a mini testing framework and used it to test our functionality.

Now we're going to add some more functionally, but we've seen that manipulating the DOM directly is not very pretty and hard to maintain. So before we add new features, let's write a small library (called "OfekQuery") that will aid us in making complex queries to the DOM and manipulating it.

Ultimately, the user of this library should be able to run these commands:

$('div') // returns an OfekQuery object of all of the div elements in the page

$('div .fancy') // returns an OfekQuery object of all div elements that have descendant elements of class "fancy"

$('#baba-man') // returns an OfekQuery object of the element with id "baba-man"

$('.pretty-box').addClass('big-box') // adds the "big-box" class to all of the elements of class "pretty-box"

$('ul').filter(function(el) {

   return (el.childElementCount === 2);

}) // returns a new OfekQuery object containing all ul elements that have exactly 2 children elements

$('ul').any(function(el) {

 return el.childElementCount > 5

}) // returns true if any ul elements have more than 5 children elements

**LIBRARY SKELETON**

First off, we will build the library skeleton, with no actual functionality.

Our API is a single constructor, named '$', that creates a new OfekQuery object.

Internal utility functions for the library (static private functions) should be seperate from instance methods. Recall prototypes (for instance methods) and immediate functions (for private functions and modularity).

**OFEKQUERY CORE FUNCTIONALITY**

The OfekQuery constructor will take a single string parameter. This parameter is the query to be run. The constructor will then run the query on the document (explained soon) and store the result elements in an encapsulated field (a private field of the object).

As seen in the example above, the queries are **very similar to CSS selectors**. The query is a single space delimited token. Each token can be one of three things: element type, element id or a class name.

* If the token starts with a hash (e.g. #baba-bubu, #test-1) then it is an id.
* If the token starts with a dot (e.g. .fancy-box, .passed) then it is a class name.
* Any other case is an element type (e.g. div, ul, input).

If there is just one token in the query - just look for any suitable elements in the whole DOM tree.  
However, if there is more than one token - the searching is hierarchical:  
"a b c" should look for all a elements, then look for all b elements that are descendants of a elements, then look for all c elements that are descendants of b elements who are descendants of a elements.  
For example, for a DOM tree such as this:  
document  
\*a  
\*\*c id=1  
\*a  
\*\*b  
\*\*c  
\*\*\*b  
**\*\*\*\*c id=2**  
\*a  
\*\*b  
**\*\*\*c id=3**  
**\*\*\*c id=4**  
\*\*\*\*b  
and the query above ("a b c"), only c elements with ids 2, 3 and 4 should be returned.

In case an empty value (non-string, null, undefined or empty string) is recieved - just return from the constructor.

**INSTANCE METHODS OF OFEKQUERY**

Now we will add some neat functionality to OfekQuery (these should be implemented as instance methods):

* *addClass(class\_name) -*adds a CSS class to the elements held by the object*.*
* *removeClass(class\_name) -*removes a CSS class to the elements held by the object.
* *each(fn)* - invokes function fn on each of the elements held by the object.
* *map(fn)* - appends to a new array the result of applying the function fn on that element. Should return a new array of the modified changes without changing the original array.  
  for example:

<span class="exmapleClass">Hello</span>

<span class="exmapleClass">My</span>

<span class="exmapleClass">This is the best</span>

var newArray = $(".exampleClass").map(function(element) { return element.innerHTML + " world" });

newArray ==> ["Hello world", "My world", "This is the best world"]

* *any(fn1, fn2, ...)* - returns true if any of the elements held by the object satisfies all passed functions, false otherwise.
* *all(fn1, fn2, ...)* - returns true if all of the elements held by the object satisfies all passed functions, false otherwise.
* *filter(fn1, fn2, ...)* - returns a new OfekQuery object with the elements that satisfied all passed functions.
* *css(property, value)* - sets the css on all elements held by the object.
* *count()* - returns the number of elements the object holds.
* *appendChild(childElement)* - appends a child element to all elements held by the object.
* *getAttribute(attributeName)* - get an array of attributes of the elements held by the object.
* *setAttribute(attributeName, attributeValue)* - set the value of an attribute of all elements held by the object.
* *get(index)* - returns the HTML element at the index from the elements array of the object.

All instance methods except addClass and removeClass are of a functional nature.

OfekTwitter with OfekQuery

Add the relevant files to your OfekTwitter project, include them in OfekTwitter's pages (index and users) and use OfekQuery to replace your direct DOM manipulation code.

[Here](http://javascript-ofek.bitballoon.com/) is the final result.