MICROSOFTS WEB-FRAMEWORKS IM VERGLEICH

Die Qual der Wahl

Foto: Shutterstock / bizvector

ASP.NET Web Forms, ASP.NET MVC, ASP.NET Core, ASP.NET Web Sites und ASP.NET Dynamic Data – Wer blickt da noch durch?

Das Web-Framework ASP.NET, das Entwicklern beim Erstellen von Webanwendungen unter die Arme greift, hat sich im Lauf der Jahre zu einem veritablen Kleinkosmos von Unter-Frameworks entwickelt. Da gibt es „Web Forms" und „MVC" für ASP.NET, das Basisangebot „Core" und dann noch „Website" und „Dynamic Data". Einige diese Frameworks werden mit Visual Studio 2017 immer noch ausgeliefert, auch wenn sie kaum noch Verwendung finden.

Damit Sie sich einen Überblick verschaffen können, was sich alles an mitgelieferten ASP.NET-Technologien in Visual Studio 2017 befindet, bietet dieser Artikel eine kompakte Übersicht, um bei der Entscheidung für das richtige Framework zu helfen.

ASP.NET Web Forms

Von allen Webtechnologien, die hier vorgestellt werden, ist ASP.NET Web Forms die älteste. Sie erschien bereits Anfang 2002 mit .NET 1.0. ASP.NET Web Forms wird bis heute oft genutzt und ist derzeit bei Version 4.6 angelangt. Wie der Name nahelegt, besteht in der Art der Entwicklung eine gewollte Ähnlichkeit zu Windows Forms: HTML-Oberflächen werden über einen Designer visuell zusammengebaut, Logik wird über Eventhandler implementiert; analog zur Arbeitsweise mit Windows Forms werden Steuerelemente per Drag-and-drop auf die Nutzeroberfläche gezogen und konfiguriert. Dabei umgeht das Framework die Zustandslosigkeit des Webs, indem der Zustand der Oberfläche (View State) auf dem Client im Browser gehalten und immer wieder per *POST*-Anfrage an die Server-Logik übertragen wird. Die Entwicklung mit ASP.NET Web Forms ist UI-orientiert und relativ schnell. Der Entwickler ist in der Lage, schnell sichtbare und funktionale Ergebnisse zu erzielen.

In der Praxis gerät das Konzept dieses Frameworks allerdings immer wieder mit der Natur des Internets in Konflikt: Der HTML-Code, der durch diese Art zu entwickeln entsteht, kann schnell sehr umfangreich werden; der View State trägt dazu einen großen Teil bei.

Immer wieder kommt es zum Beispiel auch zu Konflikten zwischen dem Status und der Art, wie der Nutzer in einem Webbrowser mit der Applikation interagiert. Die Browser und ihre Refresh- und Zurück-Funktionen gehen von einem zustandslosen Web aus. Ein Refresh kann bei den ständigen *POST*-Anfragen zu unangenehmen Meldungen oder nicht beabsichtigtem Verhalten führen, wenn die Daten erneut

geladen werden sollen. Oder der Zustand der Applikation ist nicht mehr gültig, wenn der Anwender den Zurück-Button drückt.

Um bei der eventorientierten Entwicklung auch in großen Projekten noch die Übersicht zu behalten und den Code so testbar wie möglich zu gestalten, empfiehlt es sich, das Entwurfsmuster Model-View-Presenter einzusetzen. Dadurch wird der Code in den Code-behind-Dateien kleiner und die Logik verlagert sich in eine testbare Klasse, den Presenter.

ASP.NET MVC

Mit .NET 3.5 kam ein neues ASP.NET-Framework ins Spiel, das einem anderen Konzept folgt. Dabei ging es nicht mehr darum, Desktop-Entwickler ins Web zu locken, sondern darum, die Natur des Webs zu akzeptieren und sogar zu nutzen. Die Webentwicklung fühlte sich mit ASP.NET MVC wieder ein Stück „richtiger" an. Auch ist das Framework nicht mehr ereignisgesteuert wie bei Web Forms, sondern aktionsgesteuert.

ASP.NET MVC folgt dem MVC-Entwurfsmuster und dem Prinzip „Convention over Configuration". Das bedeutet, dass man sich an Konventionen halten sollte, um mit dem Framework sauber zu arbeiten, anstatt alles über Konfigurationen zu steuern. ASP.NET MVC entstand in einer Zeit, in der Ruby on Rails seinen Hype erlebte. So hat Microsoft von dort einige Konzepte übernommen.

Die Nutzeroberfläche wird mit ASP.NET oder Razor geschrieben und ist von der in C# oder Visual Basic geschriebenen Logik getrennt. Dies führt zu einer wesentlich besseren Testbarkeit.

ASP.NET MVC ist nicht mehr seitenorientiert, wie es bei Web Forms der Fall ist und wie es der Name vermuten lässt, sondern basiert auf Aktionen: Der Controller führt Aktionen (Actions) aus, die Ansichten (Views) zurückgeben. Diese Aktionen hängen vom genutzten HTTP-Verb im Header der Anfrage ab (*GET*, *POST*). Das heißt, eine *GET*-Anfrage unterscheidet sich von einer *POST*-Anfrage und führt somit verschiedene Aktionen aus. Unter ASP.NET Web Forms dagegen muss man manuell prüfen, ob es sich um ein *POST* oder ein *GET* handelt.

Durch dieses Prinzip führt ASP.NET MVC das Entwickeln näher und bewusster an das Web und dessen Eigenheiten heran. Wie auch ASP.NET Web Forms eignet sich ASP.NET MVC für fast alle Applikationsgrößen.

ASP.NET Single Page Applikation

Bei der sogenannten ASP.NET Single Page Applikation handelt es sich eigentlich um eine ASP.NET-MVC-Anwendung, die mit dem ASP.NET Web API und etwas JavaScript im Client (genauer gesagt mit der Bibliothek Knockout [1]) aufgepeppt wird. In Zeiten von Angular und React wirkt die Projektvorlage etwas veraltet. Auch ist die Umsetzung nicht so umfangreich wie vielleicht erwartet.

Allerding ist dabei sehr gut zu lernen, wie sich Microsoft die Authentifizierung in Single-Page-Anwendungen vorstellt. Das Wissen lässt sich dann gut in der eigenen Applikation mit Angular oder React nutzen.

Mit einer echten Single-Page-Applikation hat das Template aber eher wenig zu tun. Für kleine Projekte ist es sicherlich gut zu gebrauchen und bewältigt seine Aufgaben. Bei großen Projekten allerdings würde man den Client-Teil (der Teil, der als Single-Page-Anwendung mit JavaScript geschrieben wird) von der Server-Applikation (der in C# geschriebene Teil, der auf dem Server ausgeführt wird) trennen und eventuell sogar getrennt hosten. Das kommt der Flexibilität und der Stabilität zugute.

ASP.NET Websites

Die ASP.NET Websites gibt es in zwei Ausprägungen: einmal als Web Forms und etwas neuer als Razor-basierte Website. Der Unterschied zu den anderen Projekten besteht darin, dass ein Website-Projekt keine Projektdatei hat. Alle Dateien eines Ordners sind Teil des Projekts.

ASP.NET Websites gibt es in der Web-Forms-Variante bereits seit .NET 1.0 und sollte in .NET 1.1 eigentlich die Web-Forms-Projekte ablösen. Einem Proteststurm aus der Community war es zu verdanken, dass das nicht passiert ist. Mit einem Update für das Framework waren die Web-Forms-Projekte sehr schnell wieder verfügbar.

Die Razor-Variante hat nichts mit ASP.NET MVC zu tun. Sie nutzen lediglich die Razor-View-Engine, um Webseiten etwas dynamischer zu machen. Sie lassen sich eher mit PHP oder klassischem ASP vergleichen. Es handelt sich also um HTML-Seiten, die mit ein bisschen C# angereichert werden, um mehr Möglichkeiten zu haben.

Eine ASP.NET-Website eignet sich hervorragend für einfache und kleine Webprojekte – kleine Sites, die ein E-Mail-Formular, ein Gästebuch oder eine andere einfache Datenbankanbindung benötigen.

Es ist nicht nötig, Websites beim Einsatz dieses Frameworks vorkompiliert auszuliefern. Beim Ablegen der Quellcodes auf dem Server werden die einzelnen Seiten beim ersten Aufruf einzeln kompiliert. Dies erleichtert das Bearbeiten zur Laufzeit, da nicht die komplette Applikation neu verteilt werden muss. Das macht es für Entwickler, die PHP oder ähnliche Skriptsprachen gewöhnt sind, wesentlich einfacher.

Was hier gerade als Vorteil beschrieben wurde, sollte allerdings nur bei einfachen Anwendungen mit geringem Risiko eingesetzt werden, da das Bearbeiten zur Laufzeit gefährlich sein kann. Bei kleinen, unkritischen Websites sollte dies aber kein Problem sein.

Die Art, wie Websites kompiliert werden, kann bei großen Applikationen zu erheblichen Einbußen beim Laufzeitverhalten führen. Falls die Programme wachsen, ist ein Vorkompilieren zweckmäßig. Dann muss auch der Quellcode nicht unbedingt ausgeliefert werden. Allerdings bleiben die genannten Vorteile auf der Strecke und es wäre sinnvoll, die Applikation in ein Web-Forms- oder ein MVC-Projekt umzuwandeln.

ASP.NET Dynamic Data

Als Gegenteil zu den auf Razor und Web Forms basierenden Websites ist ASP.NET Dynamic Data konzipiert. Trotzdem wird es als Website-Projekt angelegt. Es basiert auf der ▶

Tabelle 1: Microsofts Webtechnologien im Vergleich

	Web Forms	MVC	Single Page Application	Web Sites	Dynamic Data	ASP.NET Core	Web API	WCF Services
Betriebs-system	Windows	Windows	Windows	Windows	Windows	Windows, Mac, Linux	Windows	Windows
vorgestellt mit	.NET 1.0	.NET 3.5	.NET 4	.NET 1.1	.NET 2.0	.NET Core 1.0	.NET 3.5	.NET 2.0
Framework	.NET 4.7	.NET 4.7	.NET 4.7	.NET 4.7	.NET 4.7	.NET 4.7 / .NET Core 1.0	.NET 4.7	.NET 4.7
Plattformen	.NET Framework	.NET Framework	.NET Framework	.NET Framework	.NET Framework	.NET Framework / .NET Core	.NET Framework / .NET Core	.NET Framework
Verwen-dungszweck	Web-UI	Web-UI	Web-UI	Web-UI	Web-UI	Web-UI / Data Services	Data Services	Data Services
Anwen-dungstyp	Web-anwendung	Website, Web-anwendung	Web-anwendung	Website	Backend	Website, Web-anwendung	Datendienste	Datendienste
Projekttyp	UI- und daten-orientiert	UI-orientiert	UI-orientiert	UI-orientiert	daten-orientiert	UI-orientiert	schnitt-stellen- und daten-orientiert	schnitt-stellen- und daten-orientiert
App-Größe	*****	*****	*	**	*	*****	***	***
Einstieg	****	***	**	****	**	***	***	***
Customizing	****	****	****	**	***	****	****	**
Dokumen-tation	*****	*****	**	***	***	***	*****	**
Community	*****	*****	**	*	*	****	****	*

Web-Forms-Technologie und ist dafür gedacht, schnell und einfach Formulare über bestehende Datenbanken zu legen. Die Websites sind komplett UI-orientiert, Dynamic Data dagegen datenorientiert.

Dynamic Data gründet auf dem Entity Framework. Es erzeugt die Nutzeroberfläche automatisch, wobei es das Schema einer bestehenden Datenbank oder eines bestehenden Entity-Framework-Kontextes zugrunde legt.

Wer sehr schnell und ohne Aufwand eine Benutzeroberfläche für eine Datenbank erstellen will/muss, ist mit ASP.NET Dynamic Data gut beraten. Zum Beispiel kann es hilfreich sein, beim Entwickeln einer E-Commerce-Lösung vorerst auf ein umfangreiches Backend zu verzichten und sich nur auf das Frontend zu konzentrieren. Um dennoch Daten bequemer pflegen zu können, kann man Dynamic Data einsetzen, um schnell eine generische Oberfläche zu einer Datenbank zu erstellen.

Nur in den seltensten Fällen allerdings werden Dynamic-Data-Applikationen länger verwendet oder in umfangreichen Backends eingesetzt. Denn sobald die Oberfläche anzupassen ist oder ein spezielles Verhalten, Validierungen et cetera eingebaut werden müssen, wird es schnell komplex. In den meisten Fällen empfiehlt es sich dann, die Anwendung neu mit ASP.NET Web Forms oder ASP.NET MVC zu erstellen.

ASP.NET Core

Im Sommer 2016 wagte Microsoft einen radikalen Schritt. Nicht nur dass mit .NET Core eine neue Variante des .NET Frameworks neu geschrieben wurde, Microsoft hat sie auch als Open Source auf GitHub veröffentlicht. So geschah es auch mit ASP.NET Core, das sowohl auf .NET Core als auch auf dem vollständigen .NET Framework laufen kann. Der Fokus der Neuentwicklung liegt klar auf der Kompatibilität zu Azure und auf der Plattformunabhängigkeit. ASP.NET Core läuft auf verschiedenen Linux-Derivaten, auf Mac und natürlich auf Windows.

Ein weiterer Vorteil für die Neuentwicklung war, dass Microsoft nun endlich ein Framework hatte, das nicht die Datei *System.Web.dll* voraussetzt wie alle anderen Frameworks. Immerhin hat diese Datei Altlasten aus dem Framework 1.0 im Schlepptau, die aus Kompatibilitätsgründen nie abgelöst werden konnten.

ASP.NET Core ist nun modular aufgebaut und basiert auf einem Framework, das im Grunde genommen aus mehreren austauschbaren NuGet-Paketen zusammengesetzt ist. So kann Microsoft (oder auch der Entwickler) ganze Teile des Frameworks austauschen und nicht benötigte Teile ganz weglassen.

ASP.NET Core ist dadurch sehr leichtgewichtig und stark konfigurierbar. Eine ASP.NET-Core-Anwendung kann ihr ei-

genes Framework mitbringen, das genau auf die Art zusammengesteckt ist, wie sie es benötigt. Auf diese Weise kann ASP.NET Core unabhängig von installierten Frameworks sein. Es baut auf dem MVC-Muster auf, so ist das Entwickeln nahezu identisch zu ASP.NET MVC. Web Forms wird nicht von ASP.NET Core unterstützt und es gibt auch keine Pläne, dies in Zukunft zu tun.

Eine andere Besonderheit ist, dass ASP.NET Core die visuelle UI- und die REST-Service-Ebene gleichzeitig unterstützt. ASP.NET Web API (siehe nächster Abschnitt) ist zu einem Teil des ASP.NET-MVC-Frameworks geworden. Das ergibt Sinn, wenn man bedenkt, dass beide Frameworks das gleiche Entwurfsmuster nutzen. Der Unterschied zwischen MVC und Web API besteht nur noch in der Rückgabe: Wird eine View zurückgegeben, ist es ASP.NET MVC; kommen Daten zurück, so handelt es sich um Web API und die Daten gehen im JSON-Format an den Client.

Durch die Neuentwicklung konnte Microsoft auch einige Dinge verbessern. So ist die Konfiguration der Anwendung deutlich flexibler geworden. Ein Dependency-Injection-Container ist fest eingebaut und wird intensiv genutzt. Es ist nun nicht mehr möglich, ohne dieses Standardmuster zu arbeiten. Die Razor-Engine hat mit View-Komponenten und Tag-Helfern einige nützliche Neuerungen erhalten.

ASP.NET Core ist wie ASP.NET MