Web Technology and Standards

Dr. Alawami

Fall 2018

Agenda

- Client-Side scripting
  - Ajax & JSON
  - JQuery Crash Course
- Website design patterns
  - MVC
  - MVVM
- Front end framework
  - Angular Crash course
- Final Project Phase 1

Good Read

- https://www.bluecorona.com/blog/web-design-mistakes-to-avoid

JQuery

Caution:

DON’T USE FOR ASSIGNMENT 2

What is jQuery?

- JQuery is a fast and concise JavaScript Library that simplifies HTML document traversing, event handling, animating, and Ajax interactions for rapid web development. (jQuery.com)

Why learn jQuery?

- Write less, do more:
  - *$("p.neat").addClass("ohmy").show("slow");*
- Performance
- Plugins
- It’s standard
- … and fun!

Installing jQuery

- Download jQuery library from http://jquery.com/download/
- Make sure to download the compressed, production jQuery 2.**X** (right-click on the link and use the “Save As” option
- Save download *jquery-2.X.min.js* file to your project’s *scripts* folder.

Adding jQuery to an HTML File

- In your file’s header (between the <head></head> tags), add the following code:

```
<script language=“javascript” type=“text/javascript” src=“scripts/jquery-1.10.2.min.js”></script>
```

HTML ***script*** tag – contains programs written in supported scripting languages

language attribute – specifies which scripting language is used for programming. In this case it is JavaScript

type attribute is not required by HTML5, but older browsers need it

src attribute specifies location of your file. Note that this is a relative URL

How to use it?

- Two ways to access it:

-Download it and reference:

```
<script src="jquery.js"></script>
```

(in the same directory)

-Reference the online page:

```
<script src="http://code.jquery.com/jquery-1.4.2.min.js"></script>
```

jQuery is Used For:

- DOM traversal and manipulation
- Event handling
- AJAX calls

Aspects of the DOM and jQuery

- **Much of JQuery relies on the structure of the HTML to operate**

- **Identification:** how do I obtain a reference to the node that I want.
- **Traversal:** how do I move around the DOM tree.
- **Node Manipulation:** how do I get or set aspects of a DOM node.
- **Tree Manipulation:** how do I change the structure of the page.

window.onload

- We cannot use the DOM before the page has been constructed. jQuery gives us a more compatible way to do this.
  - The DOM way

```
window.onload = function() { // do stuff with the DOM }
```

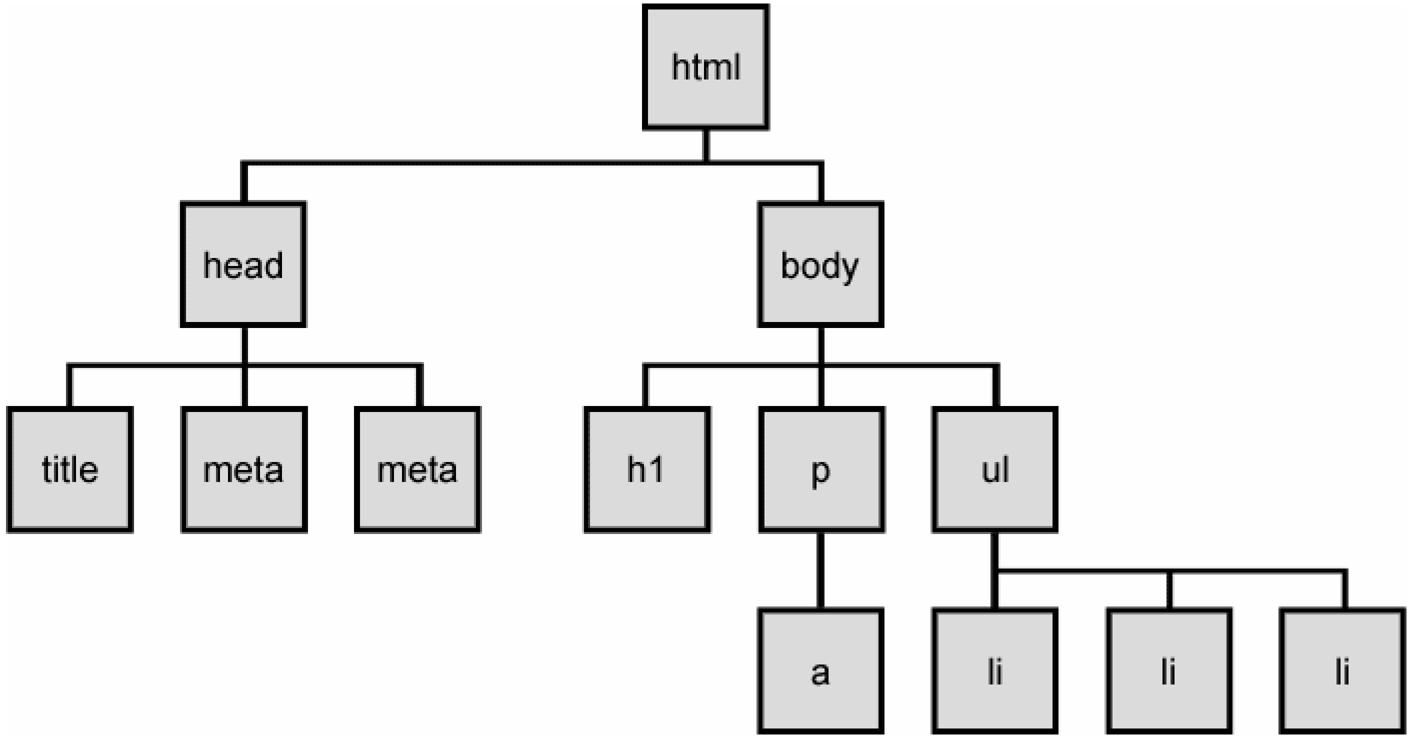
  - The direct jQuery translation

```
$(document).ready(function() { // do stuff with the DOM });
```

  - The jQuery way

```
$(function() { // do stuff with the DOM });
```

The DOM tree



Selecting groups of DOM objects

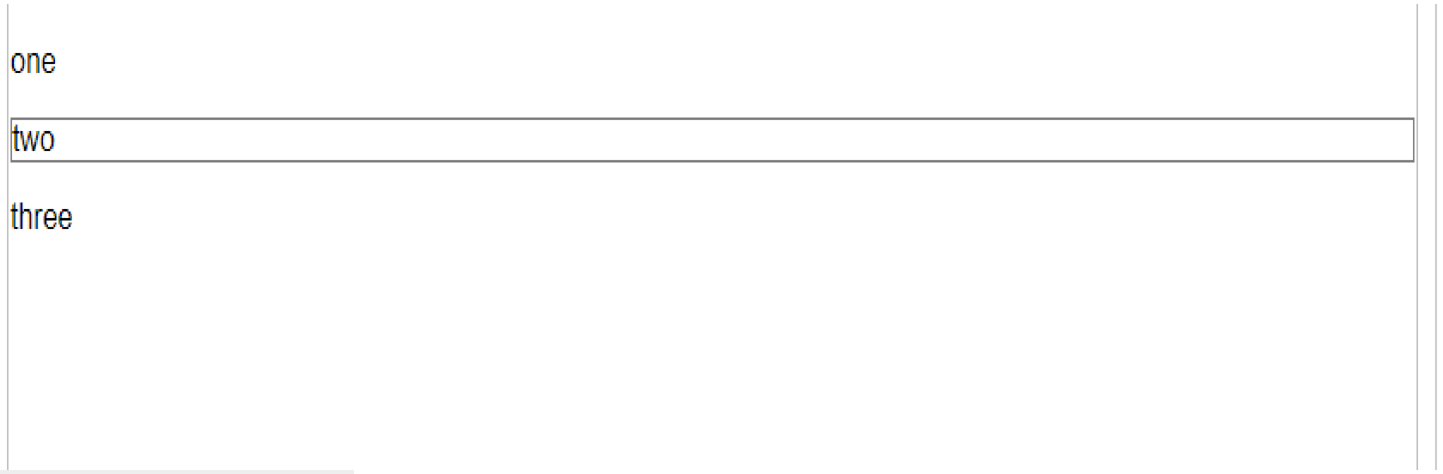
name	description
getElementById	returns array of descendents with the given tag, such as "div"
getElementsByTagName	returns array of descendents with the given tag, such as "div"
getElementsByName	returns array of descendents with the given name attribute (mostly useful for accessing form controls)
querySelector *	returns the first element that would be matched by the given CSS selector string
querySelectorAll *	returns an array of all elements that would be matched by the given CSS selector string

jQuery node identification

```
// id selector
var elem = $("#myid");

// element selector
Var elem= $("p");

// group selector
var elems = $("#myid, p");

// context selector
var elems = $("#myid < div p");

// complex selector
var elems = $("#myid < h1.special:not(.classy)");
```

JQuery
Statements
$(   )

Example


one

two

three

```
<!doctype html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>jQuery demo</title>
<script src="https://code.jquery.com/jquery-1.10.2.js"></script>
</head>
<body>
<p>one</p>
<div><p>two</p></div>
<p>three</p>
<script>
$( "div > p" ).css( "border", "1px solid gray" );
</script>
</body>
</html>
```

jQuery Selectors

- http://api.jquery.com/category/selectors/
- https://www.w3schools.com/jquery/default.asp

jQuery / DOM comparison

DOM method	jQuery equivalent
getElementById("id")	$("#id")
getElementsByTagName("tag")	$("tag")
getElementsByName("somename")	$("[name='somename']")
querySelector("selector")	$("selector")
querySelectorAll("selector")	$("selector")

jQuery traversal methods

- http://api.jquery.com/category/traversing/

Remember, it is just Javascript.

-It is nothing fancy, it is just a bunch of methods, objects, and variables (look at the document). We can use It just like we would any javascript:

```
<script src = jquery.js></script>
<script type = “text/javascript”>
var array = new Array();
var counter=0;

function onClick(id){
array[counter]=id;
counter+=1;
$(id).hide();//<is jquery
}

</script>
```

Example

```
$("p").click(function(){
    $(this).hide();
});
```

- More JQuery events

https://www.w3schools.com/jquery/jquery_events.asp

HTML5 Data Attributes

- The **data-*** attributes is used to store custom data private to the page or application.
- The **data-*** attributes gives us the ability to embed custom data attributes on all HTML elements.
- The stored (custom) data can then be used in the page's JavaScript to create a more engaging user experience (without any Ajax calls or server-side database queries).

http://www.w3schools.com/tags/att_global_data.asp

HTML5 Data Attributes

```html
<ul>
  <li data-animal-type="bird">Owl</li>
  <li data-animal-type="fish">Salmon</li>
  <li data-animal-type="spider">Tarantula</li>
</ul>
```

http://www.w3schools.com/tags/att_global_data.asp

HTML5 Data Attributes

The data-* attributes consist of two parts:

1. The attribute name should not contain any uppercase letters, and must be at least one character long after the prefix "data-"
2. The attribute value can be any string

http://www.w3schools.com/tags/att_global_data.asp

Data Attributes HTML Syntax

```
<article
  id="electriccars"
  data-columns="3"
  data-index-number="12314"
  data-parent="cars">
...
</article>
```

https://developer.mozilla.org/en-US/docs/Web/Guide/HTML/Using_data_attributes

Data Attributes JavaScript Access

```
var article = document.getElementById('electriccars');

article.dataset.columns // "3"

article.dataset.indexNumber // "12314"

article.dataset.parent // "cars"

var v1 = article.getAttribute("data-columns");

article.setAttribute("data-columns", 5);
```

https://developer.mozilla.org/en-US/docs/Web/Guide/HTML/Using_data_attributes

Data Attributes jQuery Access

```
var article = $('electriccars');


article.data(“columns”) // "3"
article.data(“indexNumber”) // "12314"
article.data(“parent”) // "cars"

article.data(“parent”, “trucks”) // Change value of data-parent
```

https://developer.mozilla.org/en-US/docs/Web/Guide/HTML/Using_data_attributes

JQuery Reference

- Guess who?
  - Click here

Ajax and JSON

The Hard way!

The usual way we operate in the Web

Ajax is every web developer dream ☺

- Typical browsing behaviour consists of loading a web page, then selecting some action that we want to do, filling out a form, submitting the information, etc.
- We work in this sequential manner, requesting one page at a time, and have to wait for the server to respond, loading a whole new web page before we continue.
- This is also one of the limitations of web pages, where transmitting information between a client and server generally requires a new page to be loaded.
- JavaScript is one way to cut down on (some of) the client-server response time, by using it to verify form (or other) information *before* it's submitted to a server.
- One of the limitations of JavaScript is (or used to be) that there was no way to communicate directly with a web server.
- Another drawback to this usual sequential access method is that there are many situations where you load a new page that shares lots of the same parts as the old (consider the case where you have a "menu bar" on the top or side of the page that doesn't change from page to page).

Things change…

- Until recently, we didn't have any alternative to this load/wait/respond method of web browsing.

- Ajax (sometimes written AJAX) is a means of using JavaScript to communicate with a web server without submitting a form or loading a new page.

- Ajax makes use of a built-in object, `XMLHttpRequest`, to perform this function.

