.NET Framework: Developing Modern Web Apps with ASP.NET MVC

WorkshopPlus

< Engineer Name >

Premier Field Engineer

Conditions and Terms of Use

Microsoft Confidential

This training package is proprietary and confidential, and is intended only for uses described in the training materials. Content and software is provided to you under a Non-Disclosure Agreement and cannot be distributed. Copying or disclosing all or any portion of the content and/or software included in such packages is strictly prohibited.

The contents of this package are for informational and training purposes only and are provided "as is" without warranty of any kind, whether express or implied, including but not limited to the implied warranties of merchantability, fitness for a particular purpose, and non-infringement.

Training package content, including URLs and other Internet website references, is subject to change without notice. Because Microsoft must respond to changing market conditions, the content should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information presented after the date of publication. Unless otherwise noted, the companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, e-mail address, logo, person, place, or event is intended or should be inferred.

Copyright and Trademarks

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

For more information, see **Use of Microsoft Copyrighted Content** at
http://www.microsoft.com/about/legal/permissions/

Microsoft®, Internet Explorer®, Outlook®, OneDrive®, Windows Vista®, Zune®, Xbox 360®, DirectX®, Windows Server® and Windows® are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. Other Microsoft products mentioned herein may be either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. All other trademarks are property of their respective owners.

How to View This Presentation

Switch to Notes Page view:

- Click **View** on the ribbon and select **Notes Page**
- Use page up or page down to navigate
- Zoom in or out as needed

In the Notes Page view you can:

- Read any supporting text, now or after the delivery
- Add your own notes

Take the presentation files home with you

Module 6: Client-Side Development

Module Overview

- MVC 6 support for client-side development
  - Client-side files in MVC 6 project templates
  - Bower, Gulpjs/Gruntjs
- Introduction to Bootstrap
- Primer on JavaScript, Jquery and AJAX
- Single Page Application explained (knockoutjs/angularjs)

Module 6: Client-Side Development

Section 1: Support

Lesson: MVC and JavaScript

Why using JavaScript in ASP.NET MVC?

- Combining server-side and client-side processing.
- More responsive web applications.
- JavaScript comes in many forms:
    - AJAX – Partial page update and refresh
    - Jquery – Elegantly and Efficiently find and manipulate HTML DOM elements
    - MooTools – Modular JavaScript and code reuse
    - Prototype – Simplify development of dynamic web application
    - Node.js – Developing high performance JavaScript via multithreading model
    - Industry standard for modern web development
    - And many more

Project Template – wwwroot folder

- All static files should be located in this folder (JavaScripts, CSS, images or HTML files)
- wwwroot folder is the root of the website
    - http://hostname/ points to wwwroot
    - All static content should be relative to the folder
- Code files should be places outside of wwwroot (C# or Razor).
- Static files created through compilation or pre-processing should be copied into wwwroot.
- Can be renamed by changing “webroot” setting in the project.json file.

Project Template – JavaScript Libraries Part 1

- _references.js
    - Contains JavaScript reference in the form of comments
    - Allows Visual Studio to augment IntelliSense support for JavaScript
    - The autosync flag set to true:
        - JavaScript files are automatically be added.
        - Automatic update when a referenced file is moved.
    - Manual update is possible via right clicking on the file.
- Bootstrap.js and bootstrap.min.js
    - HTML, CSS and JavaScript-based design templates for creating responsive web sites.
- Note! Always use the .min version of a JavaScript file to improve load time.

Project Template – JavaScript Libraries Part 2

- Bootstrap-touch-carousel & hammer.js
  - A slideshow component for cycling through elements, like a carousel.
  - Enable gestures on touch devices using hammer.js, a javascript library for multi-touch gestures.

Project Template – JavaScript Libraries Part 3

- Jquery-version.intellisense.js
  - Extending IntelliSense for jQuery library
- Jquery-version.js and jquery-version.min.js
  - Main JQuery version
- Jquery-version.min.map
  - Allows to map to the un-minified version of JQuery for troubleshooting purposes.
- Jquery.validate.js (Jquery.validate.min.js)
  - Provide client side validation using jQuery
  - Multilanguage support
- Jquery.validate.unobtrusive.js and jquery.validate.unobtrusive.min.js

Client-Side Development Configuration Files

- gulpfile.js
  - Javascript Configuration of Gulp tasks
- Project.json
  - Main project file. NuGet package dependencies are listed here
- Package.json
  - Lists npm packages
- Bower.json
  - Lists Bower packages

Module 6: Client-Side Development

Section 1: Support

Lesson: Bower and Gulp

Why using Gulp (or Grunt) and Bower?

- Modern web applications incorporate various and rich client-side libraries such as jQuery, TypeScript, Bootstrap etc.
- Easy management of client-side packages
- Automating build tasks such as scripts compilation, bundling, minification or unit testing.
- Leverage existing tools from the web development community.

What is Bower?

- “A package manager for the web” (http://bower.io)
- Installs and restores client-side libraries.
- Keeps track of all the packages in a manifest file, bower.json
- Improves page load.

What is Gulp?

- “The JavaScript Task Runner” (http://gulpjs.com)
- An application to automates routine client-side development tasks (compilation, bundling, minification, unit testing etc…)
- gulpfile.js contains Gulp tasks with JavaScript-like configuration
- Grunt is another task runner.
    - Gulpfile.js uses JSON-like syntax for configuration

Bundling and Minification

- Bundling reduces the number of requests to the server
    - Combining multiple files into a single file.
    - Create CSS, JavaScript and other bundles.
    - Use the “Include” or “IncludeDirectory” of the Bundle Class
- Minification reduces the size of the requested assets (CSS and JavaScript).
- MVC 6 uses Gulp or Grunt to achieve bundling and minification.

Demo: Bower and Gulp

Module 6: Client-Side Development

Section 2: Techniques

Lesson: Styling with Bootstrap

Intro to Bootstrap

Why use it?

- CSS is can be tricky
- Cross browser support can be a challenge
- Solves basic tasks (e.g. page layout without tables)
- Bootstrap 3 makes it easier

Browser Support

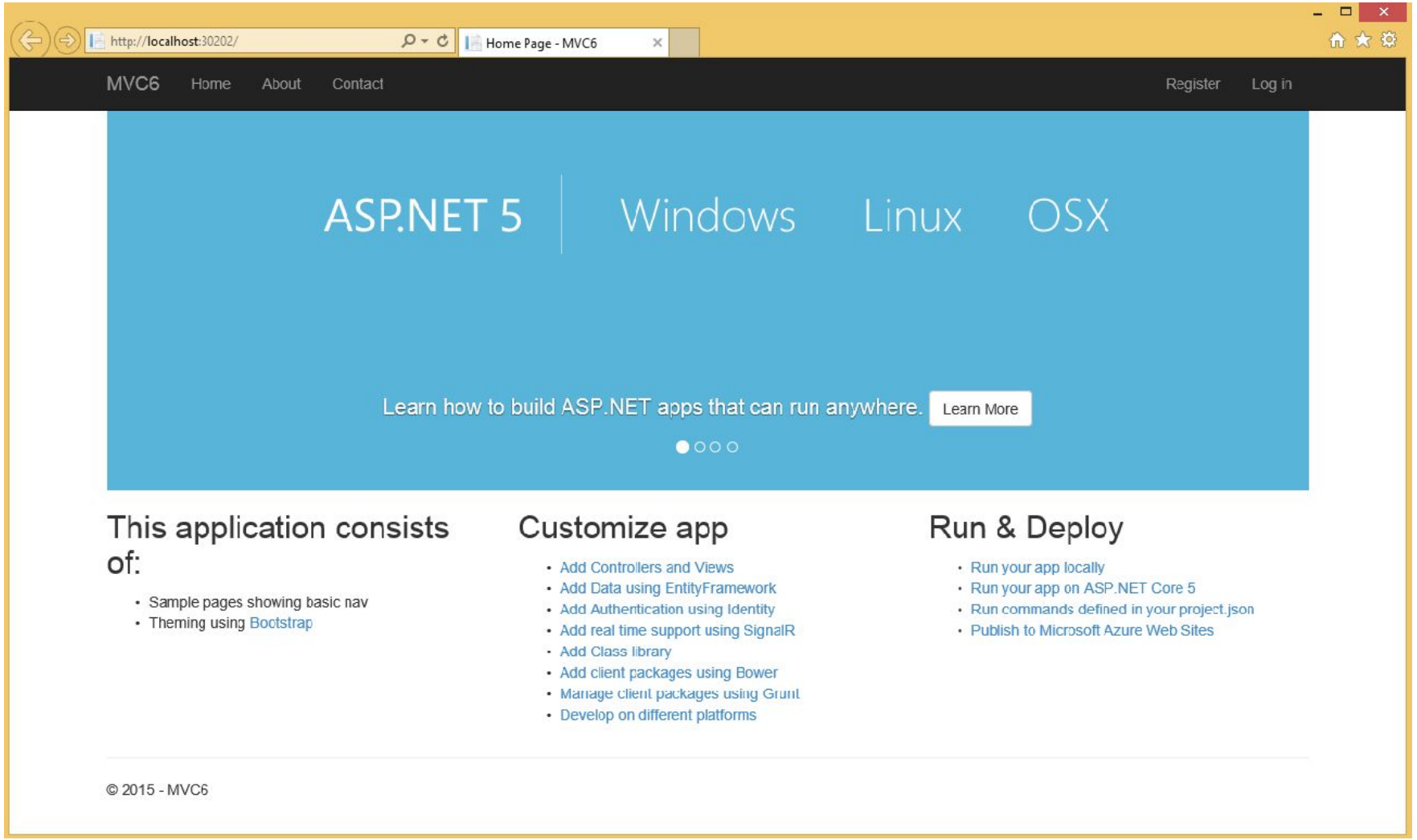
Support the latest version of browsers on the following systems:

	Chrome	Firefox	Internet Explorer	Opera	Safari
Android	✔ Supported	✔ Supported	N/A	✖ Not Supported	N/A
iOS	✔ Supported	N/A		✖ Not Supported	✔ Supported
Mac OS X	✔ Supported	✔ Supported		✔ Supported	✔ Supported
Windows	✔ Supported	✔ Supported	✔ Supported	✔ Supported	✖ Not Supported

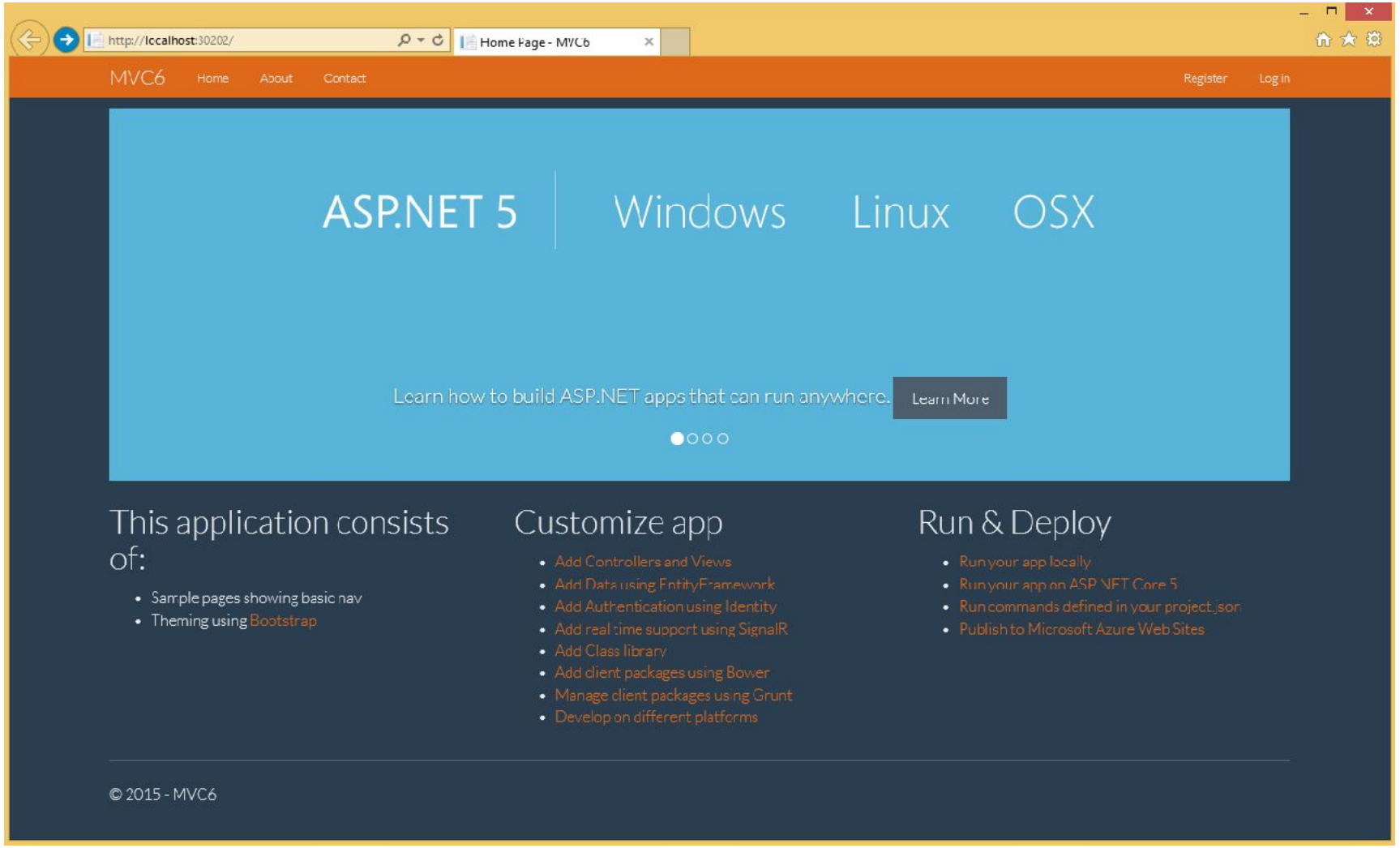
Bootstrap Features

- Theme Support
- Responsive
- Grid
- Components
  - Pagination
  - Buttons
  - Modal
- Great Visual Studio support

Theme Support (Default)



Easy Theme Support (Custom)



Responsive Layout

One framework, every device.

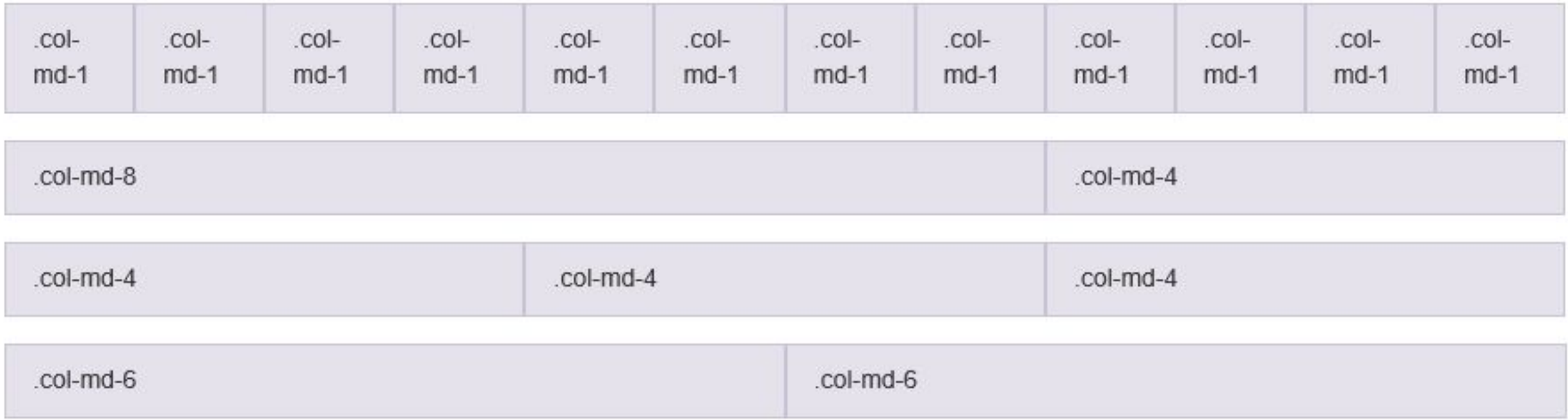
Bootstrap easily and efficiently scales your project with one code base, from phones to tablets to desktops.

