# AJAX and REST in jQuery Recap

Lesson 5

# Plan for this lesson

- Ajax
  - Code along - JavaScript and Ajax
- Code along - jQuery and Ajax
  - jQuery and Ajax
- jQuery and Ajax load()
  - Code along - jQuery and Ajax (load)
- JSON
- jQuery and JSON
  - Code along - jQuery and JSON
- Rest
- Repetition
  - Introduction to JavaScript
  - JavaScript Data Types
  - Variables
  - Objects
  - The DOM
  - jQuery example, $ vs jQuery
  - jQueryUI Library
  - jQueryUI Library Datepicker
  - jQueryUI Library Selectable
- S

# AJAX

# Ajax New data without reloading

AJAX = Asynchronous JavaScript and XML.

AJAX is about loading data in the background and display it on the webpage, without reloading the whole page. The goal is thus to create an asynchronous Web application.

In practice, modern implementations commonly substitute JSON for XML due to the advantages of being native to JavaScript.

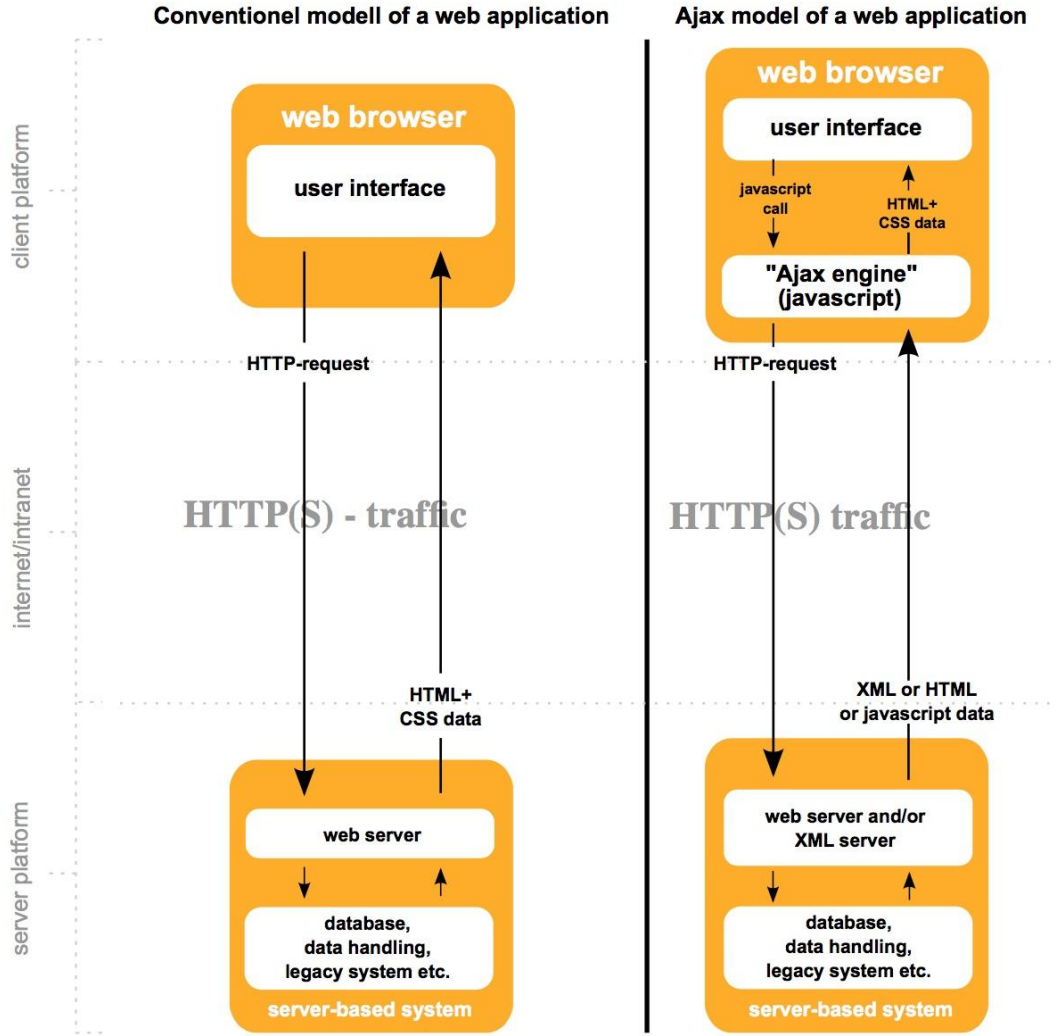
Examples of applications using AJAX: Gmail, Google Maps, Youtube, and Facebook tabs.

