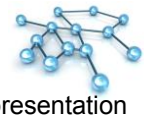
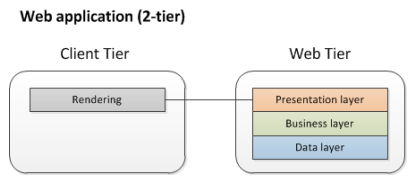
Rich Internet Applications

7. Single-page application architecture

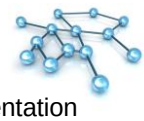
Presentation layer



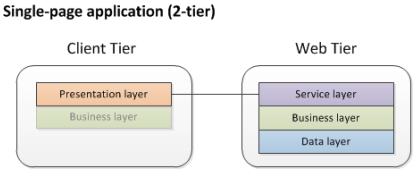
- In a “traditional” web application, presentation layer is within web tier, renders output to client tier (browser)



Presentation layer



- In a single-page application, presentation layer moves to client tier
- May also have an element of business logic in client tier



Client Tier presentation

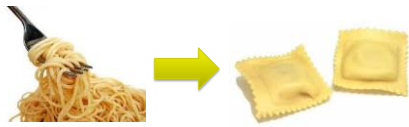
- Presentation layer technologies are HTML, JavaScript, CSS
- Easy to end up with spaghetti code – global variables, functions whose purpose is unclear, etc.
- Can improve this situation using patterns which have been tried and tested in the presentation layer in the Web Tier

Presentation layer patterns

- Model View Controller (MVC)
  - Widely used and well documented and supported in .NET by ASP.NET MVC framework
- Model View Presenter (MVP)
  - Largely superseded in ASP.NET by MVC, but basis for…
- Model View ViewModel (MVVM)
  - Not really a Web Tier pattern, used in desktop (WPF) and web (Silverlight) clients
  - Well documented for those applications

Benefits of presentation layer patterns

- Provide structure for presentation layer
- Separation of concerns
- Understandability
- Maintainability
- Testability

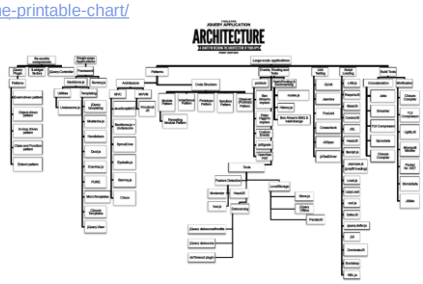


Support for client side presentation development

- Patterns and documentation
- Many JavaScript libraries….see next slide
- Take care - wide range in scope, adoption and support for these
- Libraries play different roles in supporting development:
  - Framework for implementing patterns, e.g. Backbone.js for MVC, Knockout for MVVM
  - Support for implementing specific functionality within overall architecture, e.g. History.js

Many JavaScript libraries

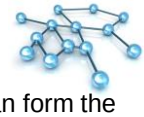
- http://addyosmani.com/blog/tools-for-jquery-application-architecture-the-printable-chart/



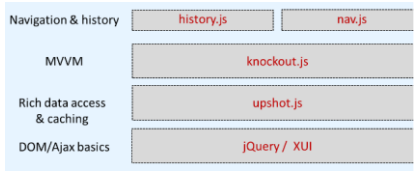
Aspects of architecture

- Identify aspects of developing for the Client Tier presentation layer
  - Patterns
  - Models
  - Data binding
  - Controllers/routing
  - Views/templates
  - Testing
- Will look at example(s) of use of libraries which support each of these

MVC 4 SPA template libraries



- Includes a set of libraries which can form the basis of a SPA
- Can choose to use or replace any or all of these

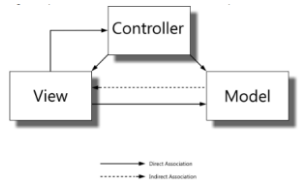


MVC pattern

- Based on pattern which is widely used in ASP.NET MVC, Ruby on Rails, J2EE, PHP…
- Roles of components similar to but not identical to Web Tier application
  - JavaScriptMVC
  - Backbone.js
  - Sproutcore

