Client-Side Functionality

Dan Wahlin

pluralsight
hardcore developer training

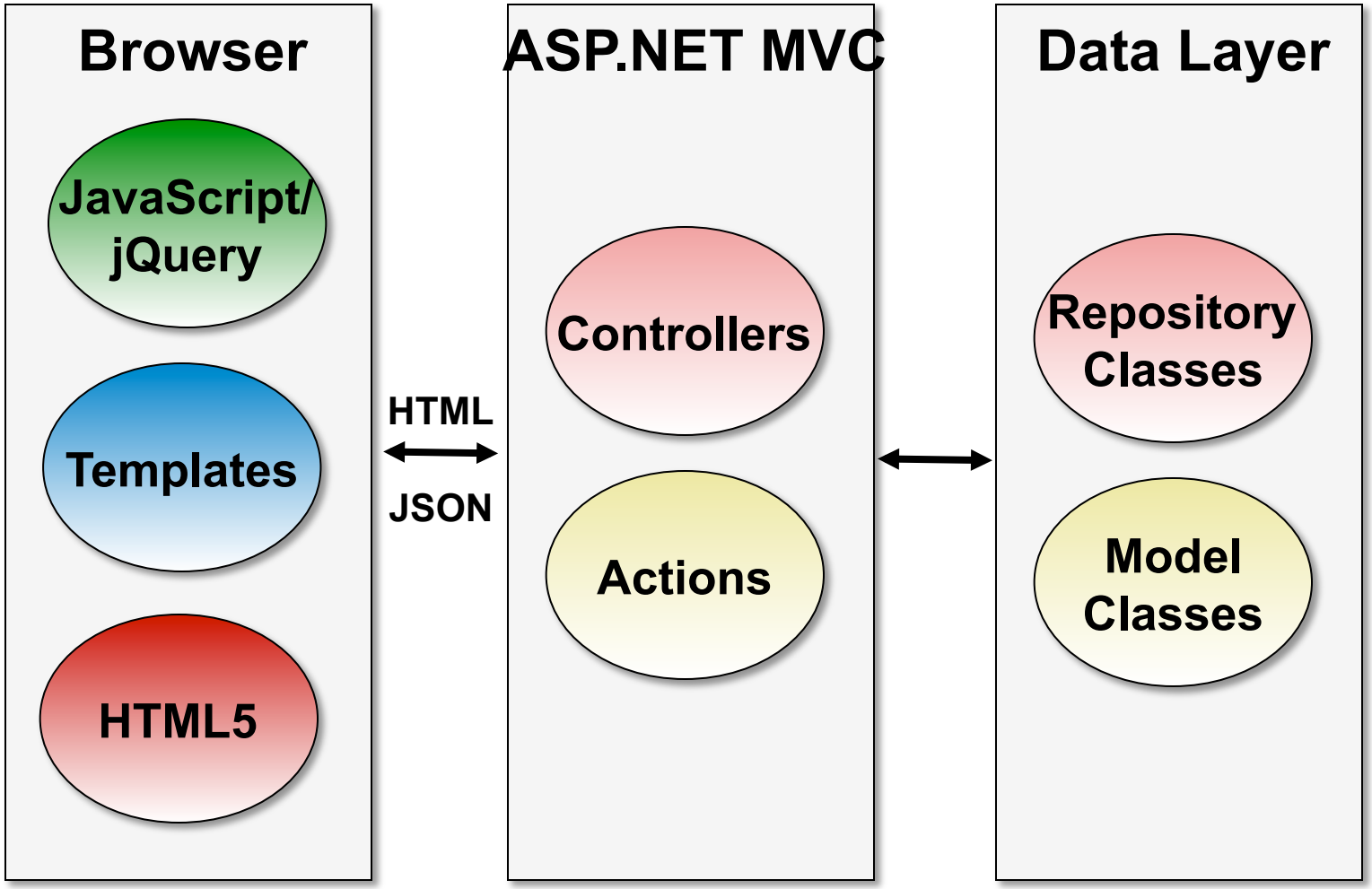
Agenda

- Client-Side Technology Overview
- Defining Tile Layout using Object Literals
- Retrieving JSON Data using jQuery Ajax Functions
- Rendering JSON Data using jQuery Templates
- Charting Data with Canvas and SVG
- Integrating HTML5 Video

Agenda

- Client-Side Technology Overview
- Defining Tile Layout using Object Literals
- Retrieving JSON Data using jQuery Ajax Functions
- Rendering JSON Data using jQuery Templates
- Charting Data with Canvas and SVG
- Integrating HTML5 Video