# Ajax New data without reloading

Ajax is not a technology, but a group of technologies. HTML and CSS can be used in combination to markup and style information. The DOM is accessed with JavaScript to dynamically display – and allow the user to interact with – the information presented. JavaScript and the XMLHttpRequest object provide a method for exchanging data asynchronously between browser and server to avoid full page reloads.

# Ajax New data without reloading

The conventional model for a Web Application versus an application using Ajax



# Ajax New data without reloading

Two commonly used methods for a request-response between a client and server are: GET and POST.

**GET** - Requests data from a specified resource

**POST** - Submits data to be processed to a specified resource

GET is basically used for just getting (retrieving) some data from the server.
Note: The GET method may return cached data.

POST can also be used to get some data from the server. However, the POST method NEVER caches data, and is often used to send data along with the request.

# Code along

JavaScript and Ajax

## Code from the code along HTML

```html
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>Example AJAX call using JavaScript</title>
    <script src="get-ajax-data.js"></script>
</head>
<body>
</body>
</html>
```

## Code from the code along PHP

```php
<?php
// This is the server-side script.

// Set the content type.
header('Content-Type: text/plain');

// Send the data back.
echo "This is the returned text.";
```

Since we can rely on the file stored on my server you do not need to upload this to your own webserver.

# Code from the code along JavaScript

```
1  // This is the client-side script.
2
3  // Initialize the HTTP request.
4  var xhr = new XMLHttpRequest();
5  xhr.open("get", "http://iul.dk/eal/send-ajax-data.php");
6
7  // Track the state changes of the request.
8  xhr.onreadystatechange = function () {
9      var DONE = 4; // readyState 4 means the request is done.
10     var OK = 200; // status 200 is a successful return.
11     if (xhr.readyState === DONE) {
12         if (xhr.status === OK) {
13             alert(xhr.responseText); // 'This is the returned text.'
14         } else {
15             alert('Error: ' + xhr.status); // An error occurred during the request.
16         }
17     }
18 };
19
20 // Send the request to send-ajax-data.php
21 xhr.send(null);
```

# Code along

jQuery and Ajax

## Code from the code along HTML

```html
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>Example AJAX call using JavaScript</title>
    <script src="https://code.jquery.com/jquery-3.1.1.min.js"></script>
    <script src="jq-get-ajax-data.js"></script>
  </head>
  <body>
  </body>
</html>
```

## Code from the code along jQuery

```javascript
$.get("http://iul.dk/eal/send-ajax-data.php")
    .done(function(data) {
        alert(data);
    })
    .fail(function(data) {
        alert('Error: ' + data);
    });
```

# jQuery and AJAX

# jQuery and Ajax New data without reloading

The jQuery library has a full suite of Ajax capabilities. The functions and methods therein allow us to load data from the server without a browser page refresh.

---

- $.ajax() Performs an async AJAX request
- $.ajaxPrefilter() Handle custom Ajax options or modify existing options before each request is sent and before they are processed by $.ajax()
- $.ajaxSetup() Sets the default values for future AJAX requests
- $.ajaxTransport() Creates an object that handles the actual transmission of Ajax data
- $.get() Loads data from a server using an AJAX HTTP GET request
- $.getJSON() Loads JSON-encoded data from a server using a HTTP GET request
- $.getScript() Loads (and executes) a JavaScript from a server using an AJAX HTTP GET request

# jQuery and Ajax New data without reloading

- $.param() Creates a serialized representation of an array or object (can be used as URL query string for AJAX requests)
- $.post() Loads data from a server using an AJAX HTTP POST request
- ajaxComplete() Specifies a function to run when the AJAX request completes
- ajaxError() Specifies a function to run when the AJAX request completes with an error
- ajaxSend() Specifies a function to run before the AJAX request is sent
- ajaxStart() Specifies a function to run when the first AJAX request begins
- ajaxStop() Specifies a function to run when all AJAX requests have completed
- ajaxSuccess() Specifies a function to run when an AJAX request completes successfully
- load() Loads data from a server and puts the returned data into the selected element
- serialize() Encodes a set of form elements as a string for submission
- serializeArray() Encodes a set of form elements as an array of names and values