Responsive Layout

Grid System

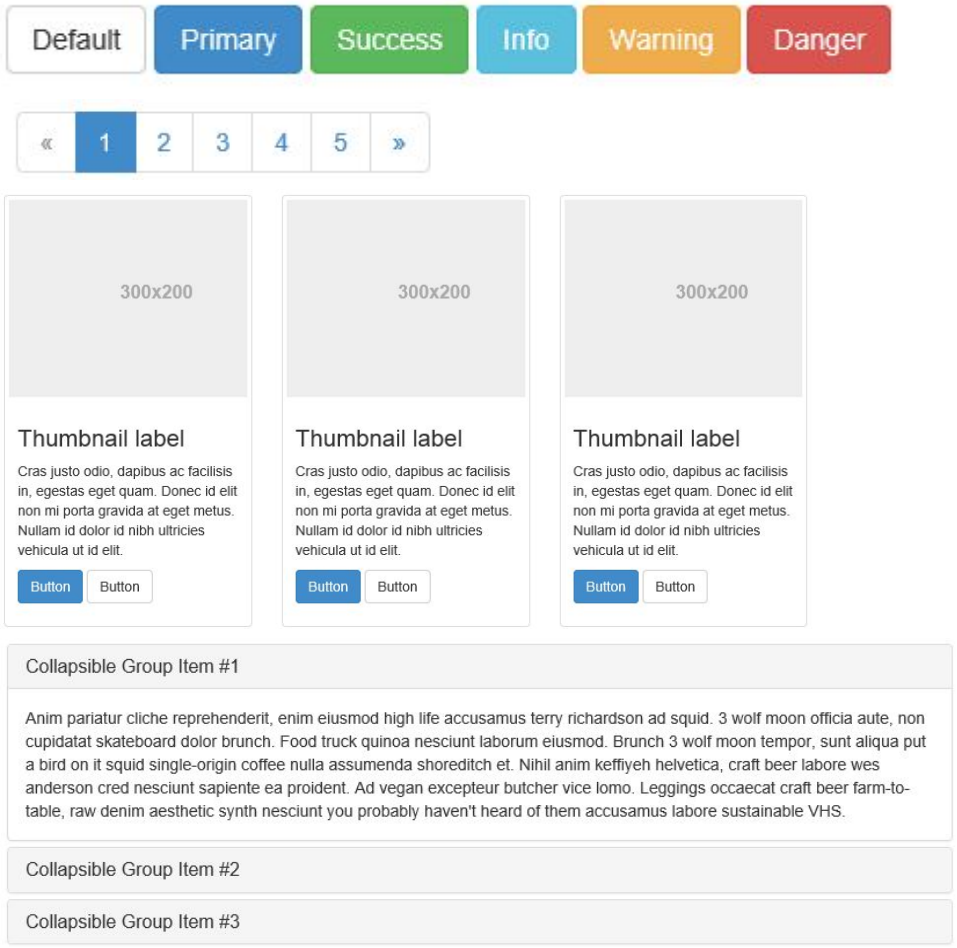
	Extra small devices Phones (<768px)	Small devices Tablets (≥768px)	Medium devices Desktops (≥992px)	Large devices Desktops (≥1200px)
Grid behavior	Horizontal at all times	Collapsed to start, horizontal above breakpoints		
Container width	None (auto)	750px	970px	1170px
Class prefix	.col-xs-	.col-sm-	.col-md-	.col-lg-
# of columns	12			
Column width	Auto	60px	78px	95px
Gutter width	30px (15px on each side of a column)			

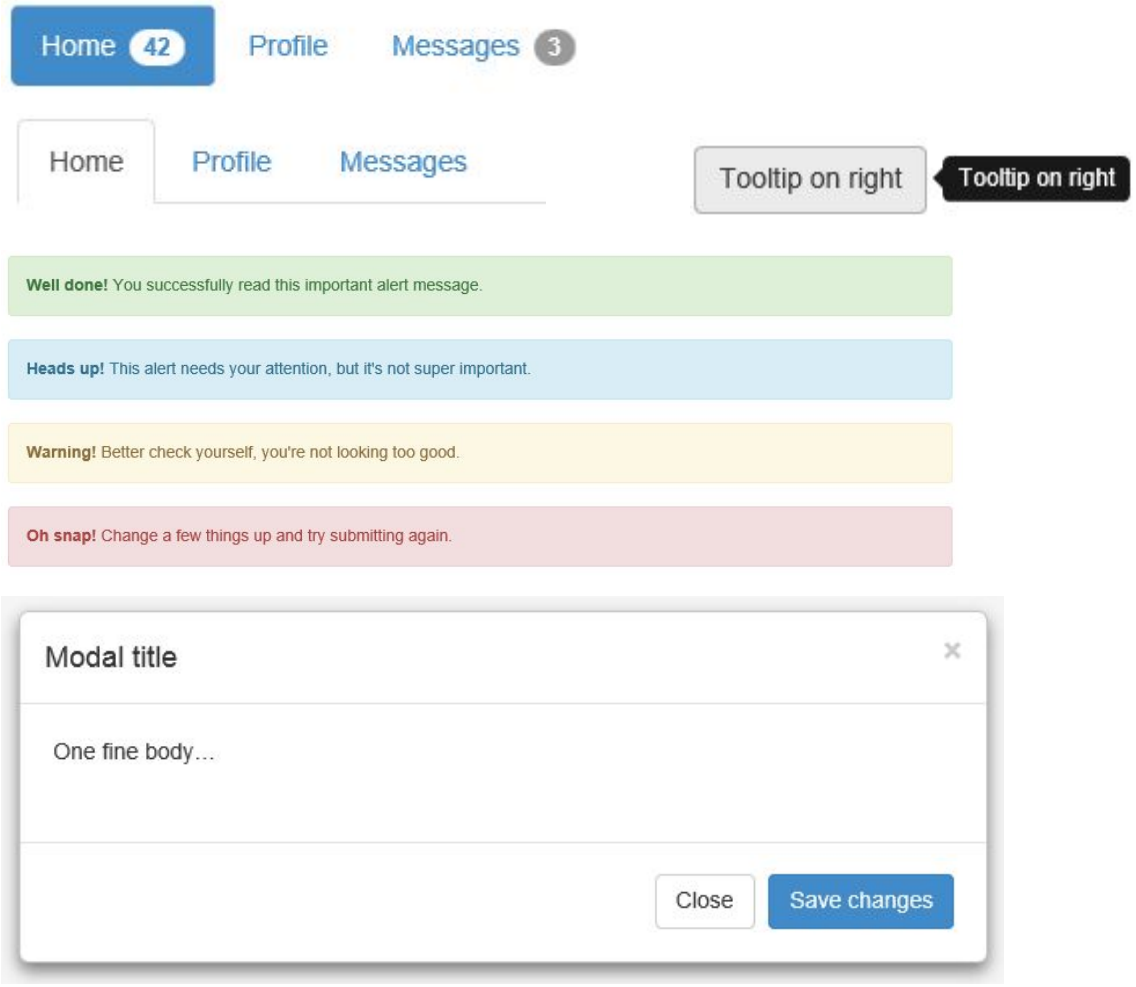
Grid System



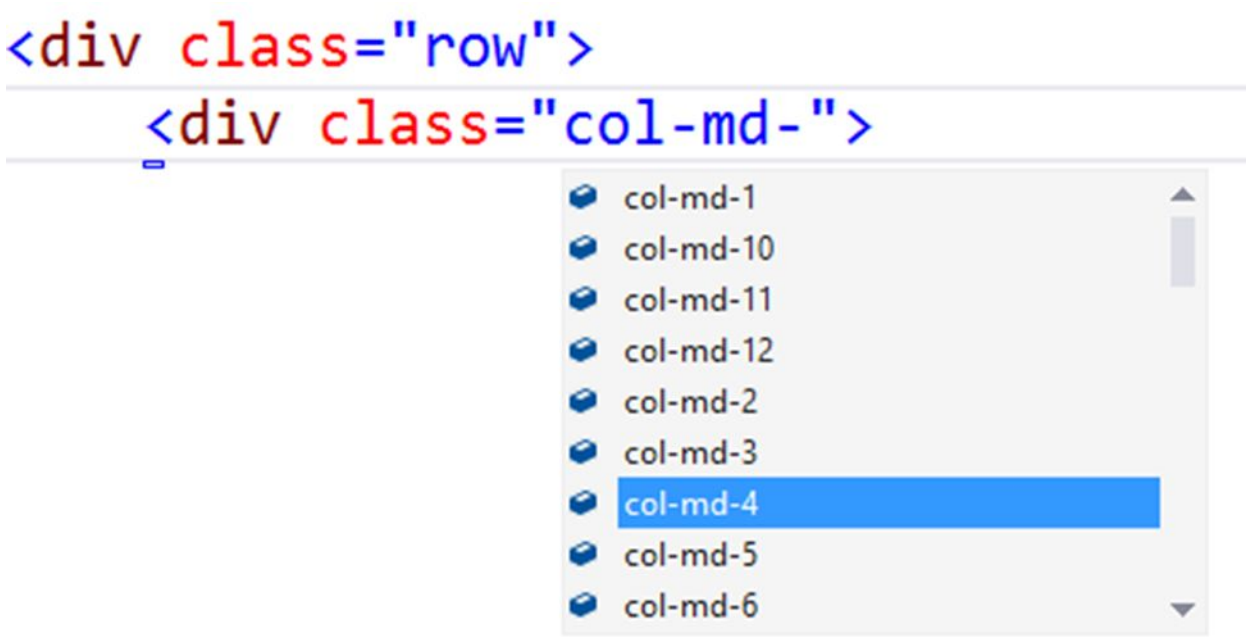
Components

- Reusable components built to provide navigation, alerts, progress bars, pagination and many more.