Client-Side Technology Overview



Client-Side Technologies

Client-Side Technologies
HTML5
Modernizr
HTML5 Boilerplate
jQuery
Canvas
SVG
CSS3
JavaScript Patterns

Client-Side Scripts

Script	Description
scene.startup.js	Start-up logic and animations
scene.layoutservice.js	Defines tile "scenes"
scene.statemanager.js	Creates tiles dynamically at runtime
scene.dataservice.js	Handles performing AJAX calls to the server
scene.tile.binder.js	Handles converting JSON data into HTML for each tile's size (small, medium and large)
scene.tile.renderer.js	Renders different tile sizes based on position
scene.tile.formatter.js	Custom formatting functionality for tiles

Scripts use the Revealing Module Pattern

```
var calculator = function() {
   var eqCtl = document.getElementById('eq'),
       doAdd = function(x,y) {
          var val = x + y;
          eqCtl.innerHTML = val;
       };

    return {
        add: doAdd
    };
}();
```

```
calculator.add(2,2);
```

Agenda

- Client-Side Technology Overview
- Defining Tile Layout using Object Literals
- Retrieving JSON Data using jQuery Ajax Functions
- Rendering JSON Data using jQuery Templates
- Charting Data with Canvas and SVG
- Integrating HTML5 Video

Tile Layout

- **Tiles are arranged using JavaScript Object Literals**

- **Each tile has 2 scenes defined:**
  1. Grid layout scene
  2. Cloud layout scene

- **Additional details about the border color, z-index, and opacity are defined with each scene**

- **A tile can have a custom formatter that's called after the tile is rendered**

Tiles in the Grid Layout Scene

Account at a Glance

ACCT#
Z48573988

QUOTES

Symbol msft Quote

Microsoft Corporation

25.99 -0.64 -2.4%

Open	Day Low	Day High
25.82	25.74	26.18

VIDEO NEWS

DOW 11,685.05

-297.74 -2.49%

S&P 500 1,221.11

-35.1 -2.8%

NASDAQ 2,613.02

-77.63 -2.9%

SECTOR SUMMARY

ACCOUNT DETAILS - JOINT BROKERAGE

Cash	$10,000.00
Available Margin	$414,888.67
Positions	$1,193,887.00

Total: $1,203,887.00

ACCOUNT POSITIONS

Symbol	Shares	Last	Change	% Chg	Open	Total
C	700	29.17	-2.42	-7.66	4.46	$20,419
DIS	1200	33.6	-1.28	-3.67	42.66	$40,320
EBAY	1300	30.89	-0.94	-2.95	31.38	$40,157
FDGFX	5100	25.66	-0.95	-3.57		$130,866
IBM	2200	181.35	-3.28	-1.78	163.81	$398,970
JPM	2400	32.71	-2.05	-5.9	46.17	$78,504
LMT	2700	73.99	-1.91	-2.52	80.55	$199,773
PG	3400	62.71	-1.28	-2	62.02	$213,214
YHOO	4800	14.93	-0.71	-4.54	16.81	$71,664

Total: $1,193,887

News

TOP Oil Ma

Ticker 3.56 -0.65 LLY 37.22 +0.06 LMT 73.99 -1.91 MSFT 25.99 -0.64 MTH 17.21 -0.54 NKE 93.88 -2.47 NOC 54.98 -2.77 NOK 6.31 -0.42 ORCL 31.69 -1.08 F

Tiles in the Cloud Layout Scene

So What's a JavaScript Object Literal?

```
Start → {
            Name: 'John Doe',
            Age: 35,
            Address: {  ← Start
                City: 'Phoenix',
                State: 'AZ',
            }  ← End
End → }
```

Arranging Tiles using Object Literals

```
tiles:  [
            {  name: 'Account Details',
               tileId: 'AccountDetails',
               formatter: tileFormatter.formatAccountDetails,
               scenes: [
                    { height: s1Mh, width: s1Mw, top: 0, left: 0,
                      opacity: 1, size: 1, borderColor: '#5E1B6B', z: 0 },
                    { height: 90, width: 210, top: 80, left: 250, size: 0,
                      borderColor: '#5E1B6B', z: '2000', opacity: .5 }
               ]
            },
            …
        ]
```