# jQuery and Ajax load()

The optional callback parameter to **load()** specifies a callback function to run when the load() method is completed. The callback function can have different parameters:

- responseTxt - contains the resulting content if the call succeeds
- statusTxt - contains the status of the call
- xhr - contains the XMLHttpRequest object

The following example displays an alert box after the load() method completes. If the load() method has succeeded, it displays "External content loaded successfully!", and if it fails it displays an error message:

# jQuery and Ajax load()

Before we used regular **get ()** (http://www.w3schools.com/jquery/ajax_get.asp), but we can also use the jQuery **load()** method (http://www.w3schools.com/jquery/jquery_ajax_load.asp), which is a simple, but powerful AJAX method.

The load() method loads data from a server and puts the returned data into the selected element.

# Code along

jQuery and Ajax - load()

# Code from the code along HTML

```
<!DOCTYPE html>
<html>
<head>
    <script src="https://code.jquery.com/jquery-3.1.1.min.js"></script>
    <script src="ajax.js"></script>
</head>
<body>
    <div id="div1">
        <h2>Let jQuery AJAX Change This Text</h2>
    </div>
    <button>Get External Content</button>
</body>
</html>
```

# Code from the code along jQuery

```
$(document).ready(function(){
    $("button").click(function(){
        $("#div1").load("http://iul.dk/eal/send-ajax-data.php", function(responseTxt, statusTxt, xhr){
            if(statusTxt == "success")
                alert("External content loaded successfully!");
            if(statusTxt == "error")
                alert("Error: " + xhr.status + ": " + xhr.statusText);
        });
    });
});
```

# JSON

# JSON stored data

JSON (JavaScript Object Notation) is an open-standard format that uses human-readable text to transmit data objects consisting of attribute–value pairs. It is the most common data format used for asynchronous browser/server communication, largely replacing XML which is used by AJAX.

JSON is a language-independent data format. It derives from JavaScript, but as of 2016, code to generate and parse JSON-format data is available in many programming languages. The official Internet media type for JSON is application/json. The JSON filename extension is .json.

```
{
  "firstName": "John",
  "lastName": "Smith",
  "isAlive": true,
  "age": 25,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021-3100"
  },
  "phoneNumbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "office",
      "number": "646 555-4567"
    },
    {
      "type": "mobile",
      "number": "123 456-7890"
    }
  ],
  "children": [],
  "spouse": null
}
```

# jQuery and JSON

# jQuery and JSON getJSON()

There would be a situation when server would return JSON string against your request. The JQuery utility function **getJSON()** parses the returned JSON string and makes the resulting string available to the callback function as first parameter to take further action.

# Code along

jQuery and Ajax - getJSON()

```
<html>
   <head>
      <title>The jQuery Example</title>
    <script src="https://code.jquery.com/jquery-3.1.1.min.js"></script>
      <script type = "text/javascript" language = "javascript">
         $(document).ready(function() {
            $("#driver").click(function(event){
               $.getJSON('demo.json', function(jd) {
                  $('#stage').html('<p> First name: ' + jd.firstName+ '</p>');
                  $('#stage').append('<p>Last Name: ' + jd.lastName+ '</p>');
                  $('#stage').append('<p>Alive : ' + jd.isAlive+ '</p>');
                  $('#stage').append('<p>Age : ' + jd.age+ '</p>');
                  $('#stage').append('<p>Address : ' + jd.address.streetAddress + ', ' +
                    jd.address.city+ ', ' + jd.address.state +'</p>');
                  $('#stage').append('<p>PhoneNumbers : ' +
                    jd.phoneNumbers[0].type + ', ' + jd.phoneNumbers[0].number + ' - ' +
                    jd.phoneNumbers[1].type + ', ' + jd.phoneNumbers[1].number + ' - ' +
                    jd.phoneNumbers[2].type + ', ' + jd.phoneNumbers[2].number + '</p>');
                  $('#stage').append('<p>Children : ' + jd.children+ '</p>');
                  $('#stage').append('<p>Spouse : ' + jd.spouse+ '</p>');
               });
            });
         });
      </script>
   </head>
   <body>
      <p>Click on the button to load result.json file</p>
      <div id = "stage" style = "background-color:#eee;">
         STAGE
      </div>
      <input type = "button" id = "driver" value = "Load Data" />
   </body>
</html>
```

# Rest

# Rest Representational State Transfer

**Representational** state transfer (**REST**) or **RESTful** is an architectural style used for web development. Systems and sites designed using this style aim for fast performance, reliability and the ability to scale (to grow and easily support extra users). To achieve these goals, developers work with reusable components that can be managed and updated without affecting the system as a whole while it is running.

