Mark Securities and Proof of Concepts. Status Report 08/31/2017

. Initial definition of the task

Initially the task was defined by Dmitri in the following terms

. Populate a Kendo UI Grid with two columns feed by and array of values following the Kendo UI examples on Telerik's site.

(Dmitri's Kendo UI proposition was agreed by me since Telerik's components and its Kendo alternative to JQuery is well documented and backed up by Itau's license ownership).

. Concentrate in an html page containing the Kendo UI grid and with no concern about what delivers the page to a client fully agnostic of server side middle tier (controllers in ASP.NET MVC).

. Each grid should have paging (20 rows per page), sorting and filtering capabilities.

. To achieve the task of Mark Securities just to use the two views (Olympic and Pershins IsAAA respectively) and just update the column IsAAA on each view and suggested the use of OData

Service (ODA Web AAPI v 4.0). A service is an important approach because he could need to invoke the functionality from other components of his system.

. The main objective of the task is to set a Proof of Concepts as a guidance for future development it is why he wants to be fully agnostic approach regarding the middle tier and mentioned that in next iterations of the task we could decide what middle tier to use and specifically mentioned Node.js. The reading of addendum 3 at the end of this document is important.

. I mentioned some elements I think we should be aware of, like security and appearance of the web page based in user profile. His consideration in this regard is to wait until we reach the need for that.

. The design/implementation process

1. The Telerik's Kendo UI cannot be installed on my box. Several atempts and for all the result is the environment gets frozen and misbehaving. For Dmitri instead, the experience is smooth.

2. I reported the fact to HD and Cesar personally has a full understanding of the problem.

3. I considered to implement and design the most stable elements first.

4. I proceeded to implement an ODA service for the Olympic view as a starting point within a web application. This Oda service is based on the ODA v3.0 provided by Visual Studio 2015 which prepares a well advanced Controller (middle tier). Oda services can be implemented in a self-hosted server but in the .NET environment any self- hosted Oda Service is to rely on a .NET controller. Therefore to achieve the agnostic middle tier approach of Dmitri, we have to use other framework. Node.js is one of them.

5. The Oda service was train and tested with an html interface by JQuery Ajax call to the Odata Controller.

6. As a tangiable result I created a JavaScript class (JQuery based) to be used with any Odata Service or any other Web API service.

7. The presence of a service implies, generally speaking, a service help page and a discoverable interface. The latter means a mechanism to be discovered and used by other applications (the essence of a service)

8. I proceeded to create a web page app with the functionality proposed by Dmitri plus others necessary not mentioned initially. They all contained in a single web page with interchangeable functional elements with a smooth user experience (Single Page APP fashion). The control of every functional elements, web interface, is done by the presence of a menu.

a. A web interface with two bootstrap tabs each to contain the Olympic View and other for the Pershing view

b. A web interface for showing the Service Help.

c. A tentative web interface for training the service (I'm studying the possibility since I have not done before)

Important aspect to notice:

1. Dmitri suggested two web pages for each view. I prepared both in one, in different tabs, to be interface concise.

Each can be separated easily and reside in corresponding page if necessary.

2. The presence of a menu implies mandatorily the need of imposing security levels in the web page. The authorization for resources is a central piece from the architectural point of view and should be taking into consideration from the beginning.

3. The web interface for managing the functionality of Mark Securities, update IsAAA column, demands we extend the natural functionalities offered by the Oda service. Not a single entity update, batch update for multiple entities in just one call.

9. The Service help page for an Oda service cannot be created by the Web API conventional mechanisms. Though some have done specific hacks, many disapprove this approach.

10. I proceeded to install the ODA service v4.0 as recommended by Dmitri. This implies to install other package to guarantee the automatic template creation, very necessary, that Visual Studio offers for Oda Services v3.0

11. I proceeded to install the Swashbuckle package to support documentation for Oda Services and found there was incompatibility between OdataCore.dll, Oda v4.0, etcetera, with packages constraints regarding versions.

12. I stopped development and start writing this report.

13. In personal communication Dmitri ask me when we can see some operational results, tomorrow? I considered we could have an operational prototype for tomorrow late in the afternoon, or early next Monday, according this line of action:

a. Populate both grids with paging mechanism in place integrated with both Oda services: OlympicAAA and PershingAAA

