ONE ASP.NET- A STRATEGY FOR HAPPINESS

Web Forms, MVC, Web API, and SignalR:
How Can I Choose Just One?

Telerik

CONTENTS

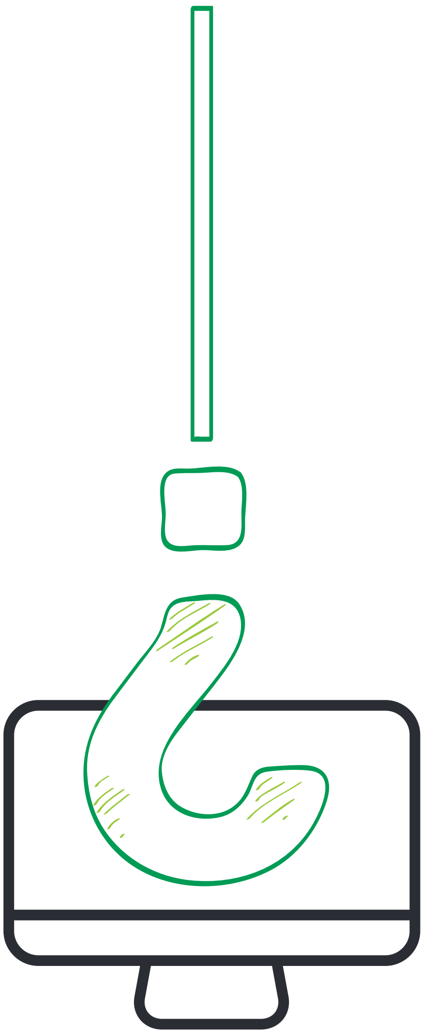
About the Author

Jeffrey T. Fritz is a Microsoft MVP in ASP.NET, an ASPInsider and Developer Evangelist for Telerik with over 15 years of experience building large-scale multi-tenant web applications in the software-as-a-service model. A Penn State graduate with a degree in Management Sciences and Information Systems, he speaks regularly at developer events through the INETA speaker program and maintains blogs at www.csharpfritz.com and **blogs.telerik.com/jefffritz**. You can find him on Twitter at **@csharpfritz** and can reach him at **jeff.fritz@telerik.com**.

WHY YOU SHOULD CONSIDER A HYBRID-PROJECT APPROACH

The web has evolved at an amazing pace. Nothing in history has grown and changed as rapidly as today's technology. Every day there are new inventions and new techniques, quickly pushing aside the older generation in favor of the new and shiny. ASP.NET is no different, with the addition of MVC, WebPages, WebAPI, SignalR, and SPA frameworks. Each one of these frameworks added something new and fresh to the landscape of ASP.NET. With each addition, millions of software developers everywhere asked themselves the question: Do I re-write my website to use "technology x"?

This document will outline the reasons your organization should consider a hybrid-project approach. We will demonstrate specific instructions for embracing this project architecture with Visual Studio 2012 and will conclude with some statements about Visual Studio 2013 and ASP.NET 4.5.1.



Don't Be Tempted to Exclusively Use the Latest and Greatest

When I was a child, both of my parents drove cars to work: my father drove a simple hatchback and my mother drove a sedan. After working hard for many years, my dad came home one day in a shiny new Corvette. He still had his little hatchback, but I distinctly remember asking, "Dad! This is the coolest car! But why didn't you get rid of your old one?" "No no...", he said, "this is just a car for fun... I'm not going to drive it everywhere. But I'm a fan of sports cars, and I finally decided it was time to get something that would be fun to drive every now and again."