To the extent that systems conform to the constraints of REST they can be called RESTful. RESTful systems typically, but not always, communicate over Hypertext Transfer Protocol (HTTP) with the same HTTP verbs (GET, POST, PUT, DELETE, etc.) that web browsers use to retrieve web pages and to send data to remote servers.

# Rest Representational State Transfer

REST systems interface with external systems as web resources identified by Uniform Resource Identifiers (URIs), for example /people/johnjohn, which can be operated upon using standard verbs such as GET /people/johnjohn.

---

In many ways, AJAX applications follow the REST design principles. Each XMLHttpRequest can be viewed as a REST service request, sent using GET.

# Repetition

# Introduction to JavaScript Where to put JS?

You can put your JS three places in your web documents

1. Inline

   ```
   <a href="#" onclick="$(this).next().fadeIn(); return false;">Display my next sibling</a>
   ```

2. In the document header

   ```
   <script type="text/javascript">
           ...
   </script>
   ```

3. Linked in the header like you do external CSS

   ```
   <script type=”text/javascript” src=”external.js”></script>
   ```

# Introduction to JavaScript Code Along JS

```
43          function consoleFunction2() {
44              if (12 / 4 === "Erik".length) {
45                  console.log("Will it be the first block?");
46              } else {
47                  console.log("Or the second block?");
48              }
49              document.getElementById("demo").innerHTML =
50              "Check your console again!";
51          }
```

We check if the math “12/4” has exactly the same length as a string with my name and print the first string to the console if true and the second if false.
Finally we change the text to make sure the user checks the console to see if there was confirmation.

# JavaScript Data Types Repetition and detail

We already looked at data types in JavaScript last lesson, but we'll look a bit more at the concept in detail here.

There are six simple data types:

- Boolean
- Null
- Undefined
- Number
- String
- Symbol

And of course objects, which are far from simple and which we'll get back to.

# JavaScript Variables Repetition and detail

In JavaScript variables are containers for storing data values.

Creating a variable in JavaScript is called "declaring" a variable.

```
12    var x = 5;
```

After the declaration, the variable has no value unless set like in the example above. (Technically it has a value of undefined if not set)

To assign a value to the variable, use the equal sign. In JavaScript, the equal sign (=) is an "assignment" operator, not an "equal to" operator.

# JavaScript Objects Containers

You have already learned that JavaScript variables are containers for data values.

Objects are variables too. But objects can contain many values.

JavaScript objects are containers for **named** values.
The values are written as **name**:**value** pairs (name and value separated by a colon).

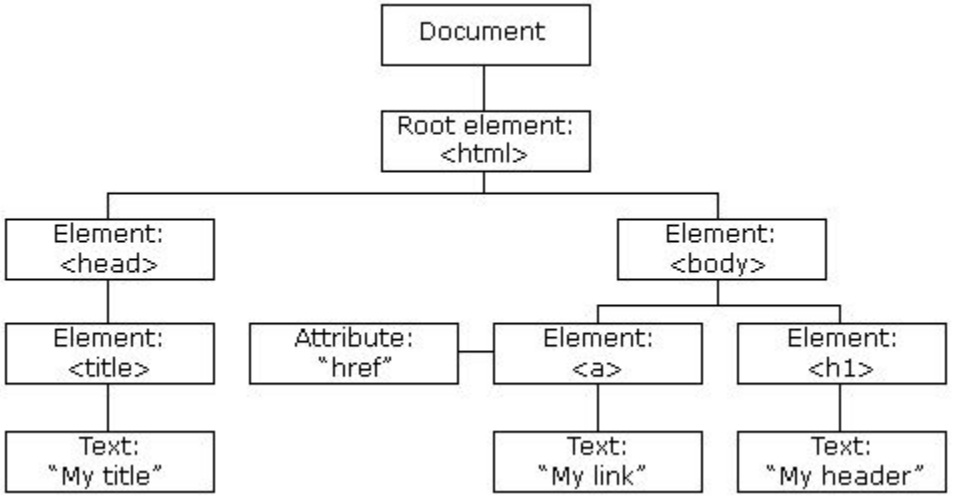
```
14    var car = {type:"Fiat", model:"500", color:"white"};
```

The name:values pairs (in JavaScript objects) are called **properties**.