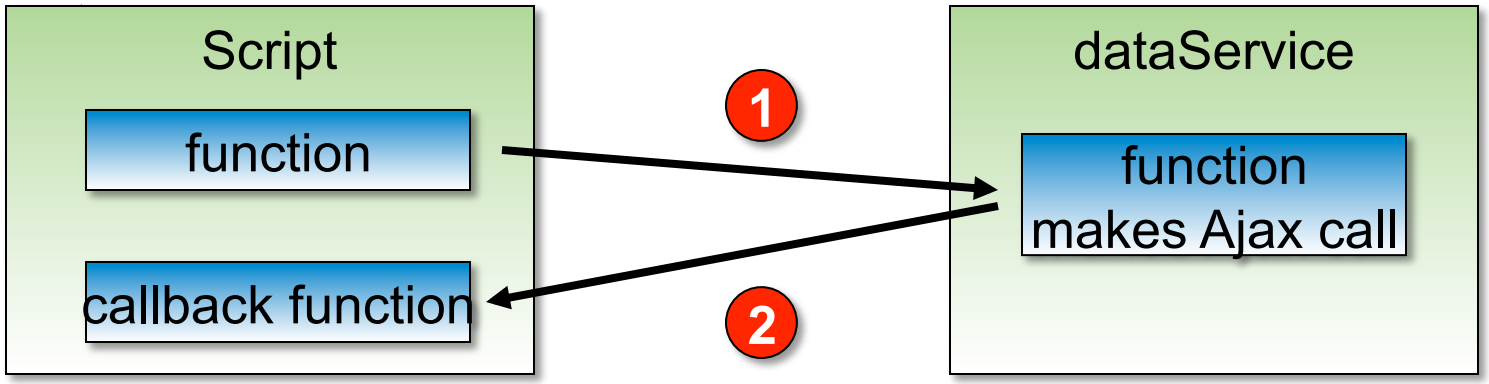
scene.layoutservice.js

Agenda

- Client-Side Technology Overview
- Defining Tile Layout using Object Literals
- Retrieving JSON Data using jQuery Ajax Functions
- Rendering JSON Data using jQuery Templates
- Charting Data with Canvas and SVG
- Integrating HTML5 Video

Retrieving JSON Data

- JSON data served up by DataServiceController
- scene.dataservice.js responsible for making Ajax calls
- dataService functions accept parameters as well as a callback function to invoke once JSON data



Using jQuery to Make Ajax Calls

```
var dataService = new function () {
    var serviceBase = '/DataService/',
    getAccount = function(acctNumber, callback) {
      $.getJSON(serviceBase + 'GetAccount',
                {acctNumber:acctNumber}, function(data) {
                callback(data);
      });
    };
    return {
       getAccount: getAccount
    };
}();
```

Callback Function

Invoke Callback Function

scene.dataservice.js

Using the dataService Object

```
var sceneStateManager = new function () {
   renderTiles = function(acctNumber) {
     dataService.getAccount(acctNumber, renderAccountTiles);
     dataService.getMarketIndexes(renderMarketTiles);
     renderDefaultTiles();
   },
   renderAccountTiles = function(data) {
      $('div.tile[id^="Account"]').each(function() {
         var tileDiv = $(this);
         renderTile(data, tileDiv, 0);
      });
   },
   ...
}();
```

Callback

scene.statemanager.js

Using the dataService Object

```
var sceneStateManager = new function () {
   renderTiles = function(acctNumber) {
     dataService.getAccount(acctNumber, renderAccountTiles);
     dataService.getMarketIndexes(renderMarketTiles);
     renderDefaultTiles();
   },
   renderMarketTiles = function(data) {
        renderTile(data, $('#DOW'), 0);
        renderTile(data, $('#NASDAQ'), 0);
        renderTile(data, $('#SP500'), 0);
   },
   ...
}();
```