- This object is not yet part of the DOM (Document Object Model) standard, but is supported (in different fashions) by Firefox, Internet Explorer, Safari, Opera, and other popular browsers.
- The term "Ajax" was coined in 2005, but the `XMLHttpRequest` object was first supported by Internet Explorer several years before this.

https://www.w3.org/TR/XMLHttpRequest/

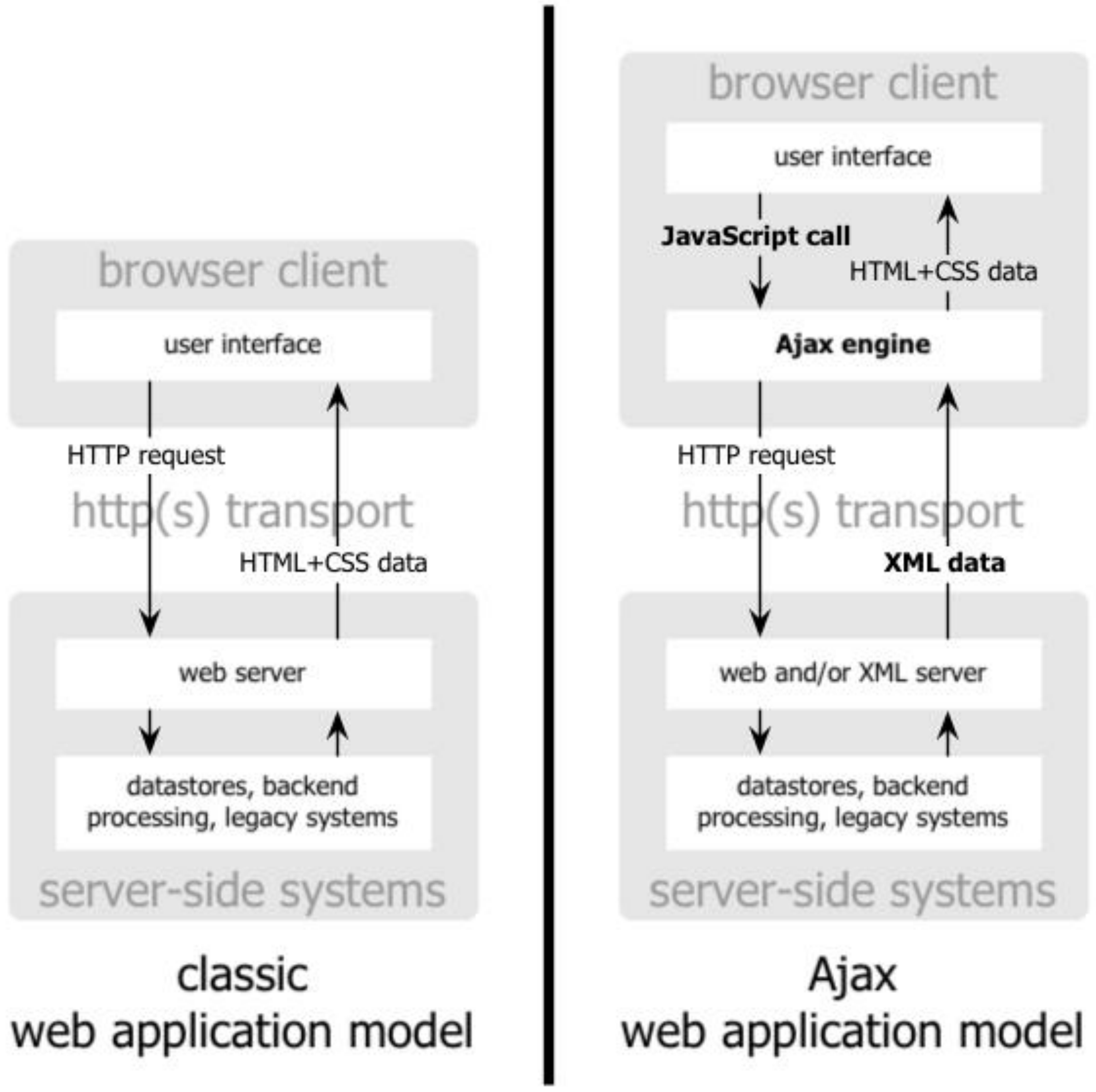
Ajax

- Ajax stands for “Asynchronous JavaScript and XML”.
- The word “asynchronous” means that the user isn’t left waiting for the server the respond to a request, but can continue using the web page.
- The typical method for using Ajax is the following:
  1) A JavaScript creates an `XMLHttpRequest` object, initializes it with relevant information as necessary, and sends it to the server. The script (or web page) can continue after sending it to the server.
  2) The server responds by sending the contents of a file or the output of a server side program (written, for example, in java, php etc).
  3) When the response arrives from the server, a JavaScript function is triggered to act on the data supplied by the server.
  4) This JavaScript response function typically refreshes the display using the DOM, avoiding the requirement to reload or refresh the entire page.

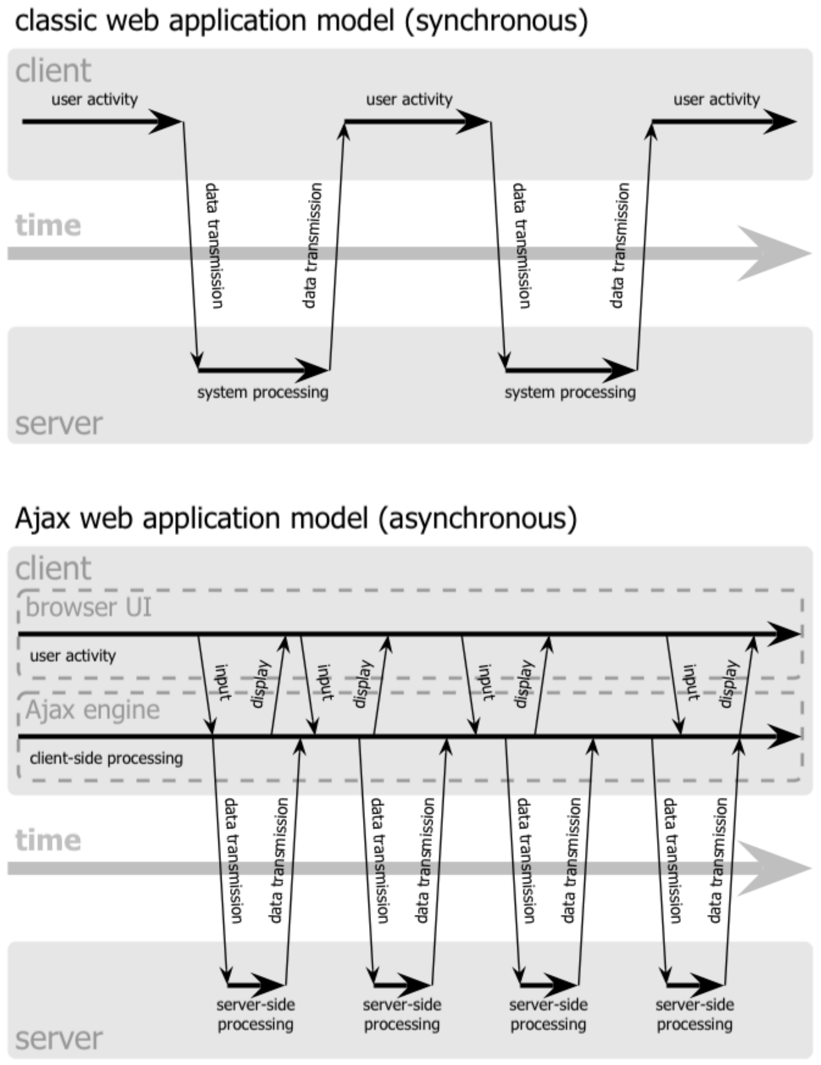
The Back End

- The part of the Ajax application that resides on the web server is referred to as the “back end”.
- This back end could be simply a file that the server passes back to the client, which is then displayed for the user.
- Alternatively, the back end could be a program, written in PHP, Perl, Ruby, Python, C, or some other language that performs an operation and sends results back to the client browser.
- An `XMLHttpRequest` object can send information using the GET and POST methods to the server in the same way that an HTML form sends information.
- Recall from our previous discussions that the GET request encodes the information inside of the URL, while a POST request sends its data separately (and can contain more information than a GET request can).

Classic Model νAJAX



Classic Model v Ajax



Writing an Ajax application

- We have to write the “front end” of the application in JavaScript to initiate the request.
- The back end, as mentioned, processes the request and sends it’s response back to the client. The back end is typically a short program we write for performing some dedicated task. This could be scripted in any language that is capable of sending back communication to the browser, like PHP or Perl.
- We also need to write the JavaScript response function for processing the response and displaying any results (or alterations to the web page).
- The “x” in Ajax stands for XML, the extensible markup language. XML looks like HTML, which is no mistake as the latest versions of HTML are built upon XML. The back end could send data back in XML format and the JavaScript response function can process it using built-in functions for working with XML. The back end could also send plain text, HTML, or even data in the JavaScript format.
- We will discuss some of these methods for sending data back to the requesting client and how it can be processed.

The XMLHttpRequest object

- The `XMLHttpRequest` object is the backbone of every Ajax method. Each application requires the creation of one of these objects. So how do we do it?
- As with most things in web programming, this depends upon the web browser that the client is using because of the different ways in which the object has been implemented in the browsers.
- Firefox, Safari, Opera, and some other browsers can create one of these objects simply using the “new” keyword.

```
<script type="text/javascript">
          ajaxRequest = new XMLHttpRequest();


</script>
```

The XMLHttpRequest object (cont.)

```
function getXMLHttpRequest()
/*   This function attempts to get an Ajax request object by trying
     a few different methods for different browsers.  */
{
   var request, err;
   try {
          request = new XMLHttpRequest();   // Firefox, Safari, Opera, etc.
       }
   catch(err) {
       try {             //  first attempt for Internet Explorer
          request = new ActiveXObject("MSXML2.XMLHttp.6.0");
           }
       catch (err) {
                     try {    //  second attempt for Internet Explorer
                     request = new ActiveXObject("MSXML2.XMLHttp.3.0");
                         }
                     catch (err) {
                          request = false;  // oops, can't create one!
                                 }
                  }             }
   return request;
}
```

If this function doesn't return "false" then we were successful in creating an `XMLHttpRequest` object.

The XMLHttpRequest object (cont.)

- As with any object in JavaScript (and other programming languages), the XMLHttpRequest object contains various properties and methods.

- The main idea is that the properties are set after the object is created to specify information to be sent to the server, as well as how to handle the response received from the server. Some properties will be updated to hold status information about whether the request finished successfully.

- The methods are used to send the request to the server, and to monitor the progress of the request as it is executed (and to determine if it was completed successfully).

XMLHttpRequest object properties

Property	Description
• `readyState`	An integer from 0. . .4. (0 means the call is uninitialized, 4 means that the call is complete.)
• `onreadystatechange`	Determines the function called when the objects readyState changes.
• `responseText`	Data returned from the server as a text string (read-only).
• `responseXML`	Data returned from the server as an XML document object (read-only).
• `status`	HTTP status code returned by the server
• `statusText`	HTTP status phrase returned by the server

We use the `readyState` to determine when the request has been completed, and then check the `status` to see if it executed without an error. (We'll see how to do this shortly.)

XMLHttpRequest object methods

Method	Description
• open('method', 'URL', asyn)	Specifies the HTTP method to be used (GET or POST as a string, the target URL, and whether or not the request should be handled asynchronously (asyn should be true or false, if omitted, true is assumed).
• send(content)	Sends the data for a POST request and starts the request, if GET is used you should call send(null).
• setRequestHeader('x','y')	Sets a parameter and value pair x=y and assigns it to the header to be sent with the request.
• getAllResponseHeaders()	Returns all headers as a string.
• getResponseHeader(x)	Returns header x as a string.
• abort()	Stops the current operation.

The `open` object method is used to set up the request, and the `send` method starts the request by sending it to the server (with data for the server if the POST method is used).

A general skeleton for an Ajax application

```
<script type="text/javascript">
       // *****  include the getXMLHttpRequest function defined before
var ajaxRequest = getXMLHttpRequest();

if (ajaxRequest) {   //  if the object was created successfully

      ajaxRequest.onreadystatechange = ajaxResponse;
      ajaxRequest.open("GET", "search.php?query=Bob");
      ajaxRequest.send(null);
   }



function ajaxResponse()  //This gets called when the readyState changes.
{
   if (ajaxRequest.readyState != 4)  //  check to see if we're done
      {  return;  }
   else {
     if (ajaxRequest.status == 200) //  check to see if successful
         {   //  process server data here. . . }
     else {
       alert("Request failed: " + ajaxRequest.statusText);
         }
     }
}
</script>
```

A first example

- Here's an example to illustrate the ideas we've mentioned (inspired by an example in the book Ajax in 10 Minutes by Phil Ballard).
- The main idea is that we're going to get the time on the server and display it to the screen (and provide a button for a user to update this time). The point I want to demonstrate here is how to use Ajax to do this update without updating/refreshing the entire webpage.

- We use a (very) small PHP script to get the date from the server, and return it as a string as a response to the request. Here is the script:

```
<?php
echo date('H:i:s');
?>
```

- I saved this as the file "telltime.php".
- The HTML file and JavaScript code follows.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml11/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">

<head>
<title>Ajax Demonstration</title>
<style>
body {
   background-color: #CCCCCC;
   text-align: center;
     }
.displaybox {
   margin: auto;
   width: 150px;
   background-color: #FFFFFF;
   border: 2px solid #000000;
   padding: 10px;
   font: 1.5em normal verdana, helvetica, arial, sans-serif;
            }
</style>

<script type="text/javascript">
var ajaxRequest;

function getXMLHttpRequest()
/*   This function attempts to get an Ajax request object by trying
     a few different methods for different browsers.  */
{
      //  same code as before. . .
}
```

```
function ajaxResponse()  //This gets called when the readyState changes.
{
   if (ajaxRequest.readyState != 4)  //  check to see if we're done
      {  return;  }
   else {
     if (ajaxRequest.status == 200) //  check to see if successful
         {
                   document.getElementById("showtime").innerHTML =
                                 ajaxRequest.responseText; }
     else {
       alert("Request failed: " + ajaxRequest.statusText);
         }
     }
}


function getServerTime()   //  The main JavaScript for calling the update.
{
   ajaxRequest = getXMLHttpRequest();
   if (!ajaxRequest)  {
         document.getElementById("showtime").innerHTML = "Request error!";
         return;      }
   var myURL = "telltime.php";
   var myRand = parseInt(Math.random()*999999999999999);
   myURL = myURL + "?rand=" + myRand;
   ajaxRequest.onreadystatechange = ajaxResponse;
   ajaxRequest.open("GET", myURL);
   ajaxRequest.send(null);
}
</script>
</head>
```

```
<body onload="getServerTime();">
<h1>Ajax Demonstration</h1>

<h2>Getting the server time without refreshing the page</h2>

<form>
   <input type="button" value="Get Server Time" onclick="getServerTime();" />
</form>
<div id="showtime" class="displaybox"></div>

</body>
</html>
```

The main functionality is handled by the `getServerTime()` function in setting up and sending the `XMLHttpRequest` object, and the `ajaxResponse()` function to display the time.

view the output page

Using a database for the live search

- We can also utilize ajax to get information out of mySQL database.
- The JavaScript need not be changed (except for the name of the script to call).
- A PHP script or cgi can obtain data from DB and get the result back as XML document
- Sample code will follow