# The DOM What is the DOM

When a web page is loaded, the browser creates a **D**ocument **O**bject **M**odel of the page.

The **HTML DOM** model is constructed as a tree of Objects:



# jQuery exampl $ vs jQuery

```
$(document).ready(function(){
    $("button").click(function(){
        $("p:first").addClass("intro");
    });
});
```

In JavaScript $ has no special significance (no more than a or Q anyway). It is just an uninformative variable name.
In jQuery the $ is an alias for the function called jQuery, so your code can be written like this with the exact same results: jQuery(document).ready(function() {...

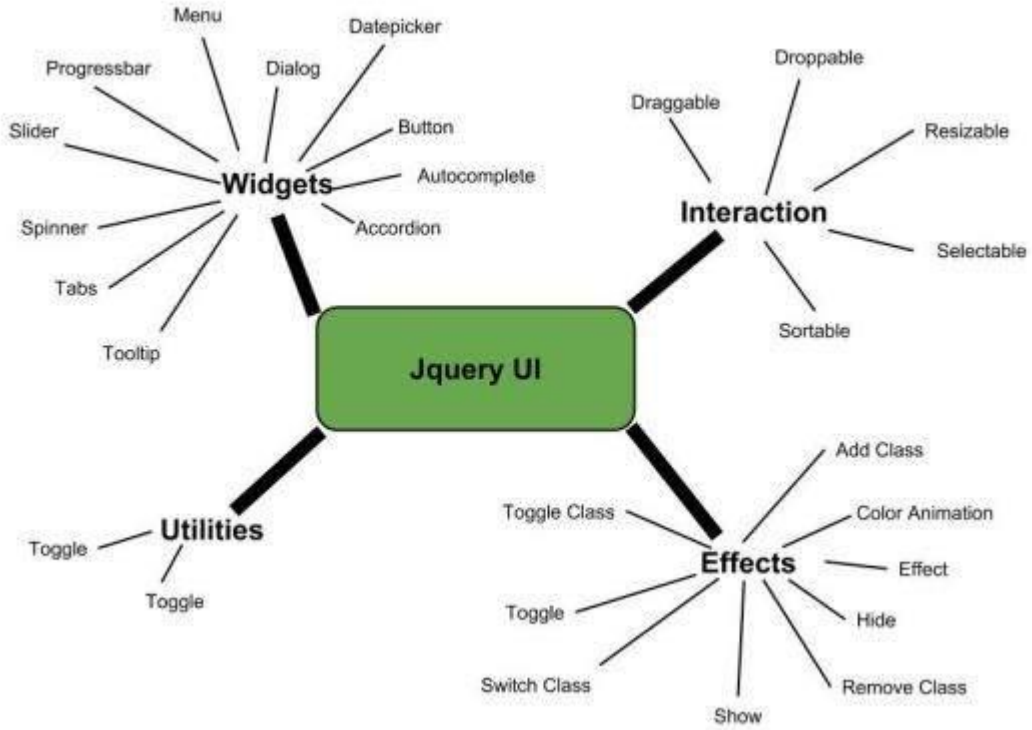
A page can't be manipulated safely until the document is "ready." jQuery detects this state of readiness for you. Code included inside $( document ).ready() will only run once the page Document Object Model (DOM) is ready for JavaScript code to execute.

addClass adds the specified class(es) to each element in the set of matched elements.

Using jQuery to change the color of the first p element by adding a CSS class to it.

We put it in a separate file so we do not need HTML tags.

# The jQueryUI library Adding to more visual effects to jQuery



# The jQueryUI library Datepicker

Datepickers in jQueryUI allow users to enter dates easily and visually. You can customize the date format and language, restrict the selectable date ranges and add in buttons and other navigation options easily.

jQueryUI provides a **datepicker()** method that creates a datepicker and changes the appearance of HTML elements on a page by adding new CSS classes. Transforms the <input>, , and  elements in the wrapped set into a datepicker control.

> By default, for <input> elements, the datepicker calendar opens in a small overlay when the associated text field gains focus. For an inline calendar, simply attach the datepicker to a , or  element.