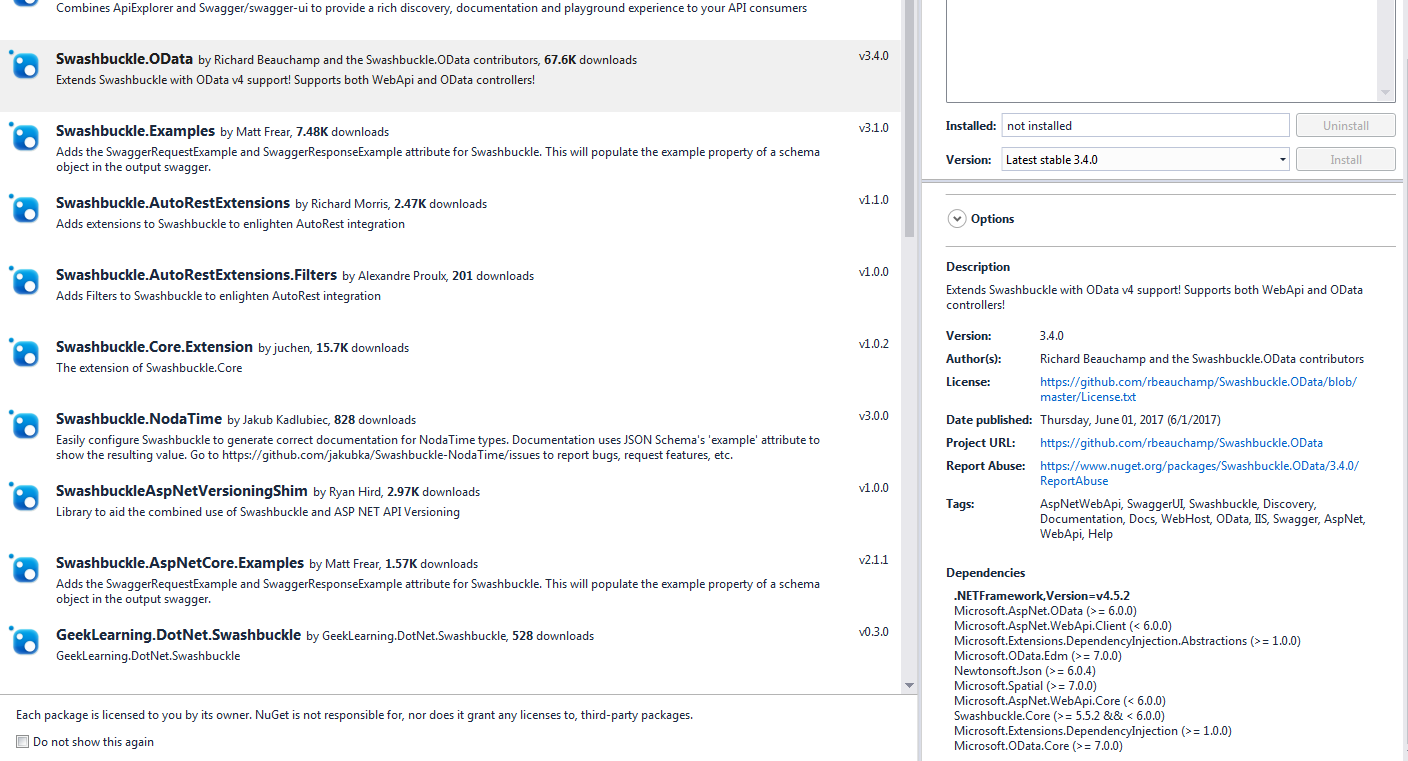
b. Create prototype web interface to simulate a help pages (a Help page per service).

c. Show and navigate the three web interfaces with smooth user experience.

d. Points a, b and c with use of conventional javascript/Jquery and Ajax calls to Oda services and simple html constructs. No Kendo UI.

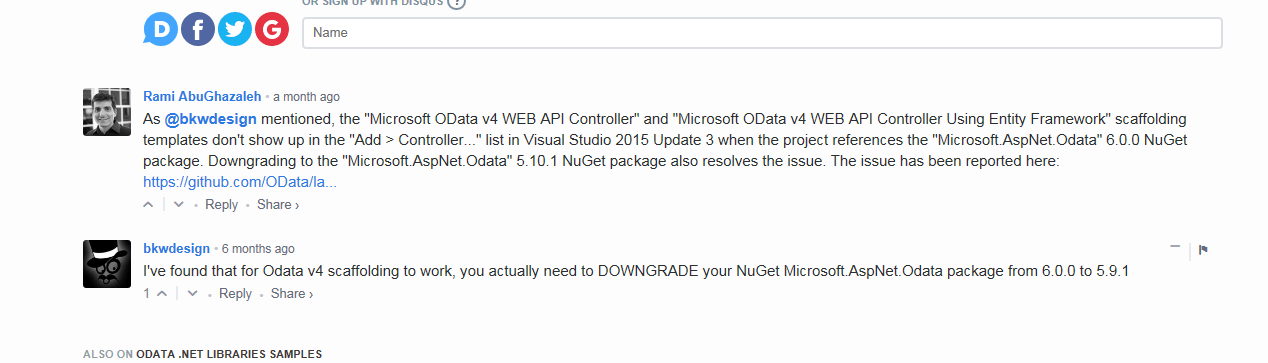
Addendum 1.

Notice the strong dependency, bottom right, of this package on different versions of previous installed dlls.



Addendum 2(from <http://odata.github.io/WebApi/11-01-OData-V4-Web-API-Scaffolding/>)

Some developers report incompatibility to install scaffolding packages.



Addendum 3

**Is Node.js learning curve longer than C# ASP.NET?**

C# [ASP.NET](http://asp.net/) takes longer.

While I’m not a user of Node.js and certainly still far away from mastery in [ASP.NET](http://asp.net/) I think that would be a safe guess.

Here is way:

1. Even if you will learn C# part of AspNET you will still have to use Javascript for client side programming. Ergo you actually have two languages to master.
2. While AspNET and Node.JS will have equivalents of all libraries (so ORM’s, services, etc) Node.JS is way younger. That means it libraries may not be so bloated (or robust) as AspNet.
3. JavaScript is more flexible language than C# this means a lot of things may be easier to quickly hack out than to painfully stack all the blocks together to get the same result in C#.

I will also throw in solo counter-argument:  
AspNet is tightly bound with .Net framework so you have a handy set of well documented libraries for nearly everything, while in Node.JS you have to use third party libraries for a lot of things. And that means you have to know the most popular ones, some may go unsupported, some may be buggy, some may have flexible backward compatibility policies, not to mention varying level of documentations.