School of Computer Science

COMP5347: Web Application Development
Week 9 Tutorial: jQuery, AJAX and Google Charting

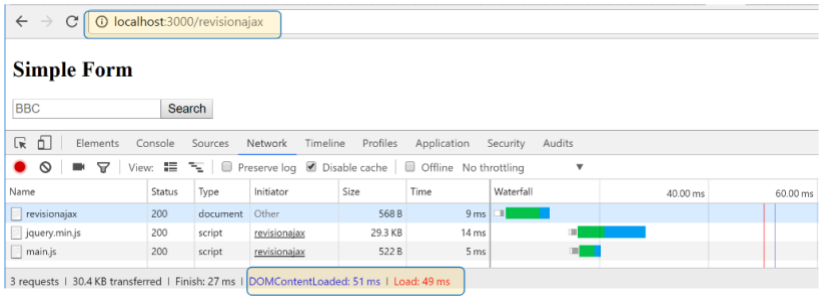
Learning Objectives

- Get familiar with jQuery selectors and syntax
- Practice Ajax requests
- Practice basic Google Charting features

Part 1: jQuery syntax and AJAX Requests

Download `week9-src.zip` from Canvas, Start Eclipse, select "open projects from file system" under file menu, go to the directory you just extracted the content and click open. In this case, all the necessary node modules are specified in `package.json`. Right click it on the project explorer panel, select "`Run as`" then "`npm install`".

You can run `revserver.js` as node.js application and you open this address with your browser: `http://localhost:3000/revisionajax`. Open the Developer Tools or Inspector from your browser and refresh the page. You should see information similar to this figure:



You can check `revision.server.routes.js` and see that path `/revisionajax` will load `titleFormAjax.pug`. If you check this pug file, it loads `main.js` file inside `js/` directory and jQuery library. Beside loading `main.js` file, this file also has a button with id "`button`" and a div with id "`results`". You will need to check the `main.js` file and try to understand what will this js file do with these two elements. When the button is clicked, it will try to find the value of "`title`" input and use it as a parameter for requesting `/getLatest` page. The results of `/getLatest` request then will be displayed inside the div with id `result`. This way, the result of the requested title will be shown in the same page of the form.

titleFormAjax.pug

```
doctype html

html(lang="en")
head
  title Ajax Search Example
    script(src="https://code.jquery.com/jquery-3.2.1.js")
    script(src="/js/main.js")
body
  h2 Simple Form
  input#title(type="search", placeholder="BBC")
  button#button(type='button') Search
  div#results
```

```
$(document).ready(function(){
  $('#button').on('click', function(e){
      var parameters = {title: $('#title').val() };
     $.get( 'revisionajax/getLatest',parameters, function(result) {
      $('#results').html(result);
    });
  });
});
```

Another way of registering event handler

Get the value of this input item, and construct parameter data based on it

Send a get request with data

Put the result as the content of this element

Part 2: Simple Charting Application

While running the server, access a new page from browser with this address: `http://localhost:3000/gchart`. This would display a simple page with two links. If you click the first link "Pie Chart", a pie chart would display. If you click the second link, you will get an error message.

Open `revision.server.routes.js` in Eclipse and inspect the content. The router defines two routes with path `/gchart` and `/gchart/getData`. The first route renders the view `graphs.pug`. The second route use a controller which demonstrates a slightly different response handling method. It does not send a text or render any view, instead it sends back a hard coded json object using `res.json()` function. The json object response is meant for client side script. But you can view it from browser by sending a request to `localhost:3000/gchart/getData` from your browser.

ExpressJS has predefined functions to configure the response in various ways, for a full list see reference document https://expressjs.com/en/api.html#res.

Now open `graphs.pug` and inspect its content in Eclipse. The `body` element of `graphs.pug` contains two anchor links. Both links are meant to be processed by client side javascript code. The `head` element of `graphs.pug` contains links to three scripts. The first two are third party libraries: jQuery and Google Charts loader. The last one is our own script: `graphs.js`

Majority of the processing in this graph page happens on client side and is implemented in `graphs.js`. Charting related functions and statements in this script are adapted from Google Charts Tutorials.

(https://developers.google.com/chart/interactive/docs/quick_start)

The first statement in this script calls the `load()` function to load the `corechart` package. This statement is required for any web page using Google Charts. Two global variables: `options` and `data` are declared in the next two statements. A global function `drawPie()` is declared to draw a pie chart based on the chart data stored in `data` variable. The chart data is stored as javascript object, the next few statements follow google Charts tutorial to prepare the `DataTable`. jQuery's `$.each()` function is used to iterate all key value pairs in the javascript object. After `DataTable` is prepared, a pie chart is created on the designed.

(https://developers.google.com/chart/interactive/docs/basic_load_libs)

Container element: `$("#myChart")[0]` is equivalent to the raw JavaScript expression `document.getElementById("myChart")`. The `chart.draw()` statement will update the element with the chart content.

The next statement registers an even handler on the `document.ready` event:

```
$(document).ready(function() {
...
}
```

Inside the event handler, an ajax request `$.getJSON(url, data, callback)` is sent to get the chart data from server. This request is similar to `$.get()` and `$.post()` except that it receives JSON data as response. The response is assigned to the global variable `data`, so that it is available to all scripts on this page.

It also registers an event handler to the click event of the link "Pie Chart". The event handler disables the default action of the link and calls the `drawPie()` function.

Part 3: Write your own code

Now write your own code to display a bar charts when a user clicks the “Bar Chart” link. You can find tutorial on bar chart below. Note that Google Charts uses different terminologies for vertical and horizontal bar charts.

https://developers.google.com/chart/interactive/docs/gallery/columnchart