See the full list of option at

# The jQueryUI library Selectable

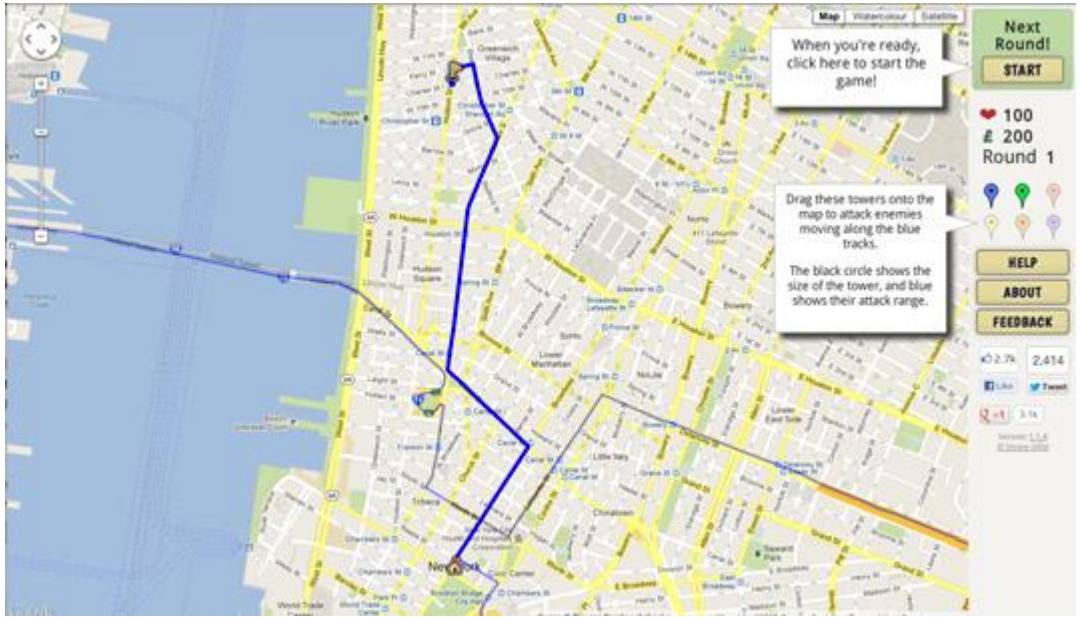
jQueryUI provides selectable() method to select DOM element individually or in a group. With this method elements can be selected by dragging a box (sometimes called a lasso) with the mouse over the elements. Also, elements can be selected by clicking or dragging while holding the Ctrl/Meta key, allowing for multiple (non-contiguous) selections.

Again there is options, six to be exact, but we will only use *selected*.

See the full list of options at
https://www.tutorialspoint.com/jqueryui/jqueryui_resizable.htm

# Examples of works using JavaScript

# MapsTD Game




MapsTD is a tower defence game. You tell it where your home is, and through Google Maps, it will produce a game in which you're defending your hometown.

It is built using the Google Maps API, with MooTools being used for the other aspects of the UI and as a general-purpose JavaScript library. It uses several bits of functionality provided by Google Maps. As well as the map itself, the biggest part is the route finder API, which is used to work out the paths the enemies follow.

# Multeor Game

Multeor is a multiplayer web game. The idea of the game is to control a meteor crashing into earth. You score points by ensuring you leave the biggest trail of destruction. Up to eight players can connect to a single game simultaneously.

Multeor is written in plain JavaScript using HTML5 Canvas and backed with a Node.js server to manage the communications between the desktop and mobile devices using WebSockets.

# Sources and further reading

- https://en.wikipedia.org/wiki/Ajax_(programming)
- http://api.jquery.com/category/ajax/
- http://www.w3schools.com/jquery/jquery_ajax_intro.asp
- http://www.w3schools.com/jquery/jquery_ajax_get_post.asp
- http://www.w3schools.com/jquery/jquery_ajax_load.asp
- https://en.wikipedia.org/wiki/JSON
- http://www.tutorialspoint.com/jquery/jquery-ajax.htm
- https://en.wikipedia.org/wiki/Representational_state_transfer
- https://code.tutsplus.com/tutorials/a-beginners-guide-to-http-and-rest--net-16340
- http://rest.elkstein.org/2008/02/ajax-and-rest.html
- http://www.creativebloq.com/web-design/examples-of-javascript-1233964