MVC components

- The **Model** refers to the data and business functionality of the application
- The **View** is the visual representation of the Model
- The **Controller** is a component which responds to user input



MVVM pattern

- Named by Microsoft, though based on earlier Presentation Model
- Makes use of two-way binding support and command model of WPF in particular
- JavaScript implementation of this pattern supported by Knockout.js

Models

- Model is presentation layer's view of the data associated with service
- Encapsulates response to an Ajax call or data to be posted
- Consists of JavaScript object(s)
- Objects can be created in Ajax callback from data in Ajax response

Creating objects

- Numerous ways to create objects in JavaScript, including:
  - Simple objects and object literals
  - Constructor functions and new
  - Object.create
  - Module pattern
  - Revealing module pattern

Models in Knockout

- Model
  - As defined previously
- View Model
  - a pure-code representation of all the data and operations on a UI
  - e.g. a list of model items to be displayed and methods to add and remove items
  - Knockout can bind view model properties to UI elements
  - Can also bind view model methods to UI events

Simple View Model example

- View model created with object literal

```
var myViewModel = {
    personName: 'Bob',
    personAge: 123
};
```

- Bind to UI element

```
The name is <span data-bind="text: personName"></span>
```

- Set up bindings initially

```
ko.applyBindings(myViewModel);
```


Knockout observables

- Often useful to update UI as property value changes
- Not just a one-time binding
- Need to use Knockout observables to allow changes in model properties to be detected and applied to UI

```
var myViewModel = {
    personName: ko.observable('Bob'),
    personAge: ko.observable(123)
};
```

View Model functionality

- Simple example had no methods and no model
- More realistic view models may have:
  - Properties which are model objects or collections thereof
  - Methods which use Ajax calls to load or post model objects
  - Methods which are bound to UI events, e.g. mouse clicks, or to changes in properties of the view model itself

View Model example

```
function ViewModel() {
    var self = this;
    self.availablePackages = ko.observableArray([]),
    self.loadPackages = function () {
        $.ajax({
            type: "GET",
            url: "/Datasources/Packages",
            dataType: "json",
            success: function (json) {
                self.availablePackages.removeAll();
                $.each(json, function (i, item) {
                    self.availablePackages.push(
                        new PackageReference(item.packagename, item.packageID));
                });
            }
        });
    },
```

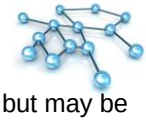
view model property

view model method

model

```
function PackageReference(packagename, packageID) {
        this.packagename = packagename;
        this.packageID = packageID;
};
```

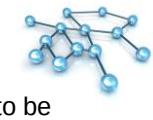

Business logic in model



- Business logic usually in web tier, but may be need for some logic on client, e.g. validation
- e.g. using Knockout.validation library
  - http://jsfiddle.net/ericbarnard/KHFn8/

```
var viewModel = {
    firstName: ko.observable().extend({ minLength: 2, maxLength: 10 }),
    lastName: ko.observable().extend({ required: true }),
    submit: function () {
        if (viewModel.errors().length >0) {
            alert('Please check your submission.');
            viewModel.errors.showAllMessages();
        }
    }
};
```

Benefits of view model



- Allows state and behaviour of UI to be encapsulated in an object
- Object does not contain any code which explicitly refers to UI
- UI and view model combine through Knockout's model binding
- View model can be unit tested in isolation from UI

Namespaces

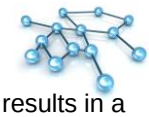
- Put models into namespace to reduce number of objects and functions that are added to the global scope
- Avoid possible name clashes
- Create empty object with name of your "namespace" and add models to that
- Common to use short name, e.g. Knockout uses **ko**

```
var my = my || {};
my.ViewModel = function () {…};
Var viewModel = new my.ViewModel();
```


Controllers

- MVC pattern has the concept of a controller
- Controller is the glue between application's views and models
- Takes user input and calls a function in response
- That function may interact with model and cause view to be rendered
- MVVM does not have controllers – their role is carried out by view model and binding

Routing



- User action in traditional web app results in a request being sent to a URL on the server
- In ASP.NET MVC, request is routed to controller/action
- In Ajax app, user action can also be expressed as URL, e.g. clicking link
- If URLs use fragment identifiers then client-side code can read these and respond to change of window.hash by calling a function depending on its value

Routing

- In a sense, the controller's job is simply to **route** URLs to functions
- Routing libraries abstract the task of reading the fragment identifier and calling the appropriate method
- This capability may be built in to framework or be provided by a standalone library
- Backbone.js has Router object, which was known as Controller in early versions

Backbone Router example

```
var AppRouter = Backbone.Router.extend({
    routes: {
        "/posts/:id": "getPost",
        "*actions": "defaultRoute"
    },
    getPost: function( id ) {
        alert( "Get post number " + id );
    },
    defaultRoute: function( actions ){ alert( actions );
    }
});
var app_router = new AppRouter;
Backbone.history.start();
…
<a href="#/posts/120">Post 120</a>
<a href="#/posts/130">Post 130</a>
```

Routing with Knockout

- Knockout uses binding rather than routing
- However, may want to have page states in single page application defined by unique URLs
- Can combine Knockout with standalone routing library
- e.g. Sammy.js, Davis.js (!)

Knockout with Sammy

```
function AppViewModel() {
    var self = this;
    self.go_to_books = function() { location.hash = "books" };
    self.go_to_about = function() { location.hash = "about" };
    Sammy(function() {
        this.get('#books', function() {
            $("#about").hide();
            $("#books").show();
        });
        this.get('#about', function() {
            $("#books").hide();
            $("#about").show();
        });
    }
}
```

routing

data binding

```
<a href="" data-bind="click: go_to_books">Books</a>
<a href="" data-bind="click: go_to_about">About</a>
```


Views

- Interface to the application
- Logicless HTML fragments managed by the application's controllers
- Data needed for view transferred to client by Ajax calls, loaded into models
- No HTML rendered by server
- Load data into views by creating DOM elements dynamically or by using templates and/or data binding

Templates

- JavaScript template is an HTML fragment interpolated with template variables
- Conceptually similar to ASP.NET MVC views except that rendering is done in client
- View consists of markup which selects templates and specifies data to render
- Further separation of concerns
- Templates can potentially be reused

Storing templates

- Various options which are available include, depending on the specific template engine:
- Inline in the JavaScript
- Inline in the HTML
- Inline in a custom script tag
- Loaded remotely
  - Usually loaded with Ajax call
- Trade-offs in performance between inline/remote templates

Some template engines

- jQuery.tmpl
  - Note: The jQuery team has decided not to take this plugin past beta…
- jsViews, jsRender
  - Replacement for jQuery.tmpl, still at early stage of development
- Mustache.js
- Knockout templates
  - Built-in capability in Knockout

Template examples - Knockout

```
<div data-bind="template: { name: 'person-template', foreach: people }"></div>

<script type="text/html" id="person-template">
   <h3 data-bind="text: name"></h3>
   <p>Credits: <span data-bind="text: credits"></span></p>
</script>

function MyViewModel() {
    this.people = [
        { name: 'Franklin', credits: 250 },
        { name: 'Mario', credits: 5800 }
    ]
}
ko.applyBindings(new MyViewModel());
```

- See also sample code on module website

Template example – Knockout with jQuery.tmpl

```
<div data-bind="template: 'peopleList'"></div>

<script type="text/html" id="peopleList">
   {{each people}}
      <p><b>${name}</b> is ${age} years old</p>
   {{/each}}
</script>

<script type="text/javascript">
   var viewModel = {
      people: ko.observableArray([
         { name: 'Rod', age: 123 },
         { name: 'Jane', age: 125 },
      ])
   }
   ko.applyBindings(viewModel);
</script>
```


Templates and data binding

- Knockout example used data binding
- Binds view model to template declaratively
- Can render jQuery.tmpl templates imperatively with $.tmpl method
- Need to create template first with $.template method
- jQuery.tmpl templates are simply strings