```
<?php
header("Content-Type: text/xml");
echo "<?xml version=\"1.0\"?>\n";
echo "<names>\n";

if(!$query) $query = strtoupper($_GET['query']);

if($query != "")
  {
     include_once('db_access.php');
     $connection = mysql_connect($db_host, $db_username, $db_password);
     if(!$connection)
        {
           exit("Could not connect to the database: <br/>" . htmlspecialchars(mysql_error));
        }
     mysql_select_db($db_database);

     $select = "SELECT ";
     $column = "name ";
     $from = "FROM ";
     $tables = "people ";
     $where = "WHERE ";
     $condition = "upper(name) LIKE '%$query%'";

     $SQL_query = htmlentities($select . $column . $from . $tables . $where . $condition);
     $result = mysql_query($SQL_query);
     while ($row = mysql_fetch_row($result))
          {
           echo "<name>" . $row[0] . "</name>\n";
          }

     mysql_close($connection);
  }
echo "</names>";
?>
```

Some cautions

- As with any JavaScript element, you can't (or shouldn't) rely upon a user's browser being able to execute JavaScript (some people turn it off on their browsers). (Of course, there are webpages that ignore this caution.)
- **Debug carefully** and on many different browsers. Ajax uses features that might not be present in all browsers or they may not operate in the same fashion.
- If you can, indicate to the user that "something is happening" or that something has changed on the page, otherwise they may not notice it.
- Ajax can possibly introduce strange behaviour, like the "Back" button on the browser doesn't act like it did before (as with any dynamic website), or that if you use some "hidden" elements in your page (generated by Ajax), then they will likely not show up in a form that search engines will recognize.
- For the sake of us all (who will be looking at your pages), ***don't let "flashy behavior" be a substitute for actual content, or a well designed layout of your web pages.***

Asynchronous JavaScript and XML (AJAX)

- The key technology for AJAX was created by Microsoft in 1999 – a JavaScript object that could send a request to a server and handle a response.
- The name results from two things
    - Javascript is an object based scripting language that uses the XML DOM as the basis for operation
        - Asynchronous JavaScript refers to the use of JavaScript to *send a message and handle the response asynchronously* – whenever it comes back
        - XML was initially proposes for use in handling the data transferred and XML also provides the Document Object Model (DOM) which is manipulated in the browser as a directed acyclic graph – a tree structure

The AJAX Object

- an XMLHttpRequest object:

```
var AJAXobj = new XMLHttpRequest();
```

- The code fragment to create the AJAXObj normally looks something like this:

```
if (window.XMLHttpRequest)

  AJAXobj=new XMLHttpRequest();
      }
   else

  AJAXobj=new ActiveXObject("Microsoft.XMLHTTP");

  }
```

Request Methods

- Simple requests
  - A simple GET request:
    - requester.open("GET", /query.cgi?name=Bob&email=bob@example.com, true);
    - requester.send(null);
  - A simple POST request:
    - requester.open("POST", "/query.cgi“, true);
    - requester.send("name=Bob&email=bob@example.com");
- In reality, requests can be much more complex and may involve the construction of XML structures or JSON objects

Processing Responses

- Given an AJAXObj, the most direct way to handle a response is to define an anonymous function that is activated every time the “ready state” of the request changes:

```
AJAXObj.onreadystatechange = function() {
        if (self.xmlHttpReq.readyState == 4)
                { do something with AJAXObj.responseText; }
        }
```

- There are four ready states that relate to the initiation, processing, and completion of the request. “4” indicates that the response is completed.
- There are also a variety of response objects besides text

Putting It All Together
Main Function

- We define four separate functions. The main function controls functions to construct an AJAXObj, construct the message, and indicate how the response should be handled:

```
var AJAXObj
function makeAJAXRequest() {
AJAXObj = getAJAXObject();
AJAXObj.open('POST', ‘http://www.server/ServerProgram’, true);
Req.setRequestHeader('Content-Type', ‘text/xml');
Req.onreadystatechange = processResponse();
Req.send(getqueryString());
}
```

Putting It All Together Construction Function

- The construction function takes into account cross browser issues and may also note the browser type for other uses.

```
var AJAXObj; //Globally defined for the JavaScript
var browserClass;
function getAJAXObject(){
browserClass = navigator.userAgent;  // this is normally simplified
if (window.XMLHttpRequest)
      {// code for IE7+, Firefox, Chrome, Opera, Safari
      AJAXobj=new XMLHttpRequest(); }
    else
      {// code for IE6&E5
      AJAXobj=new ActiveXObject("Microsoft.XMLHTTP"); }
return AJAXobj;
}
```

Putting It All Together Request Construction

- Any number of different strategies might be used to construct messages of many different forms. This example constructs an XML fragment from a form – and assumes all form elements are simple text input elements. (The JavaScript escape function takes care of the URL encoding.)

```
function getqueryString(fname){
var myform = document.forms[fname];
qstr = “<form>”;
For (var i=0;i<form.elements.length;i++){
    qstr+ = “<“+form.element[i].name‘+”>”+
    escape(form.element[i].value)+
    “</”form.element[i].name+”>”;
    }
qstr+=“</form>”
return qstr;
}
```

Putting It All Together
Response Processor

- There are any number of forms responses can take and any number of ways that they can be handled. Here we assume it is an html fragment that can simply be appended to an object in the document prepared for it.

```
var AJAXObj

function processResponse(){
var message = AJAXObj.responsetxt;
var targetel = document.getElementById(“responsediv”);
targetel.innerHTML = message.
}
```

The Server Side

- AJAX is said to be *server agnostic* – it really doesn't matter what kind of code sits on the other side.
- For a GET, the parameters will be URL encoded and attached to the request:
  - http://some.server/someprogram?name=value
- For a post, the same query string, again URL encoded will be put in the message
- Any program acting on the server side will simply use its standard mechanism for accessing the parameters of a request as shown on the next slide

Your Second Best friend!

- Mozilla developer network

https://developer.mozilla.org/en-US/docs/Web/API

Access to AJAX Request in Java Servlets

```
protected void processRequest(HttpServletRequest req, HttpServletResponse resp)  throws ServletException,
IOException
                {
                resp.setContentType("text/plain;charset=UTF-8");
                PrintWriter out = resp.getWriter();
                try {
                String id = req.getParameter("id”);
                if (dbmscheck(id))
                                {out.println(“This user has been validated");}
                else
                                 {out.println(“This user is not validated");}
                } finally {
          out.close();
                }
}
```

Complications and Solutions

- One of the biggest issues in the use of AJAX is how data is transmitted.
  - When the data is simple, simple strings are more than satisfactory in requests
  - When both the client – the html page – and the server – e.g. a servlet – are prepared to do DOM processing, XML is a good choice.
  - If there is a lot of numeric data, and DOM processing isn’t useful or needed, the JavaScript Object Notation or JSON becomes an interesting choice.
- These last two solutions are briefly described on the next few slides

Using XML for Data Transfer

- There are a number of options for composing and interpreting XML.
  - It can be done in brute force fashion – by simply building a valid string
  - It can be done using DBMS XML construction code
  - It can be done by constructing a serializing a DOM object
- There are API's in most languages and the JavaScript and Java API's are very similar. The next page shows a sample in Java.

Reading and Parsing an XML Document

```
processRequest(HttpServletRequest req, HttpServletResponse resp)
{ try {         BufferedReader br = req.getReader();
                StringBuffer sb = new StringBuffer();
                String line = null;
                while((line=br.readLine())!=null) {sb.append(line);}
                DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
                DocumentBuilder builder = factory.newDocumentBuilder();
                InputSource is = new InputSource( new StringReader( sb.toString() ) );
                Document d = builder.parse( is );
                NodeList list = d.getElementsByTagName(“sometag”);//this is much more complex
                Element e = list[0];
                String id = e.getChild[0].getValue());
                if (dbmscheck(id))
                        {out.println(“This user has been validated");}
                else
                         {out.println(“This user is not validated");}
                } catch (Exception E){//really bad, but brief}
}
```

Building and Writing an XML Document

```
processRequest(HttpServletRequest req, HttpServletResponse resp)
{ try {         resp.setContentType("text/xml;charset=UTF-8");
                PrintWriter out = response.getWriter();
                DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
                DocumentBuilder builder = factory.newDocumentBuilder();
                Document d = builder.newDocument();
                Element x = d.createElement(“sometag”);
                Text y = d.createTextNote(“some text”);
                x.appendChild(y);
                d.appendChild(x);
                TransformerFactory tfac = TransformerFactory.newInstance();
                Transformer tf = tfac.newTransformer();
                StringWriter sw = new StringWriter();
                StreamResult result = new StreamResult(sw);
                DOMSource source = new DOMSource(d);
                trans.transform(source, result);
                out.println(sw.toString());
} catch (Exception E){//really bad, but brief}
}
```

From Objects to JSON

```java
public class Medication {
   private String medicationID;
   private String medicationName;
   private String[] sideEffects;
   private String usageDirections;
   private double maximumDosage;
   private String MAX_DOSAGE_MEASURE = "mg";
}
```

```java
public class Prescription {
   private String prescriptionID;
   private Date prescriptionDate;
   private String patientID;
   private String medicalProviderID;
   private ArrayList<Medication> medicationList = new
   ArrayList<Medication>();
   private String instructions;
}
```

```
{
        "prescriptionID" : "0001",
        "prescriptionDate" : "04/11/2013",
        "patientID" : "14343",
        "medicalProviderID" : "45465",
        "medicationList" : [
                {
                        "medicationID" : "12345",
                        "medicationName" : "Tylenol",
                        "usageDirections" : "Take 2 for headache",
                        "maximumDosage" : 1000
                },
                {
                        "medicationID" : "12346",
                        "medicationName" : "Thyrogen",
                        "usageDirections" : "Take 1 at least an hour before a meal",
                        "maximumDosage" : 175
                }
        ],
        "instruction" : "Do not take with food.  Best taken in the morning before breakfast."
}
```

JSON Rules:

- Data is in name/value pairs
- Data is separated by commas
- Curly braces hold objects
- Square brackets hold arrays

JSON – JavaScript Object Notation

- Format for sharing data
- Derived from JavaScript
- Language Independent
- An alternative to XML

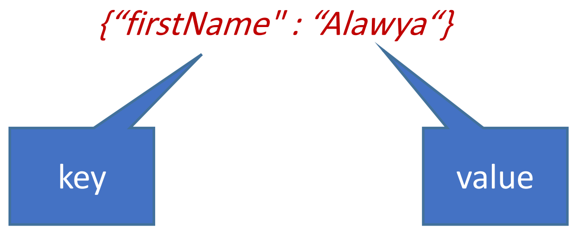
JSON – Advantages

- Easy to read
- Very few notations
- Minimal formatting
- Based on key-value pairs

JSON – Parsing

```
var userData = {
        “userID" : "0001",
        “firstName" : “Alawya",
        “lastName" : “Alawami",
        “dateOfBirth" : “01/18/1950“
}
```

JSON Objects



{“firstName" : “Alawya“}

- JSON keys are wrapped in “double quotes”
- JSON keys can be any valid string
- JSON values have to be one of six data types:

String, number, object, array, Boolean, null

Parsing JSON Strings

```
var data = '{"firstName" : "Alawya"}';
var obj = JSON.parse(data);
var backToString = JSON.stringify(info);
```

- JSON has to be parsed into JavaScript
- Can be done using *eval('(' + data + ')') ;*
- Can be done using newer JSON.parse();
- JSON.stringify does the opposite of parse

Keys and Values

```
var data = {
            “firstName"  : “Alawya“,
            “lastName”   : “Alawami”,
            “company”    : “Pitt”,
            “age”        : 68
};
```

- Multiple values are separated with commas
- *Last value is not followed by a comma*
- Use whitespace and indentation for clarity and readability
- *No quotes* around numbers, *true, false, null*

Retrieving Values

```
var data = {
            "firstName"  : "Alawya",
            "lastName"   : "Alawami",
            "company"    : "Pitt",
            "age"                      : 32
};
```

- Once parsed, JSON string becomes JavaScript object
- Keys and values are accessed through dot notation:

  data.firstName

- You can also use the [ ] notation:

  data["firstName"]

JSON Lists

```
var data = {
            “fullName"   : “Alawya Alawami”,
            “previousEmployement”:
            [
                        “University of Pittsburgh”,
                        “Aramco”,
                        “NeuroBehav”
            ]
};
```

- Lists are created with square brackets (just like Arrays)
- Access via an index number

```
data.previousEmployement[1]
```

JSON Objects in Lists

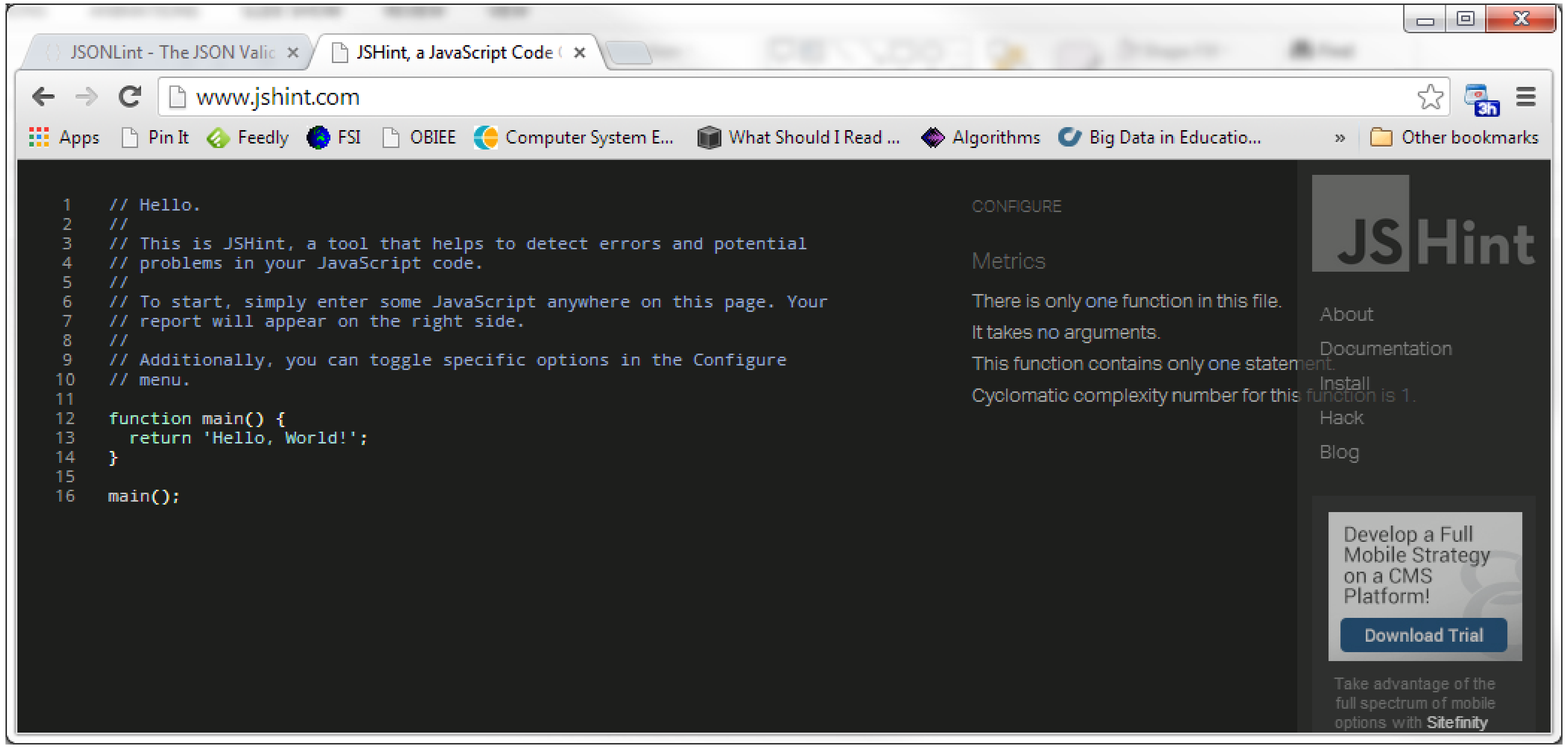
```
var data = {
            “fullName"   : “Dmitriy Babichenko”,
            “previousEmployement”:
            {
                        “job1” : “University of Pittsburgh”,
                        “job2” : “Aramco”,
                        “job3” : “NeuroBehav”
            }
};
```

- Objects are created with curly brackets
- Access via an object property

  data.previousEmployement.job1

JSON and JavaScript Validation Tools

JS Hint – JavaScript Analysis Tool: http://www.jshint.com/



```
// Hello.
//
// This is JSHint, a tool that helps to detect errors and potential
// problems in your JavaScript code.
//
// To start, simply enter some JavaScript anywhere on this page. Your
// report will appear on the right side.
//
// Additionally, you can toggle specific options in the Configure
// menu.

function main() {
  return 'Hello, World!';
}

main();
```

CONFIGURE

Metrics

There is only one function in this file.

It takes no arguments.

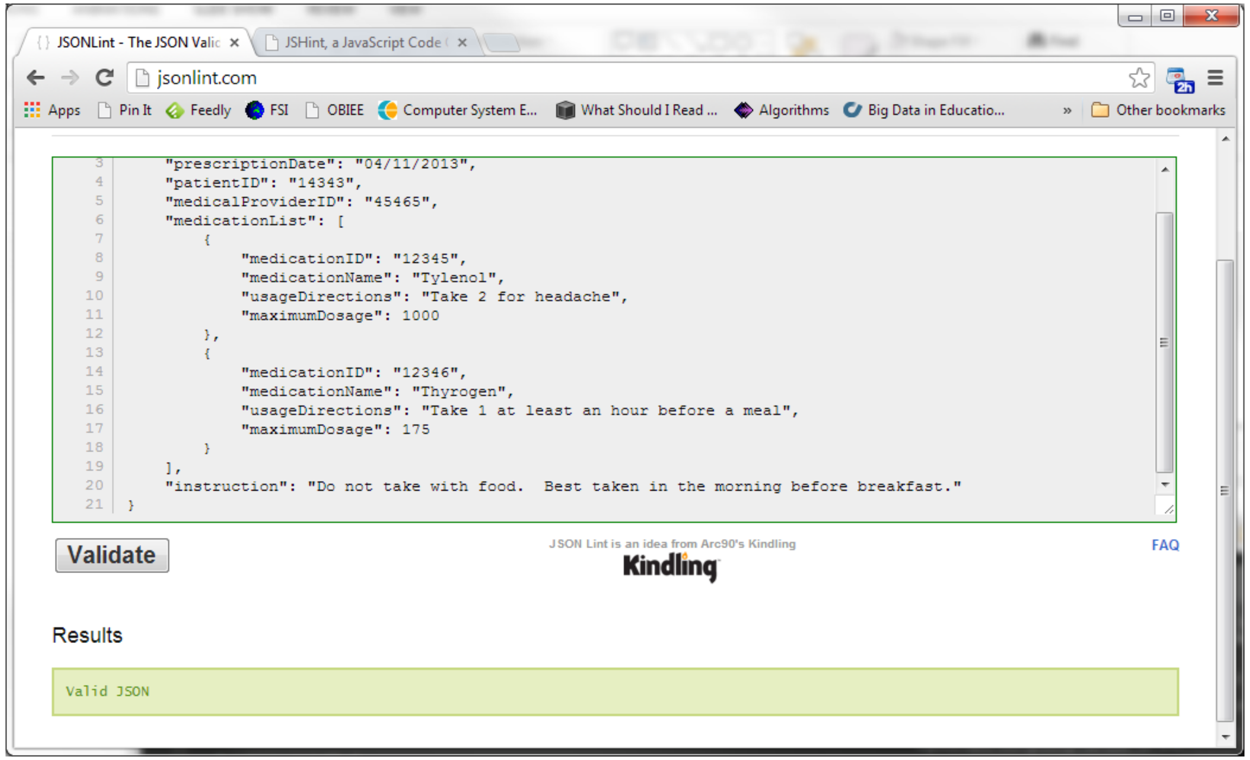
This function contains only one statement.

Cyclomatic complexity number for this function is 1.

JSHint
About
Documentation
Install
Hack
Blog
Develop a Full Mobile Strategy on a CMS Platform!
Download Trial
Take advantage of the full spectrum of mobile options with Sitefinity

JSON and JavaScript Validation Tools

JSONLint– JSON Validator: http://jsonlint.com/

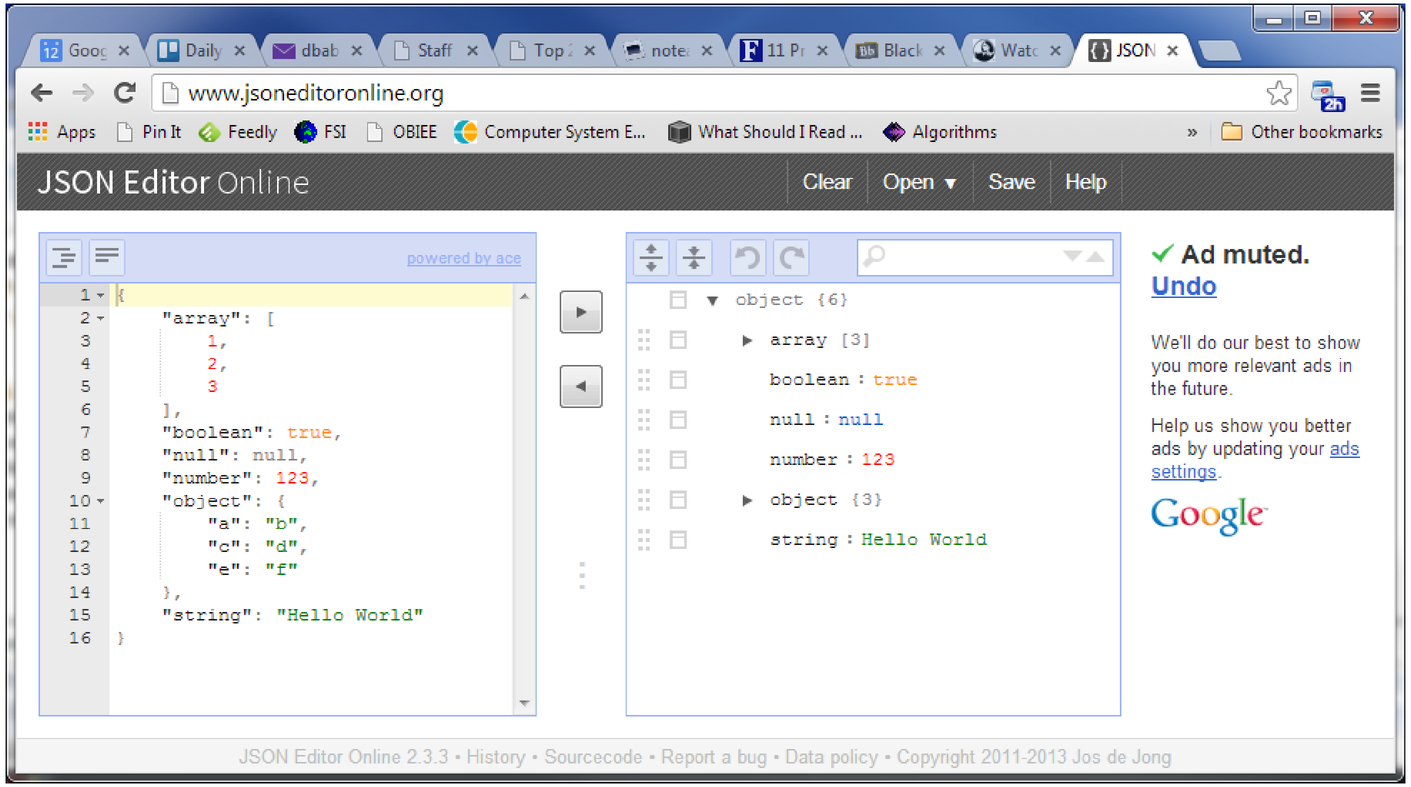


```
    "prescriptionDate": "04/11/2013",
    "patientID": "14343",
    "medicalProviderID": "45465",
    "medicationList": [
        {
            "medicationID": "12345",
            "medicationName": "Tylenol",
            "usageDirections": "Take 2 for headache",
            "maximumDosage": 1000
        },
        {
            "medicationID": "12346",
            "medicationName": "Thyrogen",
            "usageDirections": "Take 1 at least an hour before a meal",
            "maximumDosage": 175
        }
    ],
    "instruction": "Do not take with food.  Best taken in the morning before breakfast."
}
```

```
Valid JSON
```

JSON and JavaScript Validation Tools

JSON Editor Online: http://www.jsoneditoronline.org/



```
{
    "array": [
        1,
        2,
        3
    ],
    "boolean": true,
    "null": null,
    "number": 123,
    "object": {
        "a": "b",
        "c": "d",
        "e": "f"
    },
    "string": "Hello World"
}
```

```
▼ object {6}
    ▶ array [3]
      boolean : true
      null : null
      number : 123
    ▶ object {3}
      string : Hello World
```

✓ Ad muted.
Undo

We'll do our best to show you more relevant ads in the future.

Help us show you better ads by updating your ads settings.

Google

JSON Editor Online 2.3.3 • History • Sourcecode • Report a bug • Data policy • Copyright 2011-2013 Jos de Jong

Additional Resources for JavaScript

- W3CSchools - http://www.w3schools.com/js/
- JavaScript for Web Designers (Lynda.com) - http://www.lynda.com/JavaScript-tutorials/JavaScript-Web-Designers/144203-2.html
- JavaScript: Enhancing the DOM (Lynda.com) - http://www.lynda.com/HTML-tutorials/JavaScript-Enhancing-DOM/122462-2.html

JSON Resources

- Working with Data on the Web (Lynda.com) - http://www.lynda.com/CSS-tutorials/Working-Data-Web/133326-2.html
- JavaScript: Enhancing the DOM (Lynda.com) - http://www.lynda.com/JavaScript-tutorials/JavaScript-JSON/114901-2.html
- JSON Validation Tool: http://jsonlint.com/

Building and Sending a JSON Object

- A JSON Object can be constructed manually:

```
var jobj = {“name1”:”value1 (a string)”, “name2”: 13, “name3”: 65.3};
```

- They can be nested and arbitrarily complex.
- They can also be constructed through code as shown here

```
var myform=document.getElementById(formid);
for (i=0;i<myform.length;i++){
    if (myform[i].type == "text")
      {jobj[myform[i].name]=myform[i].value; }
}
```

- Once Constructed, they can be send using AJAX as follows:

```
sxhr.open("POST", target, true);
sxhr.send(JSON.stringify(jobj));
```

Toolkits

- A variety of toolkits have been developed and wrapped around AJAX to make it easier to use.
  - They attempt to overcome issues related to browser incompatibility
  - They attempt to simplify building dynamic capabilities into web pages
- The common toolkits include:
  - JQuery
  - Various Front-end frameworks
  - Dojo
  - Direct Web Remoting
  - Yahoo UI Library
  - Google Web Toolkit

The Easy Way!

JQuery

(Ajax and DOM manipulation)

jQuery Object/Method Structure

$* (dollar sign) is a shortcut for jQuery. Every time you see a JavaScript statement that begins with *$, it is a jQuery statement

```
$("#divExample").html();
```

id attribute of a <div> tag somewhere in the body of an HTML page.

A method call. Note the . (dot) notation – methods of an object are referenced the same way as in Java. This particular method gets or sets HTML contents of a DOM element.

(hashtag) signifies that we are trying to find an HTML element by its *id* attribute

DOM Manipulation

You can find everything you ever wanted to know about jQuery methods for manipulating DOM elements at http://api.jquery.com/category/manipulation/

Event Handling

- JavaScript will allow you to attach an event to any HTML element
- An event consists of 3 components:
  - Event – something that a user does with the application's UI
  - Event listener – a piece of code that waits for an event to occur. A listener is essentially a trigger
  - Event handler – a function that is executed when an event listener recognizes than an event has occurred.
- Native HTML/JavaScript event is somewhat cumbersome

Event Handling

id attribute of a button element somewhere in the body of an HTML page.

click method – specifies an event listener – now our HTML page waits for user to click on a button with an id attribute of “btnTest”

```
$( "#btnTest" ).click(function() {
    alert( "Handler for .click() called." );
});
```

Note that we are passing an entire function to the event listener method. That function is our ***event handler***.

Event Handling

```
$( "#btnTest" ).mouseover(function() {
     alert( "Handler for .mouseover() called." );
});
```

Event Handling

You can find everything you ever wanted to know about jQuery event handling at http://api.jquery.com/category/events/

jQuery AJAX Calls

```
jQuery.ajax({
      type: "GET",
      url: targetUrl,
      contentType: "application/json; charset=utf-8",
      dataType: "json",
      success: function(data, status, jqXHR) {
         console.log(data);
      },
      error: function(jqXHR, status) {
         console.log(status);
      }
});
```

HTTP method

URI of your web service (or any target web page)

Type of data we are expecting to get back

Event handler to process successfully returned data

Event handler to deal with errors

An Even Simpler jQuery AJAX Call

jQuery method (mirrors HTTP get method)

URI of your web service (or any target web page)

```
$.get( targetUrl, function( data ) {
        $( "#divExample" ).html( data );
});
```

Event handler to process successfully returned data

jQuery AJAX Calls

You can find everything you ever wanted to know about AJAX with jQuery at http://api.jquery.com/category/ajax/

jQuery Resources

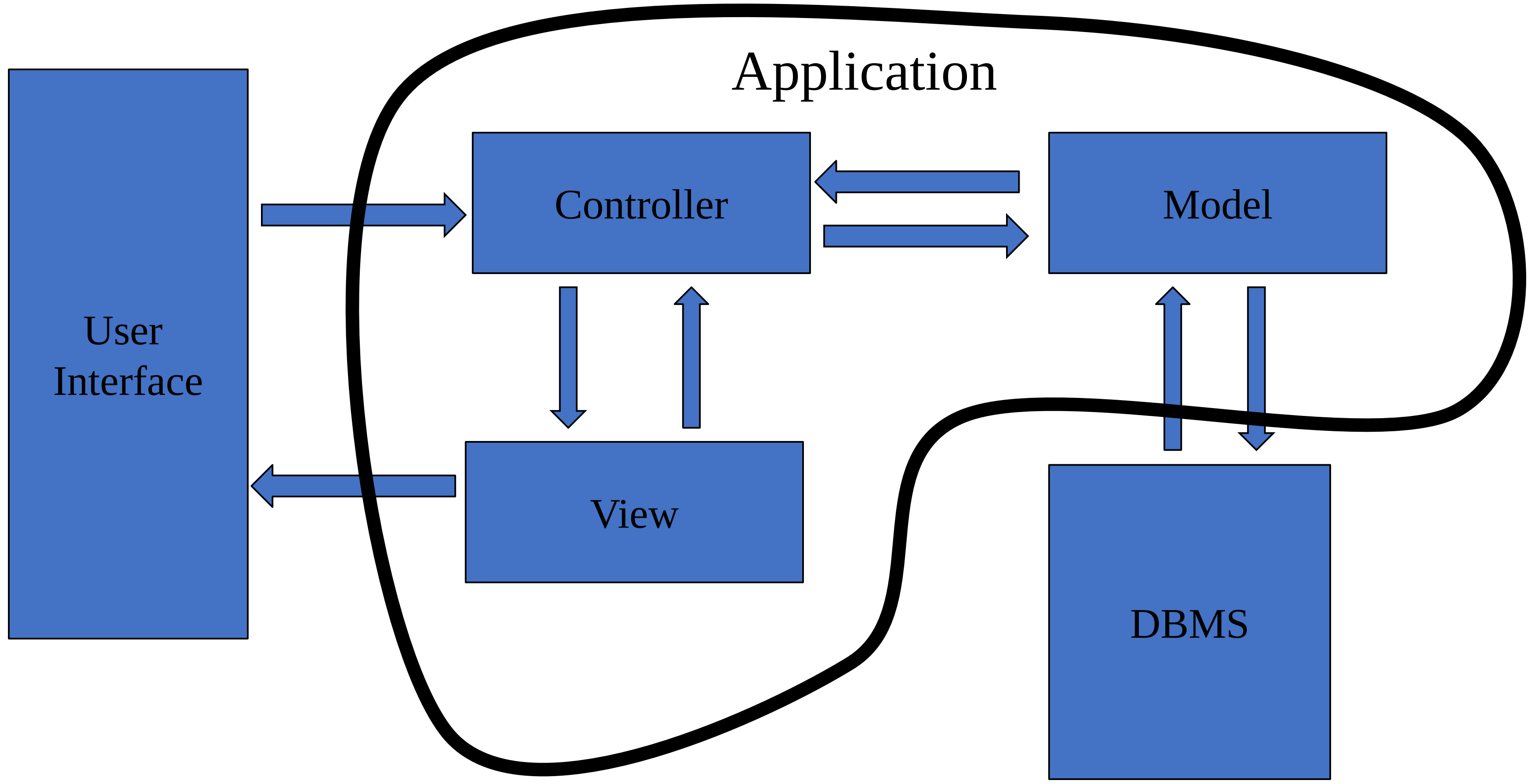
- jQuery Essential Training (lynda.com): http://www.lynda.com/jQuery-tutorials/essential-training/48370-2.html

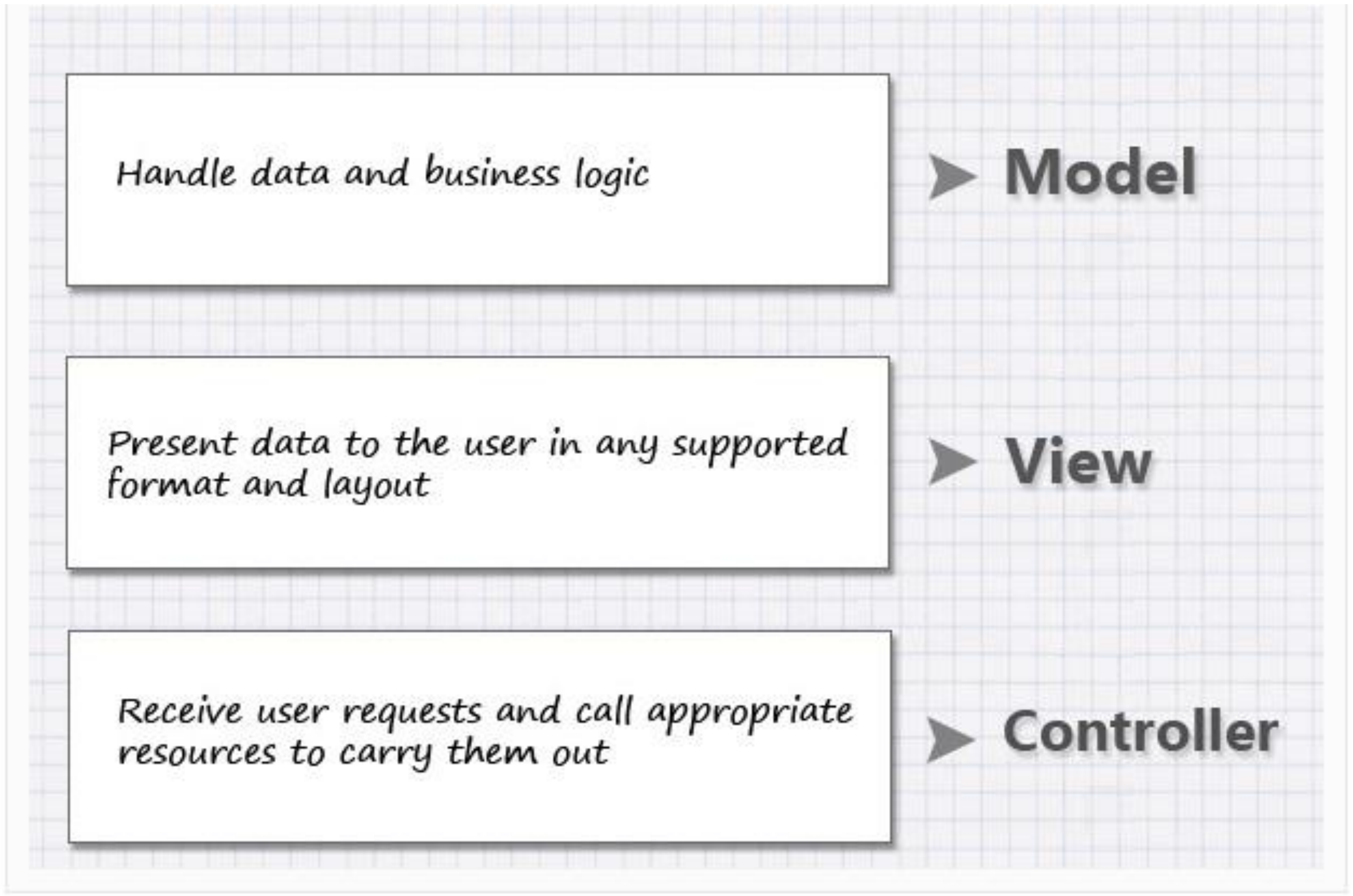
Websites Design Patterns

MVC Architecture

- The Model View Controller (MVC) Architecture is more than 30 years old.
  - It came from work at Xerox PARC on Smalltalk.
- It takes on various forms in various languages and involves three to five components:
  - <u>Model</u> – the abstract data model under control
  - <u>View</u> – one or more depictions of the model
  - <u>Controller</u> – business logic that determines the view based on model manipulation
  - <u>DB </u>– the ultimate repository for the model instantiation
  - <u>Interface</u> – the view from the client side which allows user input to the controller

MVC Graphically

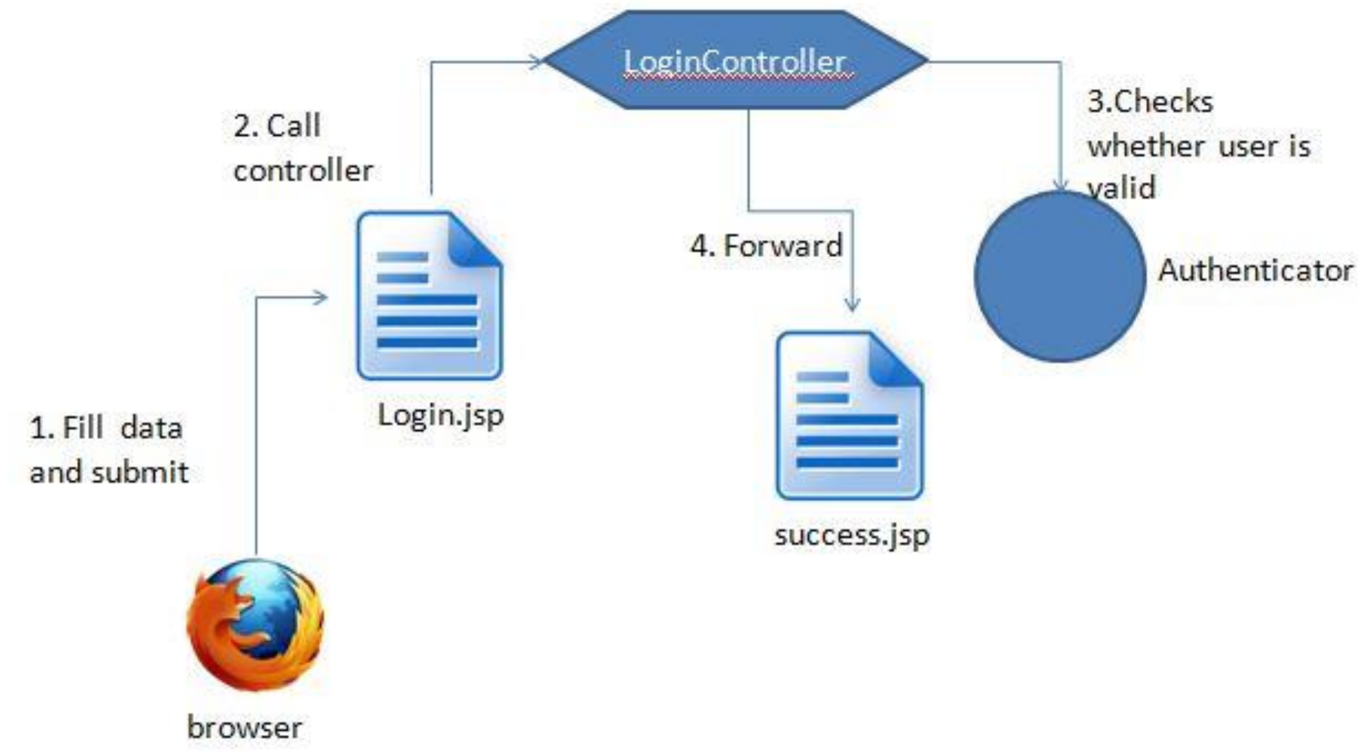



Handle data and business logic
Model
Present data to the user in any supported format and layout
View
Receive user requests and call appropriate resources to carry them out
Controller

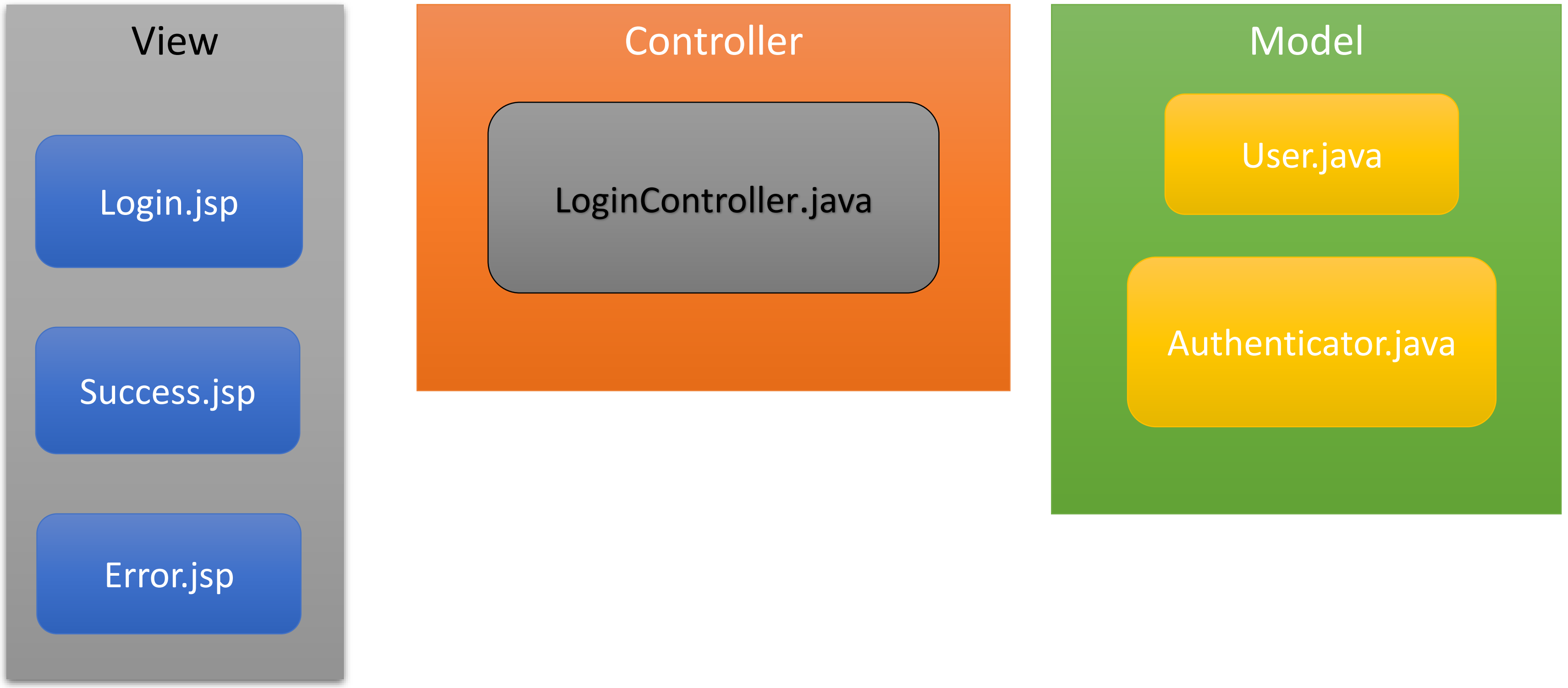
Java Web MVC Architecture

- The MVC architecture is implemented or reflected in a variety of ways in Java generally and for Java Web applications specifically
  - Web pages are considered a view and are produced by JSP
  - Servlets function best as controllers
  - Beans and Databases are considered "models"
- There are a variety of supporting technologies to support this architecture:
  - **Beans and Request dispatchers** are mechanisms to pass information from one servlet to another.
  - **Servlets** optimized for producing html pages are created using Java Server Pages (JSP) that are easier to write
  - **Session objects** and contexts may be obtained to store state information
  - Ultimately, tag libraries and frameworks provide additional functionality

MVC


LoginController
2. Call controller
3.Checks whether user is valid
4. Forward
Authenticator
Login.jsp
success.jsp
1. Fill data and submit
browser

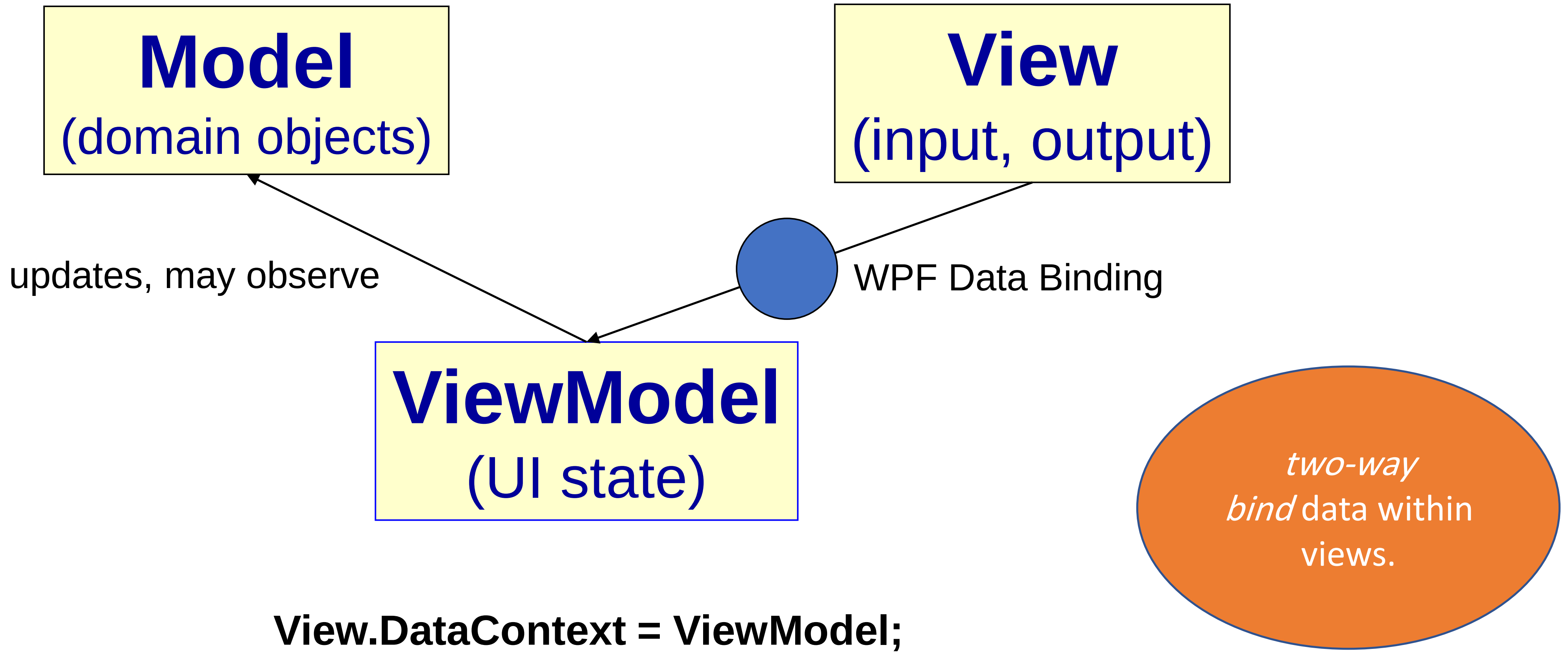
MVC example


View
Login.jsp
Success.jsp
Error.jsp
Controller
LoginController.java
Model
User.java
Authenticator.java

MVVM

Overview of MVVM

Model-View-ViewModel



View.DataContext = ViewModel;

http://blogs.msdn.com/b/johngossman/archive/2005/10/08/478683.aspx

Overview of MVVM

Separation of Concerns

Model	View	ViewModel
Read list of countries from the database	Position UI elements on screen	Validate input and show error indicators if necessary
Create shipment	Control visual appearance of the UI elements: colors, fonts, etc.	Call model to create shipment with data entered by the user
	Translate keystrokes to navigation and edit actions	Disable subdivision combo box if selected country has no subdivisions
	Translate mouse clicks to focus changes and button commands	

Overview of MVVM

Important MVVM Traits

- View is isolated from the model
- ViewModel does not manipulate controls directly
- **Most** of the View $\leftrightarrow$ ViewModel interaction is via data binding
- Codebehind is therefore kept to a minimum

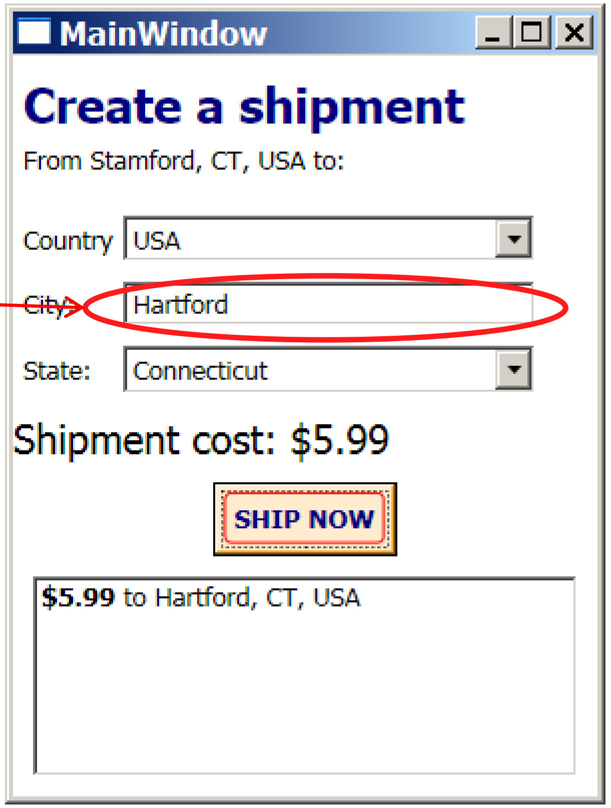
Overview of MVVM

WPF Data Binding

<TextBox Text=“{Binding City}” />

```
class MainWindowViewModel
{
    public string City
    {
        get { … }
        set { … }
    }
    …
}
```

binding



Country: USA
City: Hartford
State: Connecticut

Shipment cost: $5.99

$5.99 to Hartford, CT, USA

What is the difference?

Does view model replace controller (MVVM vs MVC)

MVVM VS MVC

- MVVM
  - Client side
  - Two way binding data
- MVC
  - a way of separating concerns *on the server-side*.

Angular js

Slides modified from Stanford University CS142

Slides from CMU university Prof.**Michael J. McCarthy**

What is Angular js?

- MVVM Java Script Framework by Google for Rich Web Application Development
- Why?

Overcome HTML Shortcoming

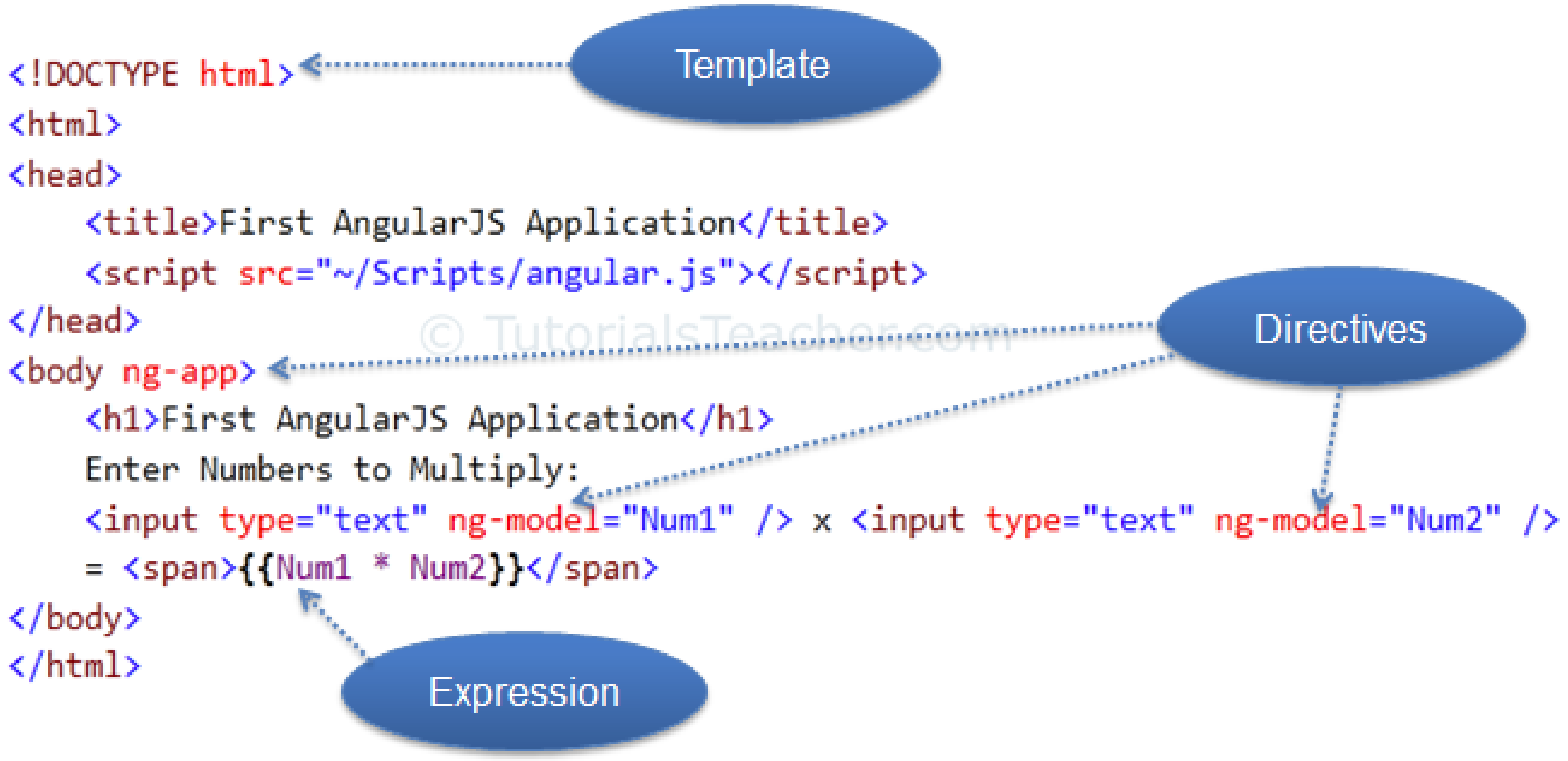
Why Angular?

“Other frameworks deal with HTML’s shortcomings by either abstracting away HTML, CSS, and/or JavaScript or by providing an imperative way for manipulating the DOM. Neither of these address the root problem that HTML was not designed for dynamic views”.

- Structure, Quality and Organization
- Lightweight (< 36KB compressed and minified)
- Free
- Separation of concern
- Modularity
- Extensibility & Maintainability
- Reusable Components

“ HTML? Build UI Declaratively! CSS? Animations! JavaScript? Use it the plain old way!”

Big picture



```
<!DOCTYPE html>
<html>
<head>
    <title>First AngularJS Application</title>
    <script src="~/Scripts/angular.js"></script>
</head>
<body ng-app>
    <h1>First AngularJS Application</h1>
    Enter Numbers to Multiply:
    <input type="text" ng-model="Num1" /> x <input type="text" ng-model="Num2" />
    = <span>{{Num1 * Num2}}</span>
</body>
</html>
```

First AngularJS Application

In AngularJS, a template is HTML with additional markups. AngularJS compiles templates and renders the resultant HTML.

JQuery

- Allows for DOM Manipulation
- Does not provide structure to your code
- Does not allow for two way binding

Simple Angular js Page

