Redwood

MVVM in ASP.NET

# The problem

Since ASP.NET WebForms is past its days and Microsoft plans to continue with MVC, Web Pages and ASP.NET Web API, there is a gap for a new framework. MVC is good for front-end applications with simple forms because it gives you control over HTML that is rendered to the client.

However in the world of enterprise Intranet sites there is a demand for big applications with hundreds of complicated forms that can’t be easily fit in the MVC pattern, partly because MVC doesn’t help you with maintaining the state as WebForms‘ viewstate did. The rendered HTML or the size of “viewstate” is not an issue on Intranet sites, but the speed of development and maintenance costs are very important.

The MVVM pattern is better suited for this “desktop-like” applications and many Javascript MVVM frameworks are being used today. The biggest issue in applications built e.g. in Knockout JS and ASP.NET Web API is that you need to write code in 2 languages and most of the boring work includes transforming data and handling the communication between client and server. These things require many lines of javascript code and the debugging and finding errors is also quite tricky. Instead of focusing on the business logic and the application itselt, the developer has to spend time fighting the infrastructure.

# How it works

Redwood is a framework that allows you to write MVVM applications in pure C#. There are 2 files the developer has to write for each web page – the view and the viewmodel. The view is a HTML file with special syntax extensions that allows you to use data-binding and server controls. The viewmodel is a C# class that holds all the state information of the web page, and also contains commands that can be invoked from the page.

When the browser enters the Redwood page and sends a HTTP GET request, the framework instantiates the viewmodel and processes the HTML. All bindings in the HTML are translated to Knockout JS bindings and server controls are rendered. Then the viewmodel is serialized into JSON and embedded in the HTML to be sent to the client. Also a small Javascript library is referenced in the page so the Redwood can work properly. When the client loads the HTML page, the viewmodel is deserialized and passed to the Knockout JS.

When the user clicks a button that invokes a command in the viewmodel, the Redwood JS library serializes the viewmodel, and sends it in the AJAX POST request to the server. The framework deserializes the viewmodel from the client, calls the function, serializes the viewmodel again and sends it back to the client.

Some commands in the viewmodel only change properties of a viewmodel and don’t have any external dependencies. They don’t access the database, the server filesystem etc., so making a postback in that case is unnecessary. In the future versions of Redwood there is a plan to translate the functions from C# to Javascript and don’t do postbacks for these simple commands.

It will also be possible to invoke stateless command that won’t send the viewmodel at all. It will be possible to pass arguments to the stateless command and on the server it will be represented by static method on the viewmodel.

The main goal of this framework is to allow the developers to write everything in C# and HTML without the need to use javascript. The framework will push everyone to use MVVM. It might be allowed to reference controls in the view by ID in the future and access their properties directly, but the data-binding will always be the preferred way.

# Sample 1 – The Basics

<div visible="{value: IsOnFirstPage}">

<h1>Enter your name</h1>

<table>

<tr>

<td>First Name: </td>

<td>

<rw:textbox text="{value: FirstName}" />

</td>

</tr>

<tr>

<td>Last Name: </td>

<td>

<rw:textbox text="{value: LastName}" />

</td>

</tr>

<tr>

<td colspan="2">

<rw:button text="Submit" onclick="{command: Submit()}" />

</td>

</tr>

</table>

</div>

<div visible="{value: !IsOnFirstPage}">

<h1>Thank you!</h1>

<p>Thank you, **{{**value: FirstName + " " + LastName**}}**</p>

</div>

public class Sample1ViewModel

{

public bool IsOnFirstPage { get; set; }

public string FirstName { get; set; }

public string LastName { get; set; }

public Sample1ViewModel()

{

IsOnFirstPage = true;

}

public void Submit()

{

// write the entry to the database

}

}

# View Syntax

The view is HTML file with extensions described below.

1. The value of any HTML attribute can be a binding – the syntax is   
     
   <div class="{value: RowCssClass}" />  
     
   After the colon, there is C# expression. There are limitations of what can be translated into javascript. It will be allowed to call some functions that we can translate to javascript.
2. If the binding is not an attribute value, it must be enclosed in double curly braces.   
     
   <p>Thank you, **{{**binding: FirstName + " " + LastName**}}**</p>
3. All HTML elements get following additional attributes:
   1. **Visible** – the value is a binding that returns a boolean value. It is translated to knockout *data-bind=“visible: Expression“*
   2. **DataContext** – the value is a binding. All bindings in the child elements are evaluated against the value of the datasource attribute.
4. All elements starting with tax prefix **rw** are mapped to classes in the namespace **Redwood.Framework.Controls**. The user can map additional prefixes to own controls in the configuration.
5. On the top of the file, the directives can be specified. No text except whitespace and BOM can precede the first directive, and no text except whitespace can be between two directives. The syntax is:  
     
   @viewmodel: Redwood.Samples.Sample1.IndexViewModel, Redwood.Samples

Currently only the **viewmodel** directive is permitted and there is a full type name including assembly name. The framework will call the default constructor of the specified type and use it as the viewmodel.

# ViewModel

The Redwood viewmodel is normal C# class with these rules.

1. All public properties are serialized and transferred to the client and back, except these cases:
   1. Read-only properties or read-write properties marked with **Bind(Direction.OneWay)** are only transferred from server to client.
   2. Properties marked with **Bind(Direction.OneWayToSource)** are transferred only from client to server.
2. All public methods can be invoked from the client.  
   If the method is marked with attribute **ClientCommand(“functionName”)**, instead of performing the postback, the function specified in the attribute, is executed on the client. The **this** variable will point to the viewmodel instance on the client.
3. The ViewModel can implement the interface **Redwood.Framework.ViewModel.IRedwoodViewModel** that defines methods **Init**, **Load** and **PreRender**. There is a base class **RedwoodViewModelBase** in the same namespace and already implements these methods (they are empty) and makes them virtual so not all of them must be overridden when only one of them would contain any code.   
   Also there is a property **Context** that holds additional data and will contain some useful functions (like redirect etc.).  
   If the interface is not implemented, the framework skips calling the methods in the page lifecycle (see the following diagram).

# The Page Lifecycle

The page lifecycle is described in the following diagram.

|  |  |
| --- | --- |
|  |  |