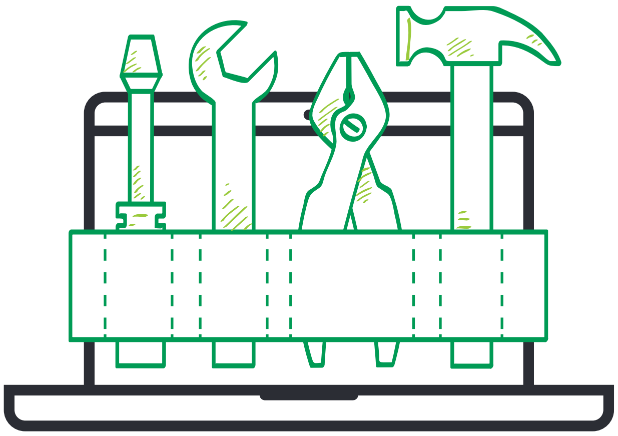
Similarly, don't look at the various ASP.NET frameworks as your everyday tool to build everything. Each of these frameworks is a tool you can use judiciously in various parts of a website to deliver the best capabilities and content, in various patterns benefitting different needs differently. You don't need to drive your father's sports car every day to the office. Some days, you need something better suited to drive through snow and rain—and that shiny Corvette won't do.

Starting in 2005, I participated in writing an application with millions of lines of code using ASP.NET 1.1. We had hundreds of webpages and controls. When ASP.NET MVC was released in 2009, we examined it and determined that it made sense to use in the next module we would build in our application. It would be JavaScript intense, have massive amounts of business logic and would benefit greatly from unit tests. You can likewise make this decision, and add MVC to an existing web forms project as the need arises.

Remember, the cost to rewrite an application is significant. Every dime spent towards technical and human resources throughout the course of application construction will be lost when the rewritten application is released. Why then are you rewriting the application? Do you need to rewrite everything in the application? Is there some shiny new framework an eager software developer wants to use that is "incompatible" with your current application's source code? These are all questions that project stakeholders should be asking themselves when faced with this scenario.

Fortunately, ASP.NET is not built with a one-size fits all mentality. Microsoft has given us a significant set of tools in this technology that nicely complement each other. You NEVER need to scrap an entire application to use the next available ASP.NET technology.

Add More Tools to Your Tool Belt: One Tool Does Not Solve All Problems



In the early days of ASP.NET, the only framework available to developers was ASP.NET Web Forms. Every page, no matter how simple or static, needed to be constructed with an ASPX file that would have ViewState, PostBacks, and a full event life-cycle. When used improperly, these features in a web form can consume significant memory and processor resources.

Forward thinking developers, who want to shorten their development processes and encourage better design patterns in their applications, would build their own frameworks on top of the web forms architecture. The creation of FubuMVC and MonoRail are two examples of frameworks built by the community that make use of the Web Forms framework.

In 2009 when Microsoft released ASP.NET MVC, there was a collective rush from developers to explore a new framework. This rush lead many developers to think that they needed to completely abandon the "old" ASP.NET and build projects from scratch in order to make use of the new MVC framework. Further, developers started to use MVC for every project, claiming it was the most efficient use of their resources. This is symptomatic of the "Golden Hammer" anti-pattern.

This anti-pattern suggests a magical tool that can be used to solve every programming challenge. Project stakeholders should want their development teams to have the broadest set of capabilities available to them. Not every challenge is the same, and each needs to be addressed in its own way.

Our toolset has been extended by Microsoft to include five server-side frameworks in ASP.NET:

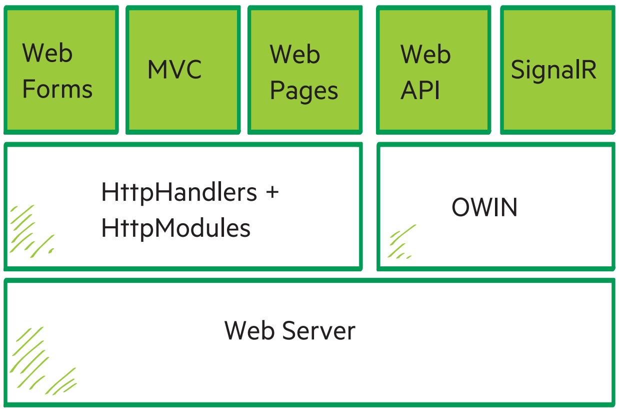


FIGURE 1 - THE ASP.NET 4.5 FRAMEWORK

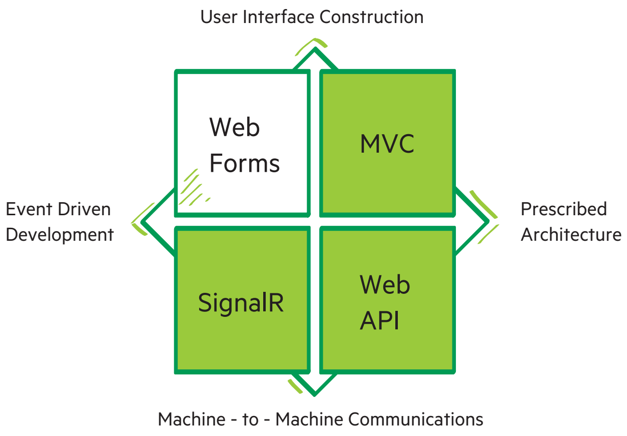


FIGURE 2 - THE FOUR QUADRANTS OF ASP.NET FRAMEWORKS

Each of these frameworks has significant features to assist your project construction. Choose the right framework to make your job easier.

The four major components of ASP.NET have the following significant features:

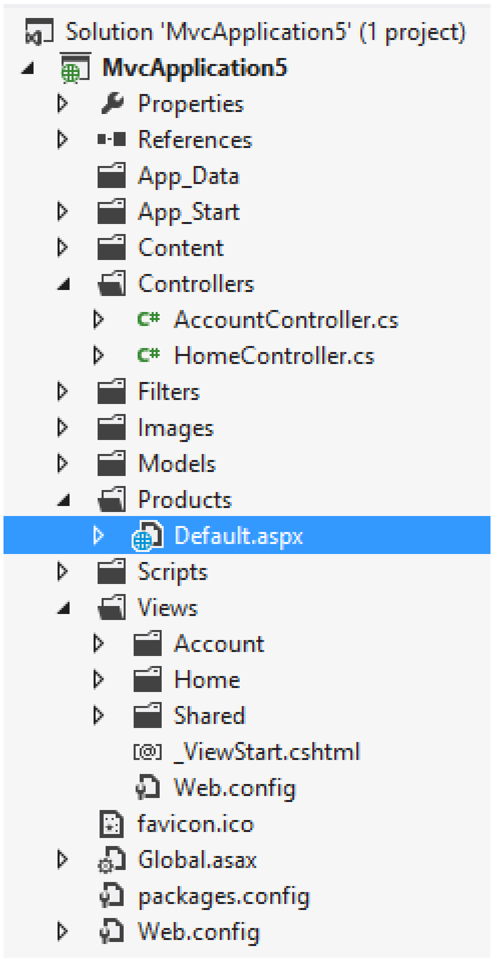
- **Web Forms:** An event-based programming model that delivers web content using an abstracted object model for web pages.
- **MVC:** An opinionated programming model that requires developers to use the Model-View-Controller architecture to deliver content. Developers have full control over all content delivered from a web server.
- **WebAPI:** A framework for controlling and managing RESTful based services that fully embrace the HTTP protocol and make full use of its features. Makes use of a similar controller architecture in use by MVC.
- **SignalR:** A framework to deliver "real-time" interactivity with attached clients using modern web communication protocols. The web server has the ability to call and execute code on each and any of the attached clients.

The three green frameworks are client-side intense, as there are significant capabilities that need to be written to consume the resources exposed on the server-side. Web Forms abstracts away that requirement, and generates appropriate content for a browser and is designed to perform all logic on server-side.

In the next sections, we will walk through how to combine each of the frameworks in one project in Visual Studio 2012 and 2013.