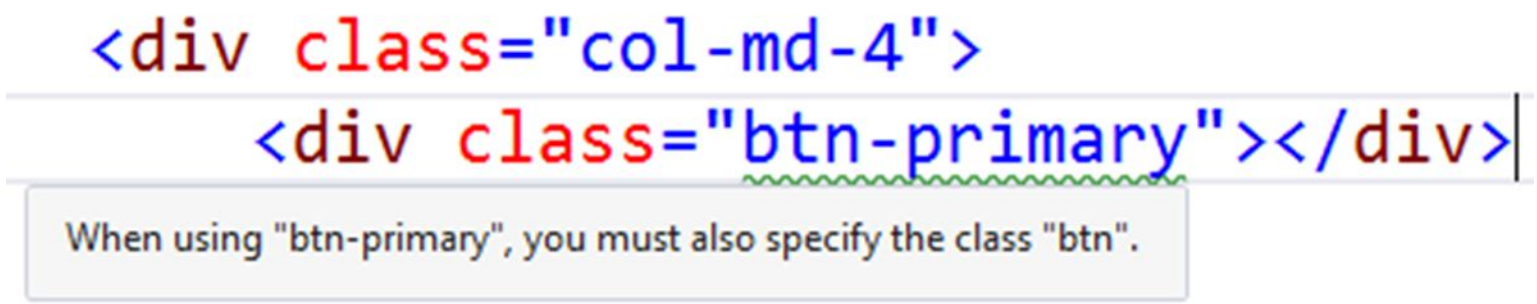




Visual Studio – Class Intellisense



Visual Studio – Missing Class Detection



Module 6: Client-Side Development

Section 2: Techniques

Lesson: JavaScript and jQuery

Using JavaScript in MVC 6

- JavaScript scripts can be defined inside a View using the html script tag like in a html page.
- Use the MVC @section tag to organize JavaScript scripts
    - The @RenderSection is used to inject JavaScript at a desired location inside the View.
- For best practices, declare JavaScript scripts inside a .js file.
- Use a minification tool in Visual Studio for optimization.
- IntelliSence support for JavaScript in Visual Studio

jQuery and Microsoft

- Lightweight open source JavaScript library
- Deprecated Microsoft.Ajax libraries in favor of jQuery.
- Distributed jQuery library with Visual Studio projects since 2008.
- Extended Microsoft product support for jQuery
    - Enterprises can open jQuery support cases 24x7 with Microsoft Support.
- Integrated Client template support
- Default templates leverage jQuery

jQuery

- Reduces client-side coding
- CSS 3-based syntax for traversing and manipulating DOM
- Concise wrappers for Ajax calls
- Abstracted to eliminate cross-browser differences
- Unobtrusive client validation
- XPath selectors to access elements in the DOM
- Elements are retrieved in the form of jQuery objects
- Start with jquery(), jquery., $() or $. to use jQuery

jQuery: Selectors

- Execute commands on a single or multiple selected DOM elements.
- Basic types of selectors:
  - Based on HTML elements IDs e.g.: $("#main")
  - Based on cascading style sheets (CSS) e.g.: $(".header")
  - Based on element tags e.g.: $("div")
  - Based on element attributes e.g.: $("[type = 'button']")
- Build more complex selectors through combination
  - $("#main p.quote") ⇔ Select paragraphs with a "quote" CSS class located inside elements with IDs equal to "main"
- $(this) operator

jQuery: Selectors (continue)

- Specific operators are used to expand selection options
  - A white space selects all elements that are descendants of the given ancestor e.g.: $(“div p”) ⇔ $(“div”).find(“p”)
  - The > operator selects direct child elements of the given ancestor e.g.: $(“div > p”) ⇔ $(“div”).children(“p”)
  - The + operator selects adjacent elements

    e.g.: $(“div + p”) ⇔ $(“div”).next(“p”)
  - The ~ operator selects all siblings elements

    e.g.: $(“div ~ p”) ⇔ $(“div”).nextall(“p”)
  - The comma operator selects all the specified elements

    e.g: $(div, p, a)

jQuery: Filters

- Used with Selectors, or alone
  - **Positional filters** :first, :even, :eq(index), :gt(index), :not (selector) etc.
  - **Child filters** :nth-child(expression), :first-child, :only-child
  - **Content filters** :contains(text), has(selector), :parent, :empty
  - **Form filter** :visible, :hidden, :button, :input, :selected
- Can be chained by appending with colon (:) for example:

jQuery: Methods

- Class/Style Methods
  - Use to apply CSS styles to the result of a selector
  - .addClass(), .css(), .height(), .position()
- DOM Methods
  - .before(), .insertBefore(), .append(), .empty(), .attr()

jQuery: Events

- jQuery simplifies events implementation
- Events are triggered by the page or end user's interaction
- Events are often used to attach a callback function
- To bind an event:
  - Use of function to bind a event directly e.g.: .click()
  - Use .on() e.g.: .on("click", ...)
  - Use .bind() e.g.: .bind("click", ...)
  - Use .live() e.g.: .live("click, ...)

Unobtrusive JavaScript

- Traditonal use of JavaScript
  - <input type="button" value="Click me" onclick="handleClick()" />
- For cleaner HTML page, remove inline JavaScript references.
- jQuery makes it easy to attach handlers to DOM elements

```
<script type="text/javascript">
        $(document).ready(function () {
                $(":button").bind("click", function () {
                        alert("Hello World!");
                });
        });
</script>
<div>
        <input type="button" value="Click me" />
<div>
```

Module 6: Client-Side Development

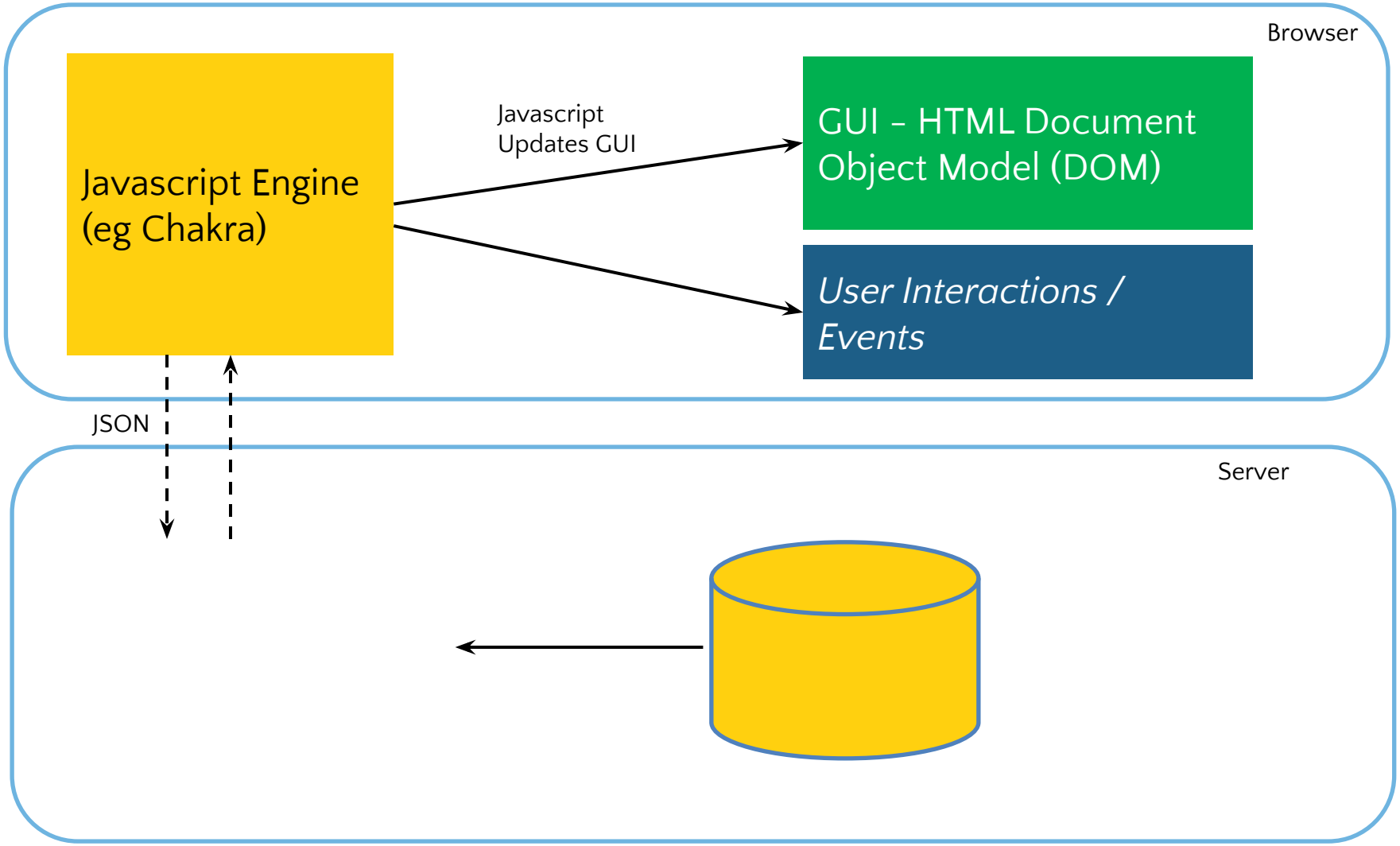
Section 2: Techniques

Lesson: AJAX

Ajax – (Asynchronous JavaScript and XML)