```
<!DOCTYPE html>
<html>
<script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></script>
<body>
<div ng-app="">
<p>Input something in the input box:</p>
<p>Name: <input type="text" ng-model="name"></p>
<p ng-bind="name"></p>
</div>
</body>
</html>
```

No installation

- It is recommended that you load the AngularJS library either in the <head> or at the start of the <body>.

Features of AngularJS

- Two-way Data Binding – Model as single source of truth
- Directives – Extend HTML
- MVC
- Dependency Injection
- Testing
- Deep Linking (Map URL to route Definition)
- Server-Side Communication

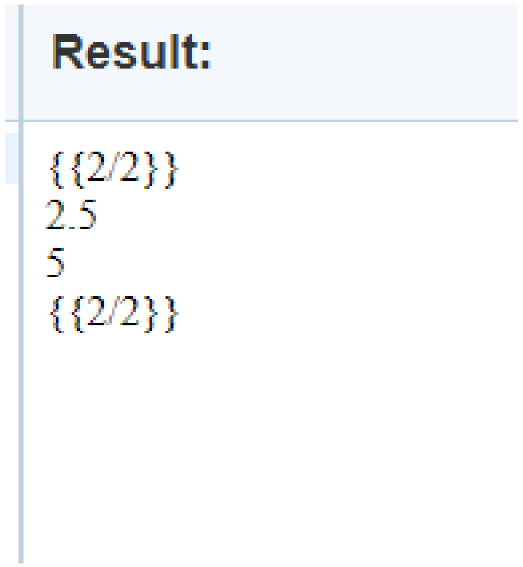
Data Binding

```
<html ng-app>
<head>
  <script src='angular.js'></script>
</head>
<body>
  <input ng-model='user.name'>
  <div ng-show='user.name'>Hi {{user.name}}</div>
</body>
</html>
```

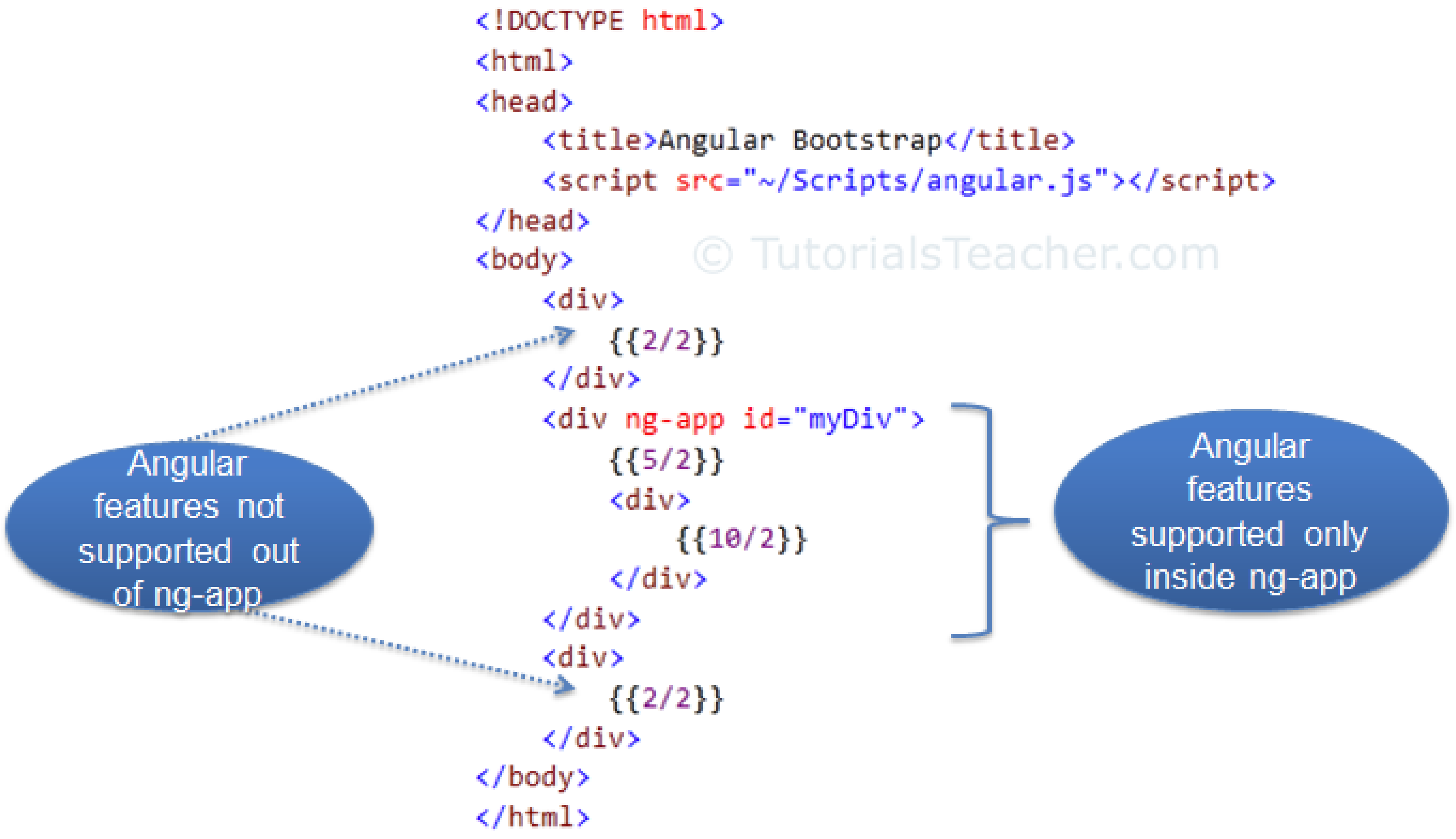
ng-app in <div>

AngularJS framework will only process the DOM elements and its child elements where the ng-app directive is applied

```
<!DOCTYPE html>
<html>
<head>
<title>ng-app Directive</title>
<script src="../Scripts/angular.min.js"></script>
</head>
<body >
<div> {{2/2}} </div>
<div id="myDiv" ng-app>
{{5/2}}
       <div> {{10/2}} </div>
</div>
<div>{{2/2}}</div>
 </body>
 </html>
```


Result:

```
{{2/2}}
2.5
5
{{2/2}}
```

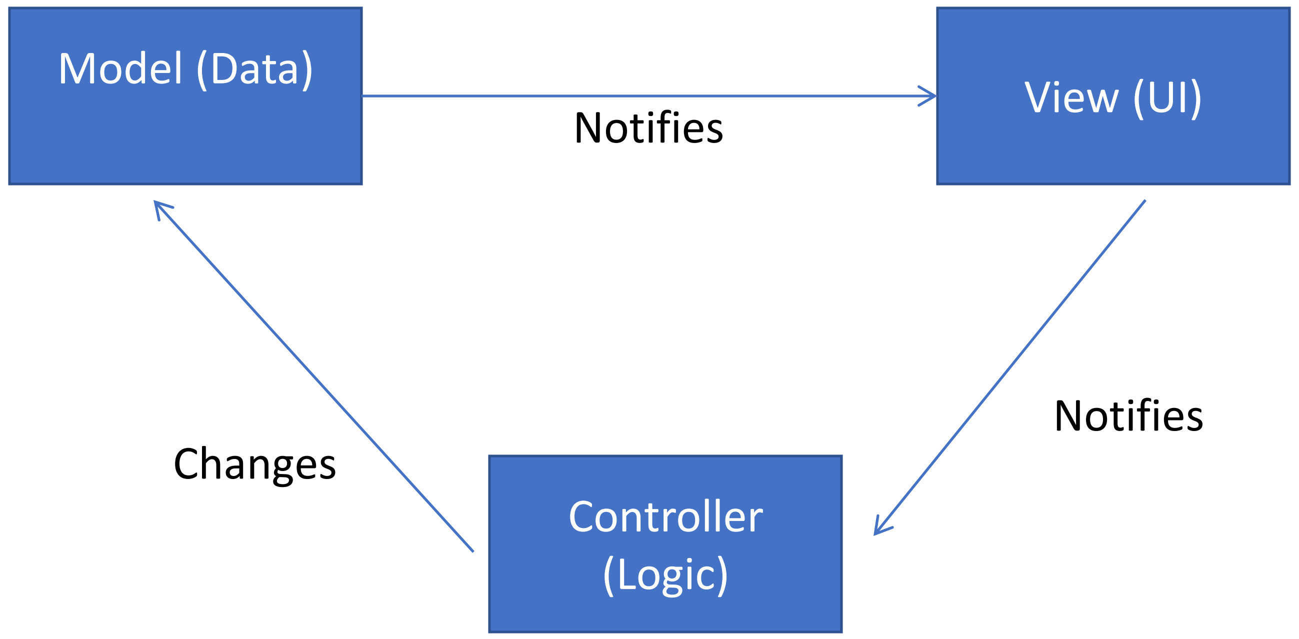


```
<!DOCTYPE html>
<html>
<head>
    <title>Angular Bootstrap</title>
    <script src="~/Scripts/angular.js"></script>
</head>
<body>
    <div>
        {{2/2}}
    </div>
    <div ng-app id="myDiv">
        {{5/2}}
        <div>
            {{10/2}}
        </div>
    </div>
    <div>
        {{2/2}}
    </div>
</body>
</html>
```

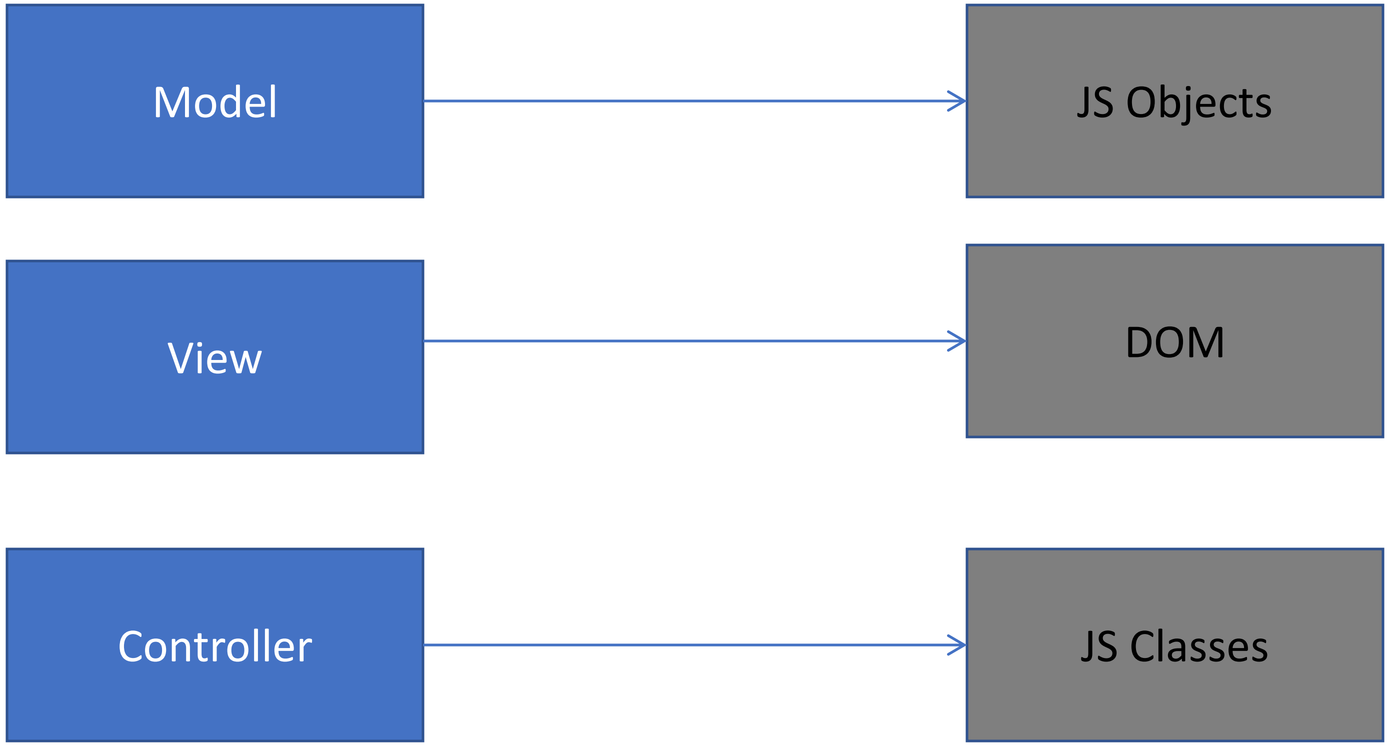
Angular features not supported out of ng-app
Angular features supported only inside ng-app

Bootstrap

MVC


Model (Data)
Notifies
View (UI)
Notifies
Changes
Controller (Logic)

MVC



MVC

```
<html ng-app>
<head>
  <script src='angular.js'></script>
  <script src='controllers.js'></script>
</head>
<body ng-controller='UserController'>
  <div>Hi {{user.name}}</div>
</body>
</html>

function  XXXX($scope) {
  $scope.user = { name:'Larry' };
}
```

Hello HTML

```
<p>Hello World!</p>
```

Hello Javascript

```
<p id="greeting1"></p>
<script>
var isIE = document.attachEvent;
var addListener = isIE ? function(e, t, fn) {
      e.attachEvent('on' + t, fn);}
  : function(e, t, fn) {
      e.addEventListener(t, fn, false);};
addListener(document, 'load', function(){
 var greeting = document.getElementById('greeting1');
  if (isIE) {
    greeting.innerText = 'Hello World!';
  } else {
    greeting.textContent = 'Hello World!'; }});
</script>
```

Hello JQuery

```
<p id="greeting2"></p>

<script>
$(function(){
  $('#greeting2').text('Hello World!');
});
</script>
```

Hello AngularJS

```
<p ng:init="greeting = 'Hello World!'">{{greeting}}</p>
```

Step by Step tutorial (very simple)

Feeder App

http://www.toptal.com/angular-js/a-step-by-step-guide-to-your-first-angularjs-app

Expressions

Expressions allow you to execute some computation in order to return a desired value.

AngularJS expressions are written inside double braces: **{{ expression }}**.

- {{ 1 + 1 }}
- {{ 946757880 | date }}
- {{ user.name }}

AngularJS will "output" data exactly where the expression is written:

you shouldn’t use expressions to implement any higher-level logic.

Directives

- Directives are markers (such as attributes, tags, and class names) that tell AngularJS to attach a given behavior to a DOM element (or transform it, replace it, etc.)

- Directives are HTML attributes with an **ng** prefix.
- You can use **data-ng-**, instead of **ng-**, if you want to make your page HTML valid.

Some directives

- The **ng-app** - Bootstrapping your app and defining its scope.
- The **ng-controller** - defines which controller will be in charge of your view.
- The **ng-repeat** - Allows for looping through collections
- The **ng-app** directive tells AngularJS that the <div> element is the "owner" of an AngularJS **application**.
- The **ng-model** directive binds the value of the input field to the application variable **name**.
- The **ng-bind** directive binds the content of the <p> element to the application variable **name**.
- https://www.w3schools.com/angular/angular_ref_directives.asp

ng-init

- **ng-init** directive initializes AngularJS application variables.