COMBINING FRAMEWORKS IN VISUAL STUDIO 2012

HOW TO USE THE TOOLS TOGETHER



```
Solution 'MvcApplication5' (1 project)
  MvcApplication5
    Properties
    References
    App_Data
    App_Start
    Content
    Controllers
      AccountController.cs
      HomeController.cs
    Filters
    Images
    Models
    Products
      Default.aspx
    Scripts
    Views
      Account
      Home
      Shared
      _ViewStart.cshtml
      Web.config
    favicon.ico
    Global.asax
    packages.config
    Web.config
```

In the early days of ASP.NET MVC, it was very difficult to use Web Forms and MVC in one project. There were configuration changes required in web.config and DLLs that needed to be placed correctly, along with a series of folders that needed to be constructed by hand. With the advent of NuGet and Visual Studio 2012, this framework integration has become significantly simpler. In the next few sections, you will learn how to add each framework to an existing project of a different type.

Adding Web Forms to MVC

This is the simplest of the projects to manage. With an existing MVC application, all of the modules and handlers are already in place to handle web forms content. You simply need to add a new web form and get started programming.

You can maintain the look and feel of the MVC URLs that don't have extensions by implementing FriendlyUrls. Simply add the Microsoft.ASP.NET.FriendlyUrls NuGet package to your project and add the following line to your RouteConfig:

```
routes.EnableFriendlyUrls();
```

Adding MVC to a Web Forms Project

We lead from the simplest hybrid project to the most complex. Adding MVC to Web Forms has been made significantly easier thanks to NuGet.

1. Add the Microsoft.ASP.NET.Mvc package to your project
2. Update App_Start\RouteConfig.cs to include information to route the MVC urls:

```
            routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

            // Existing from Web Forms project
            routes.EnableFriendlyUrls();

            routes.MapRoute(
                name: "Default",
                url: "{controller}/{action}/{id}",
                defaults: new { controller = "Home", action = "Index", id =
UrlParameter.Optional }
            );
```

   a. Note: the default controller listed here is the same as the MVC default. You should change this to whatever controller is the appropriate default in your application.

3. Add Folders for the Models, Views, and Controllers in your project
4. Add a web.config file to your Views folder to protect it from users browsing your source code. This file can be copied directly from the default MVC project template on your Visual Studio install at:

   `C:\Program Files (x86)\Microsoft Visual Studio 11.0\Common7\IDE\ProjectTemplates\CSharp\Web\1033\MvcWebApplicationProjectTemplatev4.1.cshtml\Views`

5. <optional> To activate the native Visual Studio tooling support, such as the “Add Controller” or “Add View” dialogs, a small edit needs to be made to the csproj file for your project.

   a. Right click your web project name in the Solution Explorer and choose “Unload Project”

   b. Right click your web project again, and choose “Edit MyProject.csproj” or whatever your project name is.

   c. In the XML source of the project file, add the following GUID to the front of the Project/PropertyGroup/ProjectTypeGuids element:

```
{E3E379DF-F4C6-4180-9B81-6769533ABE47};
```

   d. This element should now look like the following:

```
<ProjectTypeGuids>{E3E379DF-F4C6-4180-9B81-6769533ABE47};{349c5851-65df-11da-9384-00065b846f21};{fae04ec0-301f-11d3-bf4b-00c04f79efbc}</ProjectTypeGuids>
```

   e. Close the project source file, right-click on the project name in the Solution Explorer and choose “Reload Project” You should now be able to right-click on the Controllers folder and choose “Add Controller”

Adding WebAPI to a Web Forms Project

WebAPI is available as a standalone project template, under the list of templates available when starting an MVC project type. However, it is trivial to add WebAPI capabilities to an existing Web Forms project.

1. Right-click on a folder and choose "Add – Web API Controller Class" and Visual Studio will include the necessary libraries to get started with WebAPI
2. Add App_Start\WebApiConfig.cs to manage routes for your API controllers. The original WebApiConfig for WebAPI can be copied from:

   ```
   C:\Program Files (x86)\Microsoft
   Visual Studio 11.0\Common7\IDE\
   ProjectTemplates\CSharp\Web\1033\
   WebApiApplicationProjectTemplatev
   4.1.cshtml\App_Start
   ```

3. In Global.asax.cs register the WebAPI configuration before RouteConfig is managed in Application_Start:

```
protected void Application_Start()
{
  WebApiConfig.Register(GlobalConfiguration.Configuration);
  FilterConfig.RegisterGlobalFilters(GlobalFilters.Filters);
  RouteConfig.RegisterRoutes(RouteTable.Routes);
  BundleConfig.RegisterBundles(BundleTable.Bundles);
}
```

Adding WebAPI to an Existing MVC Project

Fortunately, this hybrid-project type is just as easy as adding web forms to MVC. The WebAPI extensions are already present and configured in an MVC 4 project, and the only step necessary is to start adding Web API controllers to the project.

Adding SignalR to an Existing Web Forms or MVC Project

This process is very similar in structure to adding WebAPI to web forms and is the same for both web forms and MVC project types.

1. Right-click on a folder in your project and choose "Add – SignalR Hub Class" and Visual Studio will add the appropriate NuGet packages to your project.
2. Add routing for the SignalR extension with the following line in your

   ```
   App_Start\RouteConfig.cs file:
   a. routes.MapHubs();
   ```

You should now be able to start communicating with SignalR hubs on the server.

ADVANTAGES OF THE HYBRID PROJECT

With your web project in this state, you can begin to take significant advantage of the mash-up frameworks. Consider what you can now achieve with this project structure:

- Use a Web Form to host Web Controls that simplify graphing or some other complex UI presentation in an MVC project.
- Use MVC to generate and manage lightweight markup and data presentation in a Web Forms application
- Add an API to a web project for other devices or client applications to communicate.
- Build out a browser-side single-page-application framework that only fetches and submits data through an API
- Add real-time updates to your existing web-based dashboards
- Perform long-running operations on the web server and be notified asynchronously once they have completed
- Transparently share static assets such as JavaScript and CSS between project components

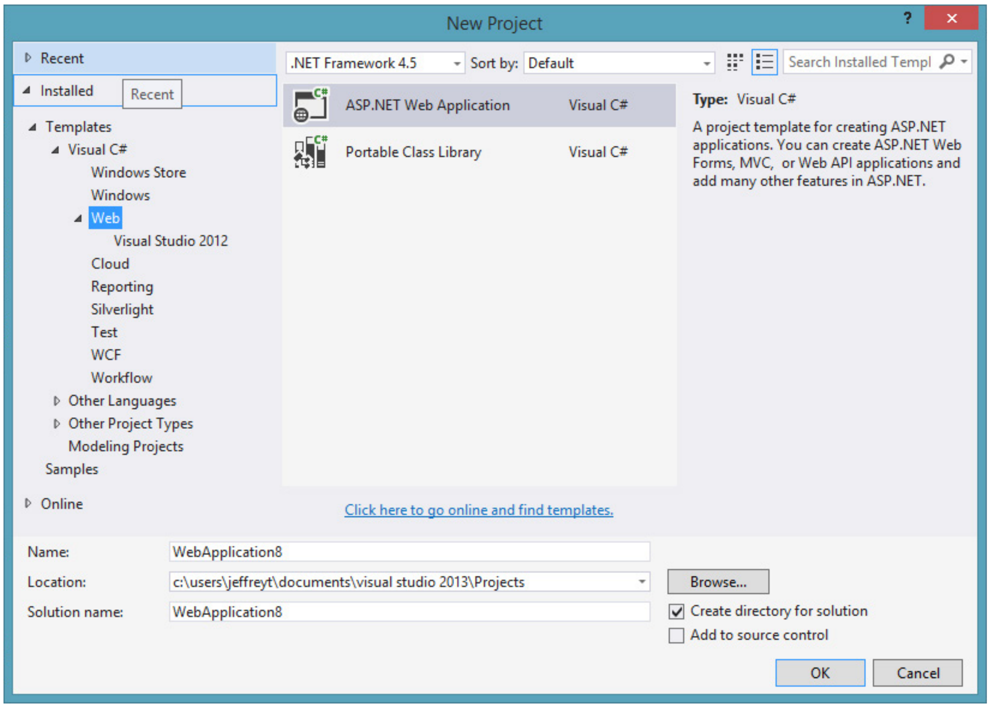
DISADVANTAGES OF THE HYBRID PROJECT

Once you start combining these technologies, there are some challenges that come along with the architecture:

- Requested Url routing confusion: For example: A browser requested the page /Product/Search. Is that a web form using FriendlyUrls or is that an MVC or WebAPI endpoint?
- Re-writing of shared assets: User Controls written for Web Forms cannot be used as part of a view with an MVC controller. MVC Views cannot be parsed and consumed by Web Forms.
- Testability is challenged: Web Forms is notoriously untestable with unit tests. When combined with an MVC project, the problem has not gone away.

HYBRID ASP.NET APPLICATION CONSTRUCTION IN VISUAL STUDIO 2013

With the release of Visual Studio 2013 and Microsoft .Net 4.5.1, we see the unification of this fragmented ASP.NET environment and the delivery of a single ASP.NET project model.



.NET Framework 4.5 Sort by: Default

ASP.NET Web Application	Visual C#
Portable Class Library	Visual C#

Type: Visual C#

A project template for creating ASP.NET applications. You can create ASP.NET Web Forms, MVC, or Web API applications and add many other features in ASP.NET.

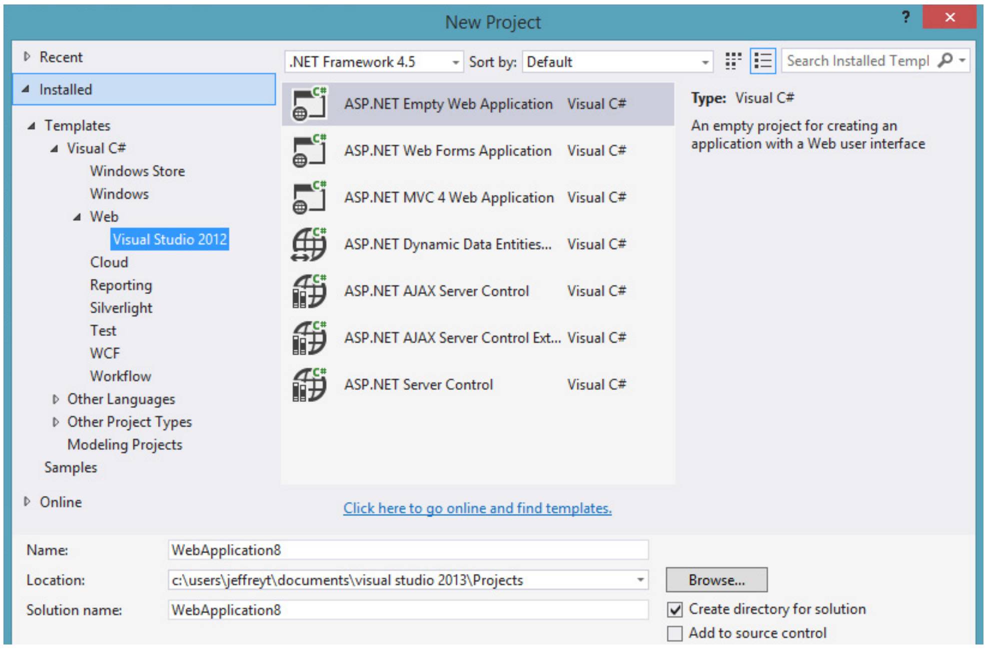
Click here to go online and find templates.