- Send and receive data from a server asynchronously (in the background) – better performance.
- Uses XMLHttpRequest object
- JSON typically used instead of XML
- Back button and bookmarking challenges
- Downgrade challenges for mobile devices that don't support JavaScript.
- Webcrawlers don't index pages created via Ajax
- Does not work across domains by default

AJAX Engine

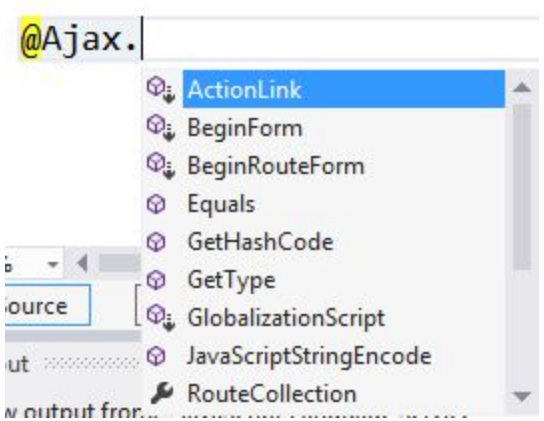

Browser
Javascript Engine
(eg Chakra)
Javascript
Updates GUI
GUI – HTML Document
Object Model (DOM)
User Interactions /
Events
JSON
Server

AJAX in Jquery

- Simplifies AJAX implementation in JavaScript.
- The full-feature .ajax() method Perform an asynchronous HTTP (Ajax) request.
- Shortcuts: .get(), .getScript(), .getJSON(), .post(), .load().
- $.ajaxSetup() method specifies settings for an Ajax call.

MVC's AJAX Helpers

- Methods providing and simplifying client-side functionality implementation in MVC.
  - Asynchronous forms and rendering links
- Methods and extensions are called using the Ajax property of the view.
- Use Request.IsAjaxRequest() for checking Ajax requests



Ajax in Actions – Get Request

- Either enable Get requests in code like this:

```
public ActionResult Details(int id)
{
    var employee = _repository.FindEmployee(id);
    if (Request.IsAjaxRequest())
    {
        return Json(employee,
        JsonRequestBehavior.AllowGet);
    }
    return View(employee);
}
```

Ajax in Actions

- Or create a new ActionMethodSelectorAttribute

```
[AttributeUsage(AttributeTargets.Class | AttributeTargets.Method)]
public class AcceptAjaxAttribute : ActionMethodSelectorAttribute
{
    public override bool IsValidForRequest(
    ControllerContext controllerContext, MethodInfo methodInfo)
    {
        return controllerContext.HttpContext
        .Request.IsAjaxRequest();
    }
}
```

```
[AcceptAjax]
public ActionResult Details(int id)
{
    var employee = _repository.FindEmployee(id);
    return View(employee);
}
```

Module 6: Client-Side Development

Section 3: Single Page Application

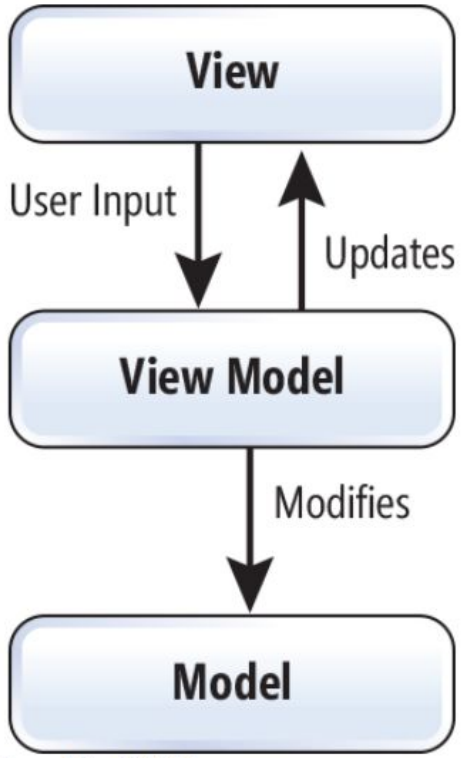
Lesson: Fundamentals

Single Page Application (SPA)

- In traditional web application, the server responds to new page requests.
- In SPA, the full page is loaded at initial request.
- Subsequent interactions take place through Ajax requests without full page reload.
- Better user experience
- Some SPA frameworks
  - AngularJS
  - KnockoutJS
  - Backbone
  - EmberJS
  - DurandalJS
  - Hot Towel

SPA – Design

- Model View View Model (MVVM) model
- Variance of MVC
- View faces the user
- Commands within the ViewModel are triggered by the View
- Changes in the ViewModel cause events that make the View update itself
- If a new View is required, the ViewModel communicates this with the application to redirect the user
- Model is the same



MVVM in general

- Main point in MVVM is the binding of data in the Model automatically to the View
  - MVVM is widely used with Silverlight and WPF projects (where data binding is “built-in”
  - When MVVM is used with ASP.NET, view-model resides in client side (JavaScript)
- Best choice for a “modern application”
  - No postbacks and less page loads
  - More asynchronous operations that don’t “freeze” the browser
  - Brings application look and feel to “web pages”

MVVM advantages

- Provides same separation benefits as MVC pattern
    - Separation of responsibilities between development teams
    - Separation of concerns between business logic and presentation
- Model is not directly exposed to client like MVC
- Works best with
    - Silverlight and WPF applications
    - Single-page web applications
    - Web applications with wizards or processes

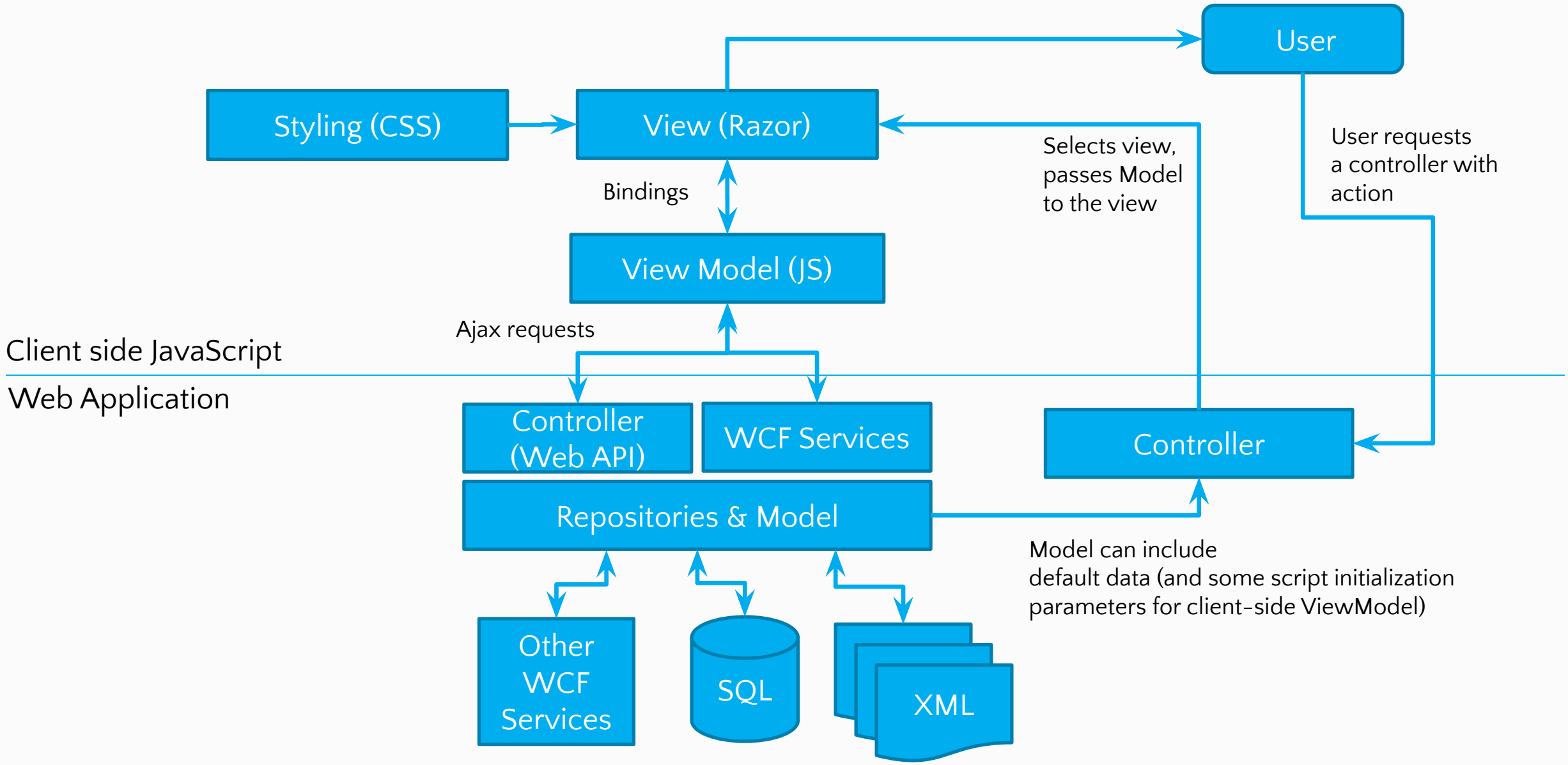
MVVM disadvantages

- Data-bindings are harder to debug
- ViewModel can be hard to design up front
    - But while using agile development methods this shouldn't be a problem
- With Large development teams, ViewModel can be easily "polluted" with duplicated code
    - Don't allow everyone in the team to make changes to ViewModel
- Data-binding to custom or 3rd party components can be complex or sometimes even impossible