Callback

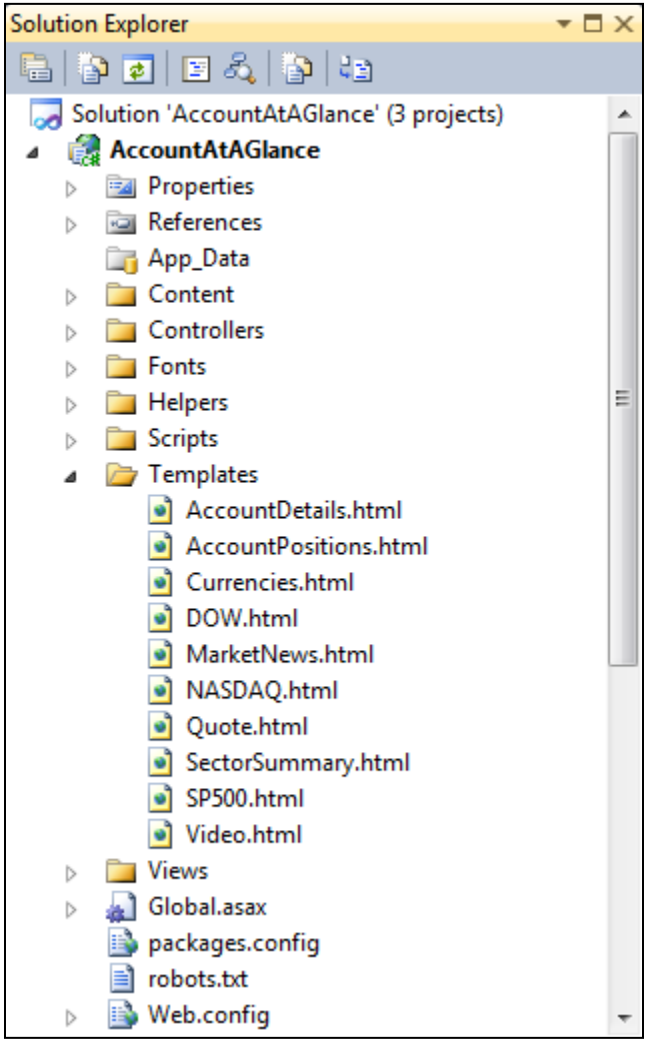
scene.statemanager.js

Agenda

- Client-Side Technology Overview
- Defining Tile Layout using Object Literals
- Retrieving JSON Data using jQuery Ajax Functions
- Rendering JSON Data using jQuery Templates
- Charting Data with Canvas and SVG
- Integrating HTML5 Video

Rendering HTML Tiles

- Application tiles use HTML5 semantic tags
- Tile templates stored on the server and retrieved dynamically by the client
- Tiles rendered using jQuery Templates

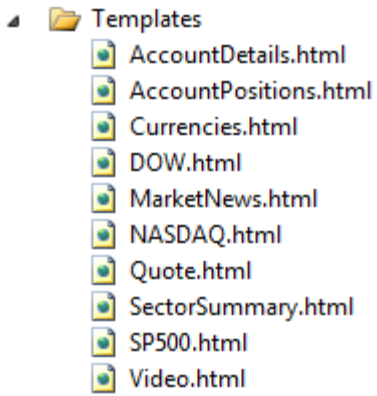


Tile Template Example

```
<script id="AccountDetailsTemplate_Small"
   type="text/x-jquery-tmpl">
     <div class="content">
        <header>
            <span>ACCOUNT TOTAL</span>
        </header>
        <section>
            <span class="Currency Positive"
                 style="font-size: 20pt">${Total}</span>
        </section>
    </div>
</script>
```

Retrieving HTML Templates

```
var templateBase = '/Templates/',
bind = function (tileDiv, data, renderer) {
    var tileName = tileDiv.attr('id');
    $.get(templateBase + tileName + '.html',
      function (templates) {
        $('body').append(templates);
        var acctTemplates = [
            tmpl(tileName, 'Small', data),
            tmpl(tileName, 'Medium', data),
            tmpl(tileName, 'Large', data)
        ];
        tileDiv.data().templates = acctTemplates;
        tileDiv.data().tileData = data;
        renderer(tileDiv);
    });
}
```



scene.tile.binder.js

Rendering Templates

```
tmpl = function (tileName, size, data) {
   var template = $('#' + tileName + 'Template_' + size);
   if (data != null)
       return template.tmpl(data);
   else
       return template.html();
 }
```

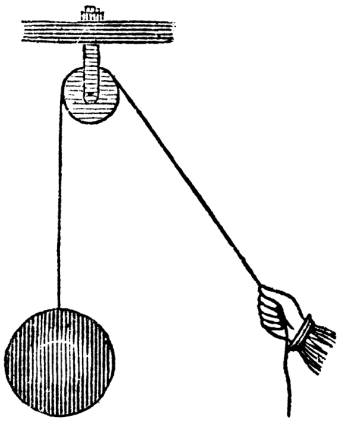
scene.tile.binder.js

Agenda

- Client-Side Technology Overview
- Defining Tile Layout using Object Literals
- Retrieving JSON Data using jQuery Ajax Functions
- Rendering JSON Data using jQuery Templates
- Charting Data with Canvas and SVG
- Integrating HTML5 Video