Template example – jQuery.tmpl

```
var markup = "<li><b>${Name}</b> (${ReleaseYear})</li>";

// Compile the markup as a named template
$.template( "movieTemplate", markup );

$.ajax({
    dataType: "jsonp",
    url: moviesServiceUrl,
    jsonp: "$callback",
    success: showMovies
});

// Within the callback, use .tmpl() to render the data.
function showMovies( data ) {
    $.tmpl( "movieTemplate", data ) .appendTo( "#movieList" );
}
```


Unit testing

- One of the benefits of structuring the presentation layer is testability
- Can test components separately, e.g. functionality of view model in Knockout
- Automated tests
- Same principles as server-side, e.g. Arrange/Act/Assert
- Need a client-side unit test framework, e.g. qUnit, Jasmine

Using qUnit

```
<html>
<head>
        <title>QUnit Test Suite</title>
        <link rel="stylesheet" href="qunit.css" type="text/css" media="screen">
        <script type="text/javascript" src="qunit.js"></script>
        <!-- Your project file goes here -->
        <script type="text/javascript" src="myProject.js"></script>
        <!-- Your tests file goes here -->
        <script type="text/javascript" src="myTests.js"></script>
</head>
<body>
        <h1 id="qunit-header">QUnit Test Suite</h1>
        <h2 id="qunit-banner"></h2>
        <div id="qunit-testrunner-toolbar"></div>
        <h2 id="qunit-userAgent"></h2>
        <ol id="qunit-tests"></ol>
</body>
</html>
```

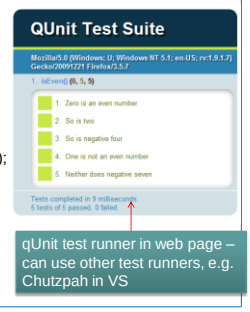
include CSS and markup for test runner

Using qUnit

- Write tests and open page

```
// test this function
function isEven(val) {
        return val % 2 === 0;
}

// test code
test('isEven()', function() {
        ok(isEven(0), 'Zero is an even number');
        ok(isEven(2), 'So is two');
        ok(isEven(-4), 'So is negative four');
        ok(!isEven(1), 'One is not an even
          number');
        ok(!isEven(-7), 'Neither is negative
          seven');
})
```

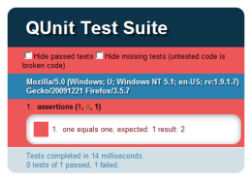


qUnit test runner in web page – can use other test runners, e.g. Chutzpah in VS

Using qUnit

- Tests can include asserts, e.g. ok, equal (equals in earlier versions)

```
test('assertions', function() {
        equal( 2, 1, 'one equals one');
})
```



Testing Ajax calls

- When testing functions which include Ajax calls, need to pause test to allow time for callback to execute

```
asyncTest('asynchronous test', function() {
        // call function under test

        // The test is automatically paused
        setTimeout(function() {
                // assertion

                // After the assertion has been called,
                // continue the test
                start();
        }, 100)
})
```

- See also sample code on module website

Mocking Ajax calls

- Mocking allows known data to be supplied in response to Ajax call under test conditions
- Repeatable, doesn't depend on state of data on server
- Use a library such as mockjax

Dependency management

- Large-scale JavaScript apps may require many library script files to be loaded, sometimes in a specific order
- Basic way of managing this is simply to add script tags to your page
- JavaScript module loader can allow the process of loading scripts more manageable and also optimise to reduce requests for files

Require.js

- Allows you to specify that a particular part of your code requires a specific set of modules to be loaded
- Will load modules asynchronously and execute callback when loaded

```
<script src="scripts/require.js"></script>
<script>
   require(["some/module", "a.js", "b.js"], function(someModule) {
      //This function will be called when all the dependencies are loaded..
      //This callback is optional.
   });
</script>
```

What's next

- Alternatives to JavaScript/Ajax
- Silverlight, GWT, etc.