Bringing MVVM to ASP.NET MVC

- You don't need to have multiple views per controller, changing the UI will be controlled by the ViewModel
    - One view can have several div-elements and their visibility is controlled by the ViewModel
- Think the server side as "initializer" and data provider
    - After 1st page load is completed, everything happens on client side
    - Except when UI needs updated data from server and executes and AJAX request
- You can use all ASP.NET MVC features the way you have used them before
- But you need to code a lot more in JavaScript

How it works


User
Styling (CSS)
View (Razor)
Selects view,
passes Model
to the view
User requests
a controller with
action
Bindings
View Model (JS)
Ajax requests
Client side JavaScript
Web Application
Controller
(Web API)
WCF Services
Controller
Repositories & Model
Model can include
default data (and some script initialization
parameters for client-side ViewModel)
Other
WCF
Services
SQL
XML

Using 3rd party MVVM-libraries

- There is no point in building scripts required for automatic binding from scratch
- ASP.NET MVC 6 is fully integrated with 3rd party libraries like:
    - knockoutJS (http://knckoutjs.com)
    - angularJS (https://angularjs.org/)
- Remember to use Bower and Grunt/Gulp to install 3rd party libraries

KnockoutJS example: Model to View

```
<div>
    <p>First name: <input data-bind="value:firstName, valueUpdate: 'keyup'" /></p>
    <p>Lirst name: <input data-bind="value:lastName, valueUpdate: 'keyup'" /></p>
    <p data-bind="text:fullName"></p>
</div>
<script type="text/javascript">
    var viewModel = function(){
        this.firstName= ko.observable('John');
        this.lastName = ko.observable('Doe');

        this.fullName = ko.computed(function(){
            return this.firstName() + ' ' + this.lastName();
        },this);
    };
    ko.applyBindings(new viewModel());
</script>
```

View Model

Model

KnockoutJS automatically binds (or "injects") the data into HTML

When this gets executed…

KnockoutJS example: View to Model

```
<div>
    <p>First name: <input data-bind="value:firstName, valueUpdate: 'keyup'" /></p>
    <p>Lirst name: <input data-bind="value:lastName, valueUpdate: 'keyup'" /></p>
    <p data-bind="text:fullName"></p>
</div>
<script type="text/javascript">
    var viewModel = function(){
        this.firstName= ko.observable('John');
        this.lastName = ko.observable('Doe');

        this.fullName = ko.computed(function(){
            return this.firstName() + ' ' + this.lastName();
        },this);
    };
    ko.applyBindings(new viewModel());
</script>
```

View Model

Model

KnockoutJS automatically binds
(or ”updates”) the data back into the model

AngularJS example: Model to View

```
<div ng-app="myapp">
    <div ng-controller="MyController">
        <p>First name: <input ng-model="viewModel.firstName"></p>
        <p>Last name: <input ng-model="viewModel.lastName"></p>
        <p>{{viewModel.firstName}} {{viewModel.lastName}}</P>
    </div>
    <script>
    angular.module("myapp", [])
            .controller("MyController", function($scope) {
                $scope.viewModel = {};
                $scope.viewModel.firstName = "John";
                $scope.viewModel.lastName  = "Doe";
            } );
    </script>
</div>
```

View Model

Model

AngularJS automatically binds (or ”injects”) the data into HTML

First contact with Angular.js

MV* framework

For **S**ingle **P**age **A**pplication

You need only one script file:

```
<script src="angular.min.js"></script>
```

Latest version here:

http://code.angularjs.org/

New constructs

- Data binding, as in {{}}
- DOM control structures for repeating/hiding DOM fragments
- Support for forms and form validation
- Attaching code-behind to DOM elements
- Grouping of HTML into reusable components

Concepts

Concept	Description
Template	HTML with additional markup
Directives	extend HTML with custom attributes and elements
Model	the data shown to the user in the view and with which the user interacts
Scope	context where the model is stored so that controllers, directives and expressions can access it
Expressions	access variables and functions from the scope
Compiler	parses the template and instantiates directives and expressions
Filter	formats the value of an expression for display to the user
View	what the user sees (the DOM)
Data Binding	sync data between the model and the view
Controller	the business logic behind views
Dependency Injection	Creates and wires objects and functions
Injector	dependency injection container
Module	a container for the different parts of an app including controllers, services, filters, directives which configures the Injector
Service	reusable business logic independent of views

Controllers

What is a Controller?

Controllers

index.html

```
<body ng-controller="cardsListController">
    <div id="cardsList">
        <div ng-repeat="card in cards" class="cardItem">
            Card {{card.id}}
        </div>
    </div>
</body>
```

cardsController.js

```
var cardsControllers = angular.module('cardsControllers', []);

cardsControllers.controller('cardsListController', ['$scope', function ($scope) {
    var cards = [{ id: 10 }, { id: 11 }];

    $scope.cards = cards;
}]);
```

Controllers

What shouldn’t a controller do?

Magic the Gathering

1. Open **CA.1**
2. Implement the **cardsListController**
3. It should set **$scope.cards** to an array of JS objects with an **id** property. **(see index.html)**

Modules

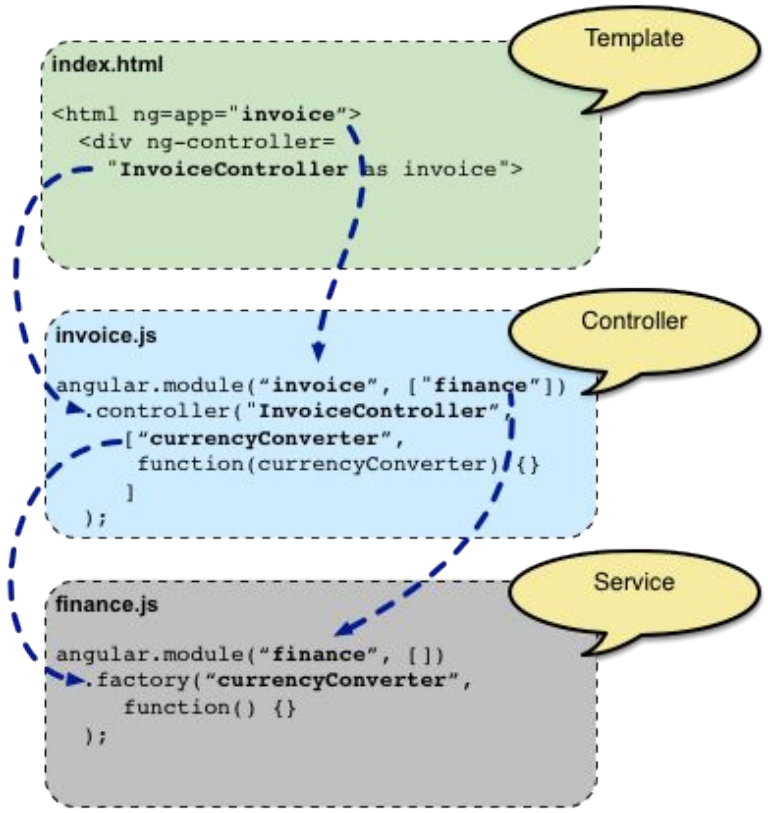
What is a Module?

You can think of a module as a container for the different parts of your app – controllers, services, filters, directives, etc.

Modules

```
var app = angular.module('workshop', []);

app.controller('workshopController', workshopController);

function workshopController($scope) {
    $scope.name = "Building Single Page Applications with ASP.NET and Angular.js";
    $scope.duration = "9:00 AM - 5:00 PM Tuesday and Wednesday.";

    $scope.getDescription = function () {
        return $scope.name + " - " + $scope.duration;
    }

    Object.defineProperty($scope, 'description', {
        get: function () { return this.getDescription(); }
    });
}
```

Modules



Modules

```
angular.module('myModule', []).
config(function (injectables) { // provider-injector
    // This is an example of config block.
    // You can have as many of these as you want.
    // You can only inject Providers (not instances)
    // into config blocks.
}).
run(function (injectables) { // instance-injector
    // This is an example of a run block.
    // You can have as many of these as you want.
    // You can only inject instances (not Providers)
    // into run blocks
});
```

Dependency Injection

What is Dependency Injection?

How can a component get ahold of it's dependencies?

Dependency Injection

It can...

- create the dependency, typically using the new operator.
- look up the dependency, by referring to a global variable.
- have the dependency passed to it where it is needed.

Dependency Injection

How does the injector locate dependencies?

Implicit location

Dependency Injection

Assume the parameter names are names of services

```
cardsListController = function($scope, shuffler) {
    var cards = [{ id: 10 }, { id: 11 }];

    $scope.cards = cards;
    $scope.shuffle = function() { shuffler.shuffle(); }
};
```

Q: Problems with this method?

Property Annotation

Dependency Injection

The $inject property is an array of services to inject.

```
(function () {
    'use strict';

    angular
        .module('app')
        .controller('cardsController', cardsController);

    cardsController.$inject = ['$http', '$scope'];

    function cardsController($http, $scope) {

    }
})();
```

Q: Thoughts?

Inline Array Annotation

Dependency Injection

Define services inline when creating the controller.

```
angular
    .module('app')
    .controller('cardsController', ['$http', '$scope', function($http, $scope) {

        // implementation

    }]);
```

Q: Thoughts?

Magic the Gathering

1. Open **CA. 2**
2. In **cardsController.js**, create a **cardsController** module. Add **cardsListController** to it.
3. In **app.js**, create a **mainApp** module. Add **cardsController** as a dependency.

angular.js

angular-resource.js

Getting data

$http

$resources

Filters

Promises

What is a promise?

Filters

Name	Description
filter	Selects a subset of items from `array` and returns it as a new array.
currency	Formats a number as a currency (ie $1,234.56). When no currency symbol is provided, default symbol for current locale is used.
number	Formats a number as text.
date	Formats `date` to a string based on the requested `format`.
json	Allows you to convert a JavaScript object into JSON string.
lowercase	Converts string to lowercase.
uppercase	Converts string to uppercase.
limitTo	Creates a new array or string containing only a specified number of elements. The elements are taken from either the beginning or the end of the source array, string or number, as specified by the value and sign (positive or negative) of `limit`. If a number is used as input, it is converted to a string.
orderBy	Orders a specified `array` by the `expression` predicate. It is ordered alphabetically for strings and numerically for numbers. Note: if you notice numbers are not being sorted correctly, make sure they are actually being saved as numbers and not strings.

Lab Work

Hands On Lab: Build a Single Page Application (SPA) with ASP.NET Web API and Angular.js

Magic the Gathering

1. Open **CA.3**
2. Modify **cardsListController** to read the data from **all.json**.
3. Push each card from **all.json** onto the **$scope.cards** array, so it's available in the view.

angular.js

angular-resource.js

angular-animate.js

angular-route.js

Routing and views

SPA uses views to build UI

Angular.js uses routing to define views

Animations

Module Components

angular.js

angular-route.js

Routing

ngView

A directive that complements the $route service by
including the rendered template of the current route
into the main layout

```
<body ng-app="mainApp" ng-controller="cardsListController">
    <div ng-view id="mainView"></div>
</body>
```

```
<section>
    <img id="bigImage" ng-src="{{'http://www.urzagatherer.com/cards/' + card.picturePath}}" />
    <div id="detailsPane">
        <div class="rowHeader">Name:</div>
        <div class="rowData">{{card.nameEn}}</div>
        <div class="rowHeader">Type:</div>
        <div class="rowData">{{card.typeEn}}</div>
    </div>
</section>
```

Module Components

angular.js

angular-route.js

Routing

Route Provider

$routeProvider is used for configuring routes.

```
mainApp.config([
    '$routeProvider',
    function ($routeProvider) {
        $routeProvider.
            when('/cards', {
                templateUrl: 'Views/list.html',
                controller: 'cardsListController'
            }).
            otherwise({
                redirectTo: '/cards'
            });
    }
]);
```

Module Components

angular.js

angular-route.js

Routing

Route Service

$route is used for deep-linking URLs to controllers and views.

```
angular.module('ngRouteExample', ['ngRoute'])

 .controller('MainController', function ($scope, $route, $routeParams, $location) {
     $scope.$route = $route;
     $scope.$location = $location;
     $scope.$routeParams = $routeParams;
 })
```

```
<div>
    <pre>$location.path() = {{$location.path()}}</pre>
    <pre>$route.current.templateUrl = {{$route.current.templateUrl}}</pre>
    <pre>$route.current.params = {{$route.current.params}}</pre>
    <pre>$route.current.scope.name = {{$route.current.scope.name}}</pre>
    <pre>$routeParams = {{$routeParams}}</pre>
</div>
```

Module Components

angular.js

angular-route.js

Routing

Route Parameters

routeParams

```
cardsControllers.controller('cardDetailController', function ($scope, $routeParams) {
    var cards = $scope.$parent.cards;
    for (var index = 0; index < cards.length; index++) {
        if (cards[index].id == $routeParams.cardId) {
            $scope.card = cards[index];
            break;
        }
    }
});
```

ASP.NET + Angular Routing

HTML from view – no problem!

HTML file – use an IIS Rewrite rule

/cards.html -> /cards

Magic the Gathering

1. Open **CA.4**
2. Add a filter to **app.js** called **filterAndReduce.** Look at **index.html** to see the parameters it requires.
3. The query parameter is optional, and it filters cards by the **card.nameEn** property.
4. The count property is always used.

Magic the Gathering

1. Open **CA.5**
2. Create a route that responds to the URI **/cards** and displays **views/list.html.** The **cardsListController** should be executed.
3. Create a route that responds to the URI **/cards/10** and displays **views/detail.html**.
   The **cardDetailController** should be executed.
4. Create a catchall route that redirects to **/cards**

Productivity Tools

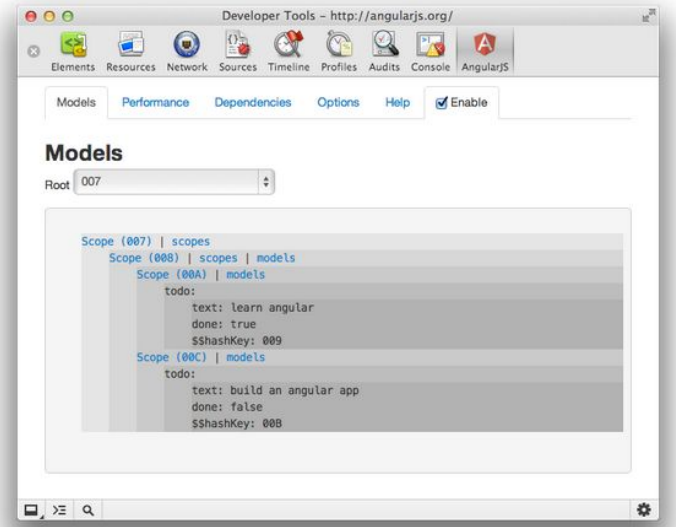
SideWaffle

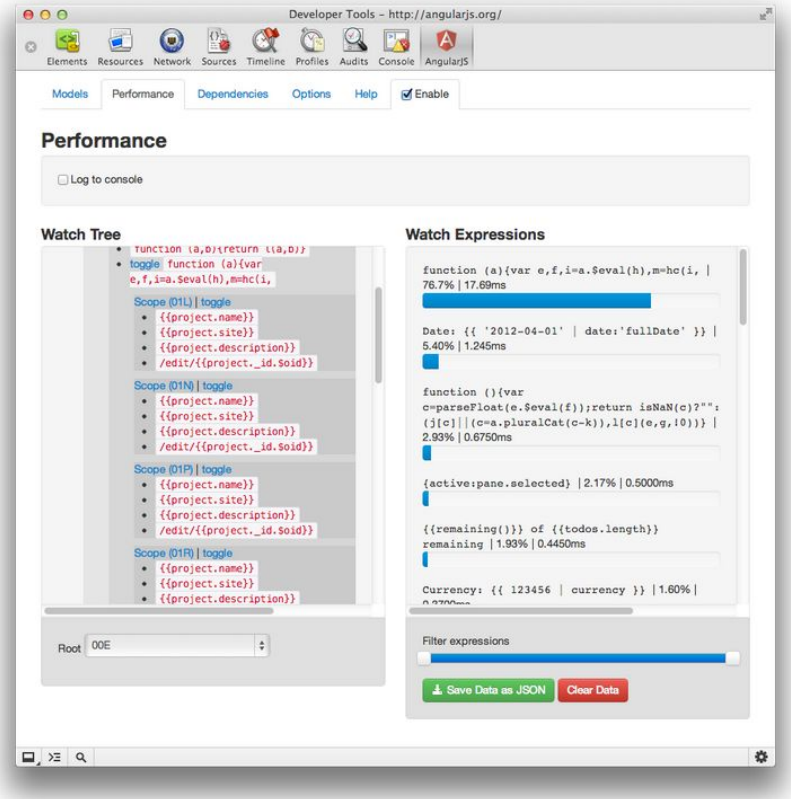
Templates for Visual Studio 2012/2013

http://www.sidewaffle.com

Batarang – Chrome Extension

Productivity Tools





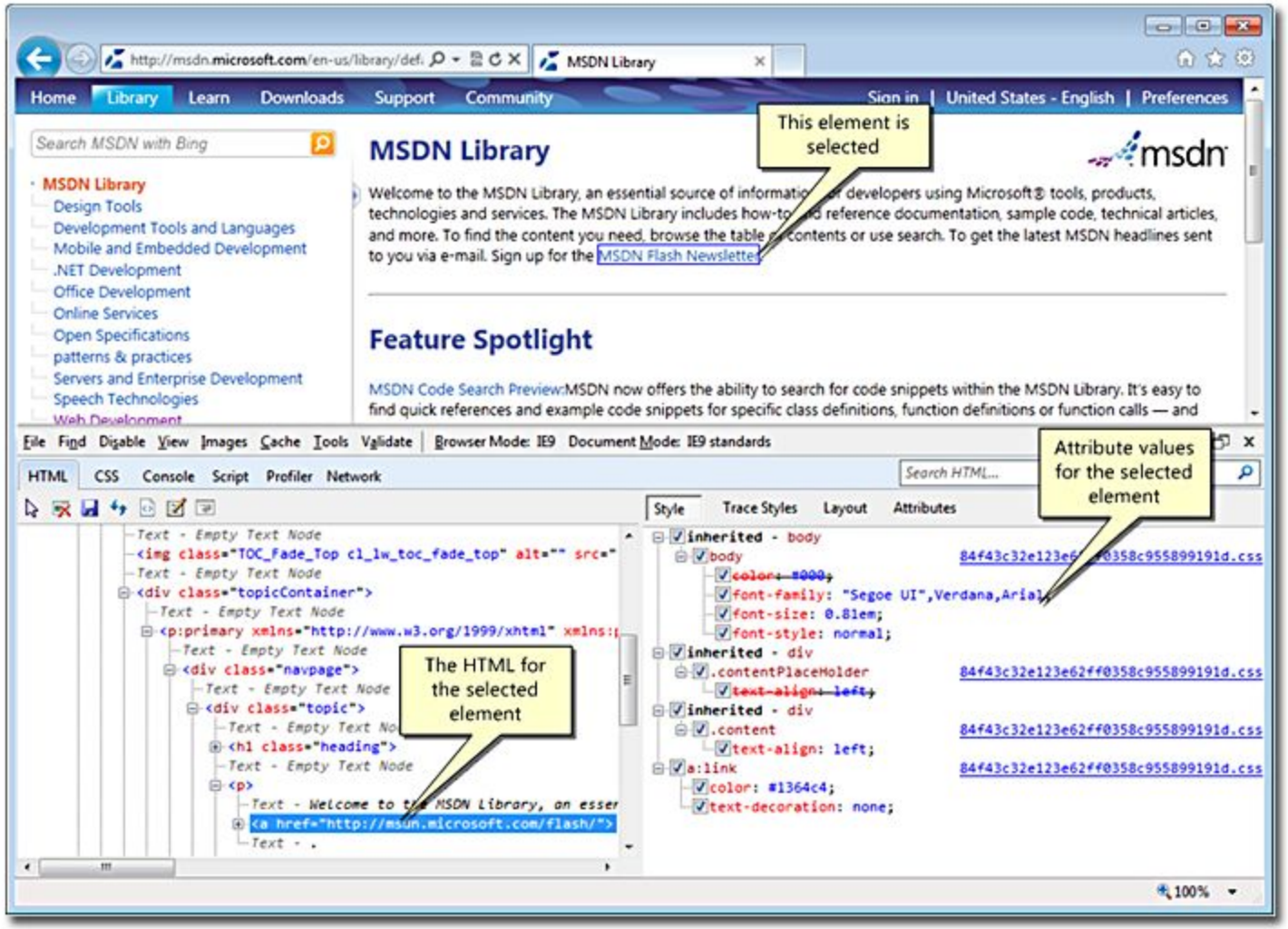
https://github.com/angular/angularjs-batarang

Module 6: Client-Side Development

Section 1: Development Tools

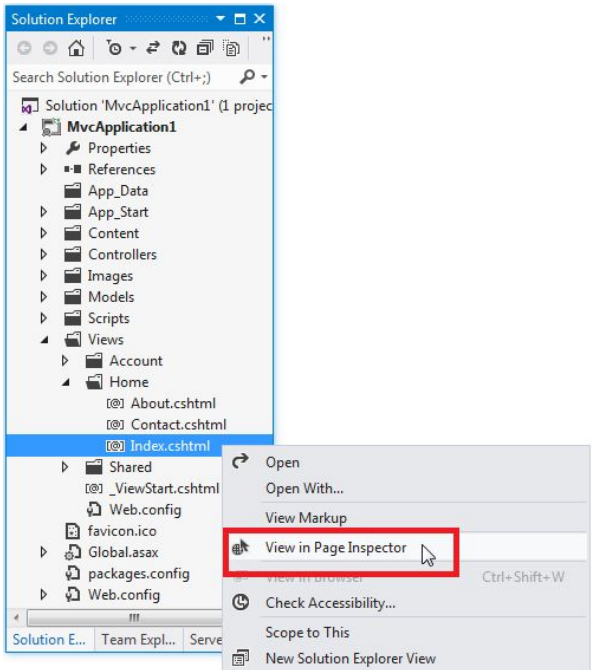
Lesson: Tools

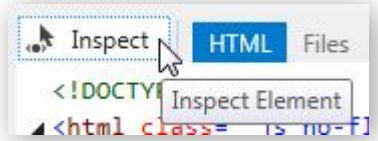
jQuery – Tools / F12

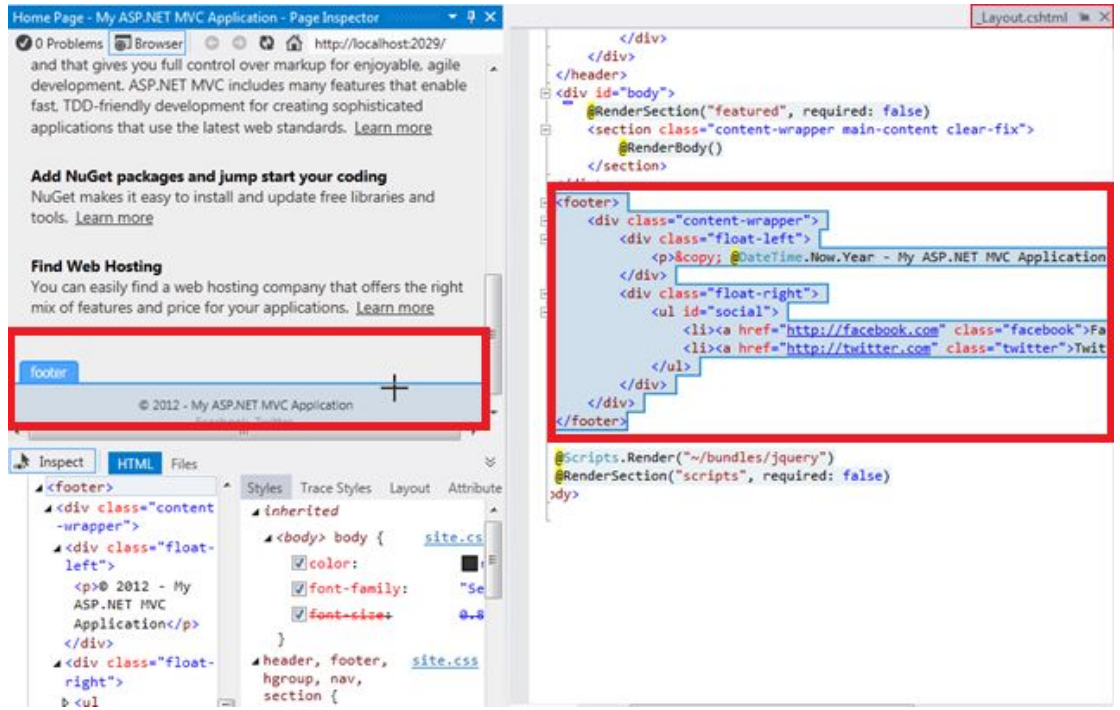


jQuery – in Visual Studio

- Page Inspector







Module Summary

MVC 6 support for client-side through default project template

Bootstrap for UI styling

JavaScript, jQuery and AJAX for scripting

Single Page Application fundamentals and frameworks

Lab: Client-Side Development


Microsoft