HTML5 Canvas

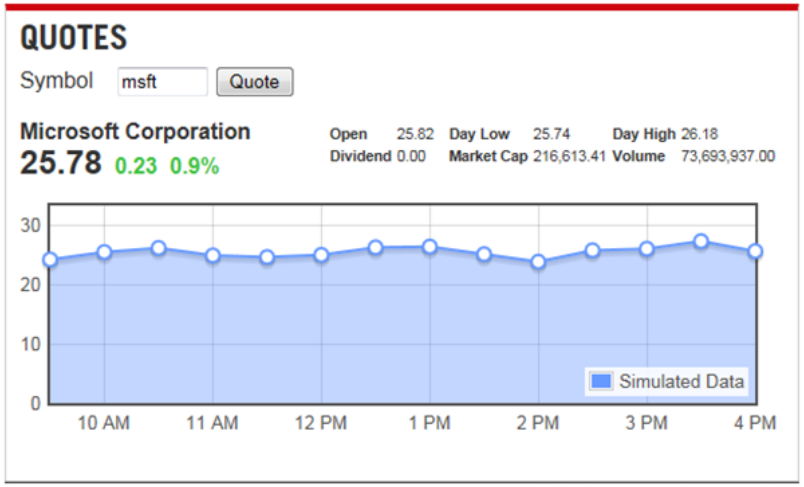
- **HTML5 canvas provides a way to render pixels using JavaScript functions**
- **Plugins can simplify working with the canvas:**
  - EaselJS
  - Fabric
  - Flot
  - Gury
  - JCanvaScript



Canvas in Account at a Glance

- **Account at a Glance uses Flot to render stock quote graphs using the canvas:**

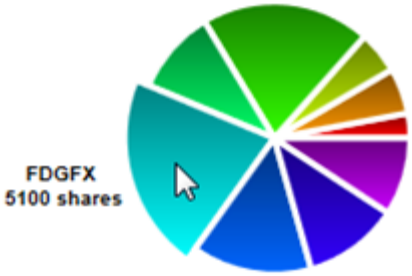
```
$.plot(canvasDiv, [{
        color: color,
        shadowSize: 4,
        label: 'Simulated Data',
        data: quoteData
    }], chartOptions);
```



Scalable Vector Graphics

- SVG provides a way to render vector graphics using script or elements
- Application uses the Raphael SVG plugin:

```
raphael('AccountPositionsSVG', 500, 420)
  .pieChart(scene.width / 2,
            scene.height / 4 + 10,
            66, values, labels, "#fff");
```



Agenda

- Client-Side Technology Overview
- Defining Tile Layout using Object Literals
- Retrieving JSON Data using jQuery Ajax Functions
- Rendering JSON Data using jQuery Templates
- Charting Data with Canvas and SVG
- Integrating HTML5 Video

Displaying Video

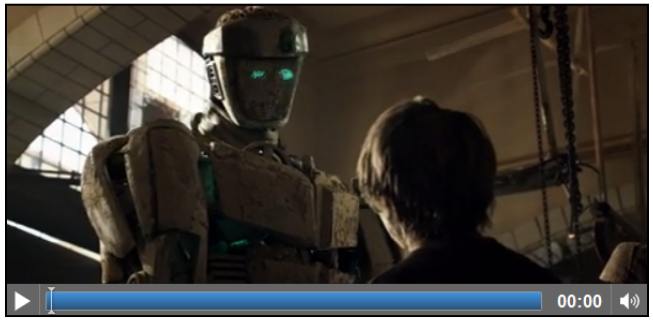
- **HTML5 video element used to display video:**

```
<video id="VideoPlayer" controls preload="auto"
       poster="/content/images/video-poster.jpg">
    <source type="video/mp4"
      src="http://.../markets_320k.mp4" />
</video>
```

Multiple Video Sources

- **Video element supports multiple sources**
- **Free conversion tool:** http://www.mirovideoconverter.com

```
<video id="video" height="323" width="600"
    controls poster="Images/poster.jpg">
    <source src="Video/test.webm" />
    <source src="Video/test.ogv" />
    <source src="Video/test.mp4"  />
 </video>
```



Summary

- JavaScript Object Literals provide a convenient way to encapsulate data
- Ajax calls can be placed in a centralized script to simplify maintenance and promote re-use
- jQuery Templates, JsRender, and other scripts are available to handle client-side templates
- Canvas and SVG can be used to render charts and other types of drawings, art, and graphics
- Video can be integrated without external plugins