```
<div ng-app="" ng-init="firstName='John'">

<p>The name is <span ng-bind="firstName"></span></p>

</div>
```

Angular Datatypes

- Exactly the same as JavaScript
- String
- Numbers
- Objects
- Array
- Same syntax
- Like JavaScript expressions, AngularJS expressions can contain literals, operators, and variables.
- Unlike JavaScript expressions, *AngularJS expressions can be written inside HTML.*

Angular module

- The ng-app directive can also specify an application module name. This application module separates different parts of your application such as controllers, services, filters etc.

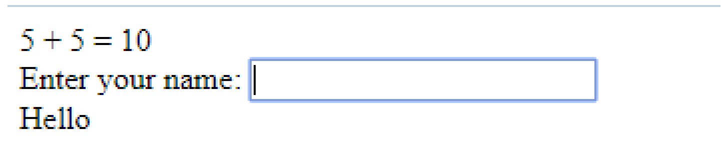
```
<!DOCTYPE html>
<html>
<head>
    <title>ng-app Directive</title>
    <script src="~/Scripts/angular.js"></script>
</head>
<body ng-app="myAngularApp">
    <div>
        {{2/2}}
    </div>
    <div>
        {{5/2}}
        <div>
            {{10/2}}
        </div>
    </div>
    <script>
        var app = angular.module('myAngularApp', []);
    </script>
</body>
</html>
```

ng-model and ng-bind

- The ng-model directive is used for two-way data binding in AngularJS. It binds <input>, <select> or <textarea> elements to a specified property on the $scope object. So, the value of the element will be the value of a property
- The ng-bind directive binds the model property declared via $scope or ng-model directive or the result of an expression to the HTML element.
- It also updates an element if the value of an expression changes

Example

```
<!DOCTYPE html>
<html >
<head>
    <script src="~/Scripts/angular.js"></script>
</head>
<body ng-app="">
    <div>
        5 + 5 = <span ng-bind="5 + 5"></span> <br />

        Enter your name: <input type="text" ng-model="name" /><br />
        Hello <span ng-bind="name"></span>
    </div>
</body>
</html>
```

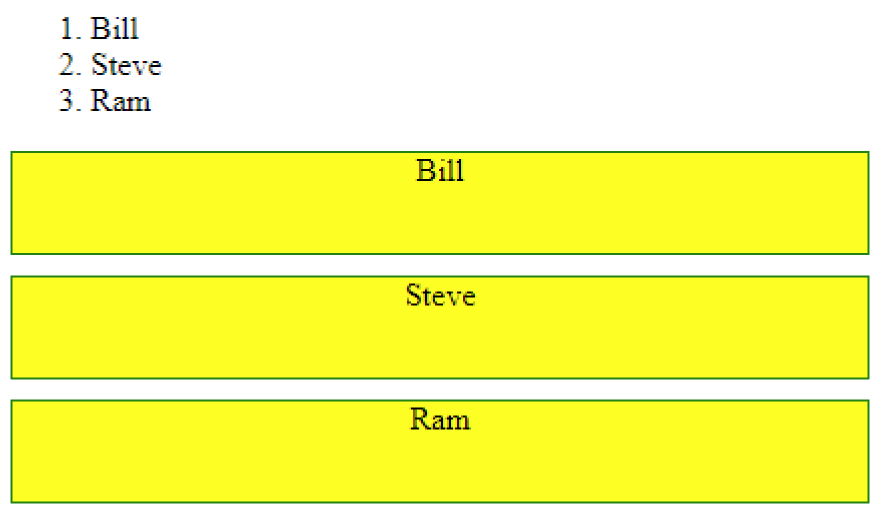

5 + 5 = 10
Enter your name:
Hello

Try it here

ng-repeat

```
<!DOCTYPE html>
<html>
<head>
    <script src="~/Scripts/angular.js"></script>
    <style>
        div {
            border: 1px solid green;
            width: 100%;
            height: 50px;
            display: block;
            margin-bottom: 10px;
            text-align:center;
            background-color:yellow;
        }
    </style>
</head>
<body ng-app="" ng-init="students=['Bill','Steve','Ram']">
    <ol>
        <li ng-repeat="name in students">
            {{name}}
        </li>
    </ol>
    <div ng-repeat="name in students">
        {{name}}
    </div>
</body>
</html>
```

The ng-repeat directive repeats HTML once per each item in the specified array collection.



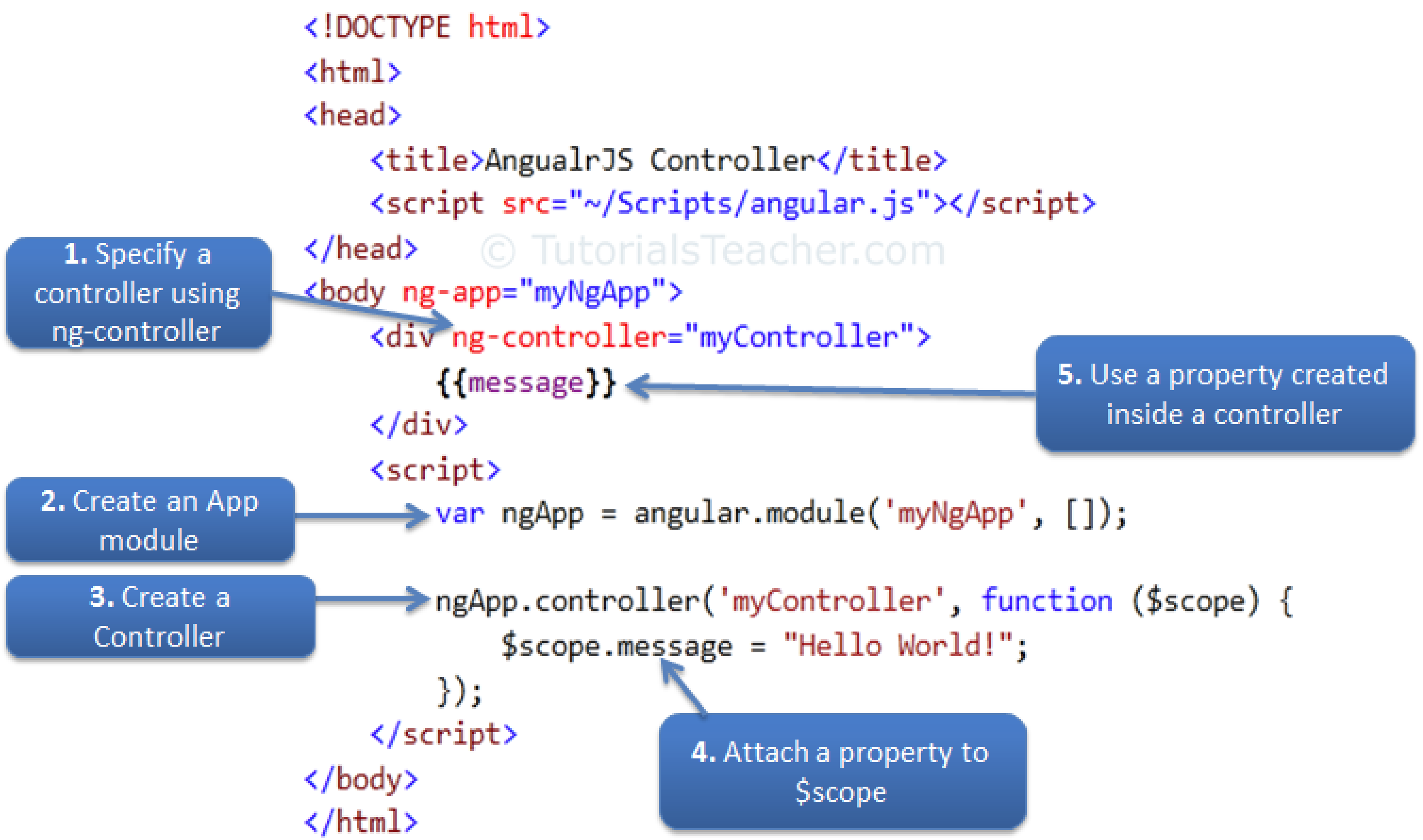
```
1. Bill
2. Steve
3. Ram
```

```
Bill
Steve
Ram
```

Angular Controller

- The controller in AngularJS is a JavaScript function that maintains the application data and behavior using $scope object.
- The $scope object is a glue between the controller and HTML

- Illustrated in the next slide.



```
<!DOCTYPE html>
<html>
<head>
    <title>AngualrJS Controller</title>
    <script src="~/Scripts/angular.js"></script>
</head>
<body ng-app="myNgApp">
    <div ng-controller="myController">
        {{message}}
    </div>
    <script>
        var ngApp = angular.module('myNgApp', []);

        ngApp.controller('myController', function ($scope) {
            $scope.message = "Hello World!";
        });
    </script>
</body>
</html>
```

1. Specify a controller using ng-controller
2. Create an App module
3. Create a Controller
4. Attach a property to $scope
5. Use a property created inside a controller

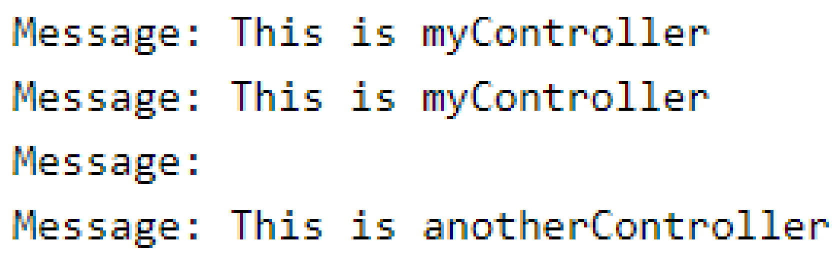
Steps to create an AngularJS Controller

Note that the properties and methods attached to the scope object inside a particular controller is only available to the HTML elements and its child elements where ng-controller directive is applied.

- Try it here

Two controllers

```
<!DOCTYPE html>
<html>
<head>
    <title>AngualrJS Controller</title>
    <script src="~/Scripts/angular.js"></script>
</head>
<body ng-app="myNgApp">
    <div id="div1" ng-controller="myController">
        Message: {{message}} <br />
        <div id="div2">
            Message: {{message}}
        </div>
    </div>
    <div id="div3">
        Message: {{message}}
    </div>
    <div id="div4" ng-controller="anotherController">
        Message: {{message}}
    </div>
    <script>
        var ngApp = angular.module('myNgApp', []);

        ngApp.controller('myController', function ($scope) {
            $scope.message = "This is myController";
        });

        ngApp.controller('anotherController', function ($scope) {
            $scope.message = "This is anotherController";
        });
    </script>
</body>
</html>
```



```
Message: This is myController
Message: This is myController
Message:
Message: This is anotherController
```

Structure

- It is common in AngularJS applications to put the module and the controllers in JavaScript files.
- In this example, "myApp.js" contains an application module definition, while "myCtrl.js" contains the controller:

```
<!DOCTYPE html>
<html>
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></script>
<body>

<div ng-app="myApp" ng-controller="myCtrl">
{{ firstName + " " + lastName }}
</div>

<script src="myApp.js"></script>
<script src="myCtrl.js"></script>

</body>
</html>
```

myApp.js
```
var app = angular.module("myApp", []);
```

myCtrl.js
```
app.controller("myCtrl", function($scope) {
    $scope.firstName = "John";
    $scope.lastName= "Doe";});
```

Advanced AngularJS Concept

- Dependency Injection
- Modularity
- Digesting
- Scope
- Handling SEO
- End to End Testing
- Promises
- Localization
- Filters

Disadvantages of AngularJS

- **Not Secure –** Its applications are not safe. Server side authentication and authorization is necessary to keep an application secure.
- **Not Degradable –** If user of your application disables the JavaScript then it displays nothing except basic page.
- **Complex at times –** At times AngularJS becomes complex to handles as there are multiple ways to do the same thing. This creates confusion and requires considerable efforts.

Useful Links

- https://angularjs.org/
- http://campus.codeschool.com/courses/shaping-up-with-angular-js/contets
- http://www.toptal.com/angular-js/a-step-by-step-guide-to-your-first-angularjs-app
- https://github.com/raonibr/f1feeder-part1

Final project

- # phases
  - Phase 1: proposal due 10/5 (unusual due date)
  - Phase 2: Front-end designs (mockup screen) + DB design(due 10/29)
  - Phase 3: back-end and full functionality
  - Technology : Mean framework
- Requirement: Complete web site that will integrate:
  - Data storage (MONGODB as we are using Mean framework).
    - Mongoose ORM is used to connect with MongoDB, you can use Sequelize which is an ORM to connect to MYSQL.
  - Multiple users with different interfaces (Admin view vs User view)
  - MVC structure
  - Admin capabilities
  - Location aware
  - Session management capabilities(chat, shopping cart using socket io)
  - RESTful Web service
  - Ex: review site, social network site, blog, chat app

For Next week

- We will wrap up Front-end
  - Angular
  - We will probably start Web storage as well which is week 8 topic
- Reading
  - Node.js in Action chapter 8 (covers databases)
  - Review MYSQL
  - https://www.w3schools.com/html/html5_webstorage.asp
  - https://www.w3schools.com/nodejs/nodejs_mysql.asp
  - https://www.w3schools.com/nodejs/nodejs_mongodb_create_db.asp
- Proposal and assignment 2 due next week.
- Install
  - Node.js
  - MongoDB
  - MYSQL(if you have XAMPP installed you probably have MySQL installed with it)