Name: WebApplication8
Location: c:\users\jeffreyt\documents\visual studio 2013\Projects
Solution name: WebApplication8

☑ Create directory for solution
☐ Add to source control

FIGURE 2 - THE FOUR QUADRANTS OF ASP.NET FRAMEWORKS

To facilitate the maintenance of projects in the legacy model, Microsoft has kept those older project types around. In Visual Studio 2013, under Web, the Visual Studio 2012 element will allow you to choose a project type from those historical project types:


New Project

- Recent
- Installed
  - Templates
    - Visual C#
      - Windows Store
      - Windows
      - Web
        - Visual Studio 2012
      - Cloud
      - Reporting
      - Silverlight
      - Test
      - WCF
      - Workflow
    - Other Languages
    - Other Project Types
    - Modeling Projects
  - Samples
- Online

.NET Framework 4.5 | Sort by: Default | Search Installed Templ

ASP.NET Empty Web Application	Visual C#
ASP.NET Web Forms Application	Visual C#
ASP.NET MVC 4 Web Application	Visual C#
ASP.NET Dynamic Data Entities...	Visual C#
ASP.NET AJAX Server Control	Visual C#
ASP.NET AJAX Server Control Ext...	Visual C#
ASP.NET Server Control	Visual C#

Type: Visual C#

An empty project for creating an application with a Web user interface

Click here to go online and find templates.

Name: WebApplication8

Location: c:\users\jeffreyt\documents\visual studio 2013\Projects | Browse...

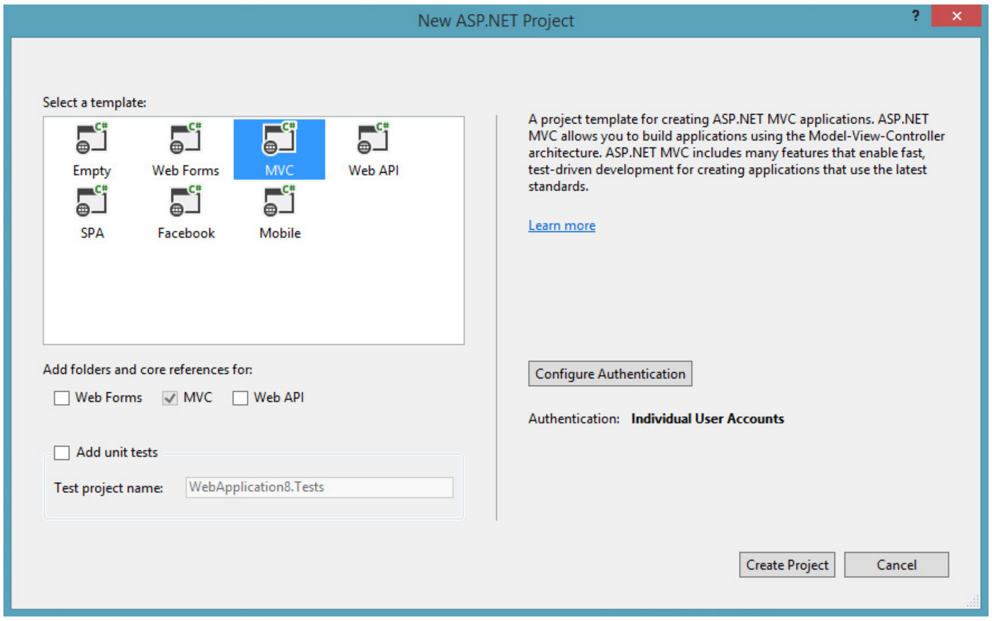
Solution name: WebApplication8

[✓] Create directory for solution

[ ] Add to source control

FIGURE 4 - LEGACY PROJECTTYPE AVAILABILITY IN VISUAL STUDIO 2013

However, once you begin a project with the new ASP.NET Web Application type, you will be led through a series of dialogue windows to assist you in the initial configuration of your application.


New ASP.NET Project

Select a template:

Empty | Web Forms | MVC | Web API | SPA | Facebook | Mobile

A project template for creating ASP.NET MVC applications. ASP.NET MVC allows you to build applications using the Model-View-Controller architecture. ASP.NET MVC includes many features that enable fast, test-driven development for creating applications that use the latest standards.

Learn more

Add folders and core references for:

[ ] Web Forms [✓] MVC [ ] Web API

[ ] Add unit tests

Test project name: WebApplication8.Tests

Configure Authentication

Authentication: **Individual User Accounts**

Create Project | Cancel

FIGURE 5 - INITIAL CONFIGURATION SCREEN FOR A NEW ASP.NET PROJECT

There are several project templates in this initial screen, similar to the project template screen we have in MVC Projects today. Each one of the templates has a description on the right detailing what it is optimally configured for, and a default set of configuration options underneath. You are not restricted from adding Web Forms to an MVC project, or adding MVC to a Web Forms project.

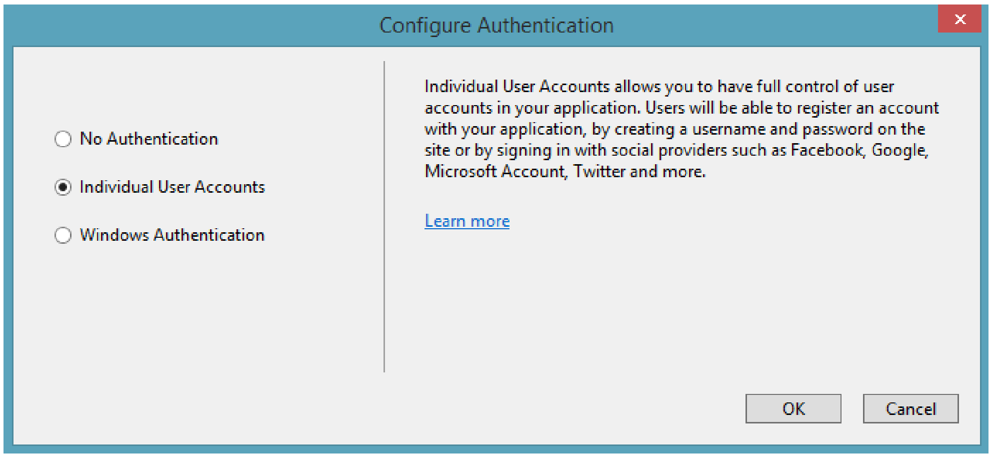
Additionally, you have the option to generate a unit test project, just as there are previously in the MVC project template dialogue.

The next option, a new element to the ASP.NET configuration wizards, is the ability to Configure Authentication.

In Visual Studio 2013 there are three options available to the developer:

- No Authentication – very self-explanatory
- Individual User Accounts – allows users to register and manage their username and password credentials
- Windows Authentication – uses the integrated Windows Authentication services available through either local or Active Directory domain accounts

This dialogue offers me some hope for the future. I can see this being extended to include options to configure OAuth or other third-party security services providers. After struggling with the configuration of OAuth and other services in the past, I am very happy to see that authentication configuration is being handled in a clear way.



Individual User Accounts allows you to have full control of user accounts in your application. Users will be able to register an account with your application, by creating a username and password on the site or by signing in with social providers such as Facebook, Google, Microsoft Account, Twitter and more.

FIGURE 6 - AUTHENTICATION CONFIGURATION DIALOGUE

After clicking "Ok" on this screen, and the "Create Project" button on the prior dialogue, our project is created with all of the extensions and libraries added appropriately. All of the NuGet packages and configuration files are properly updated in our project to enable the various programming frameworks to interoperate seamlessly.

SUMMARY

This is the vision of One ASP.NET – reduce the complexity and confusion of multiple child frameworks. You should never be forced to use a framework that you don't want to or one that isn't as efficient or productive as you need to be. ASP.NET will no longer be a pizza that you have ordered and someone dreads because you wanted extra mushrooms on it and they don't. ASP.NET is now a buffet – choose what you like, take as little or as much as you want. Don't get married to a single architecture for everything, and don't use the same tool for every project. If you're going to be using Visual Studio 2012 in the near future, then take advantage of the simple steps outlined above to take advantage of these hybrid project capabilities. If you have already upgraded to Visual Studio 2013, enjoy this evolution of ASP.NET. This change is very exciting, and it's a part of an evolution that will help us write simpler, more efficient code that enables us to deliver excellent solutions in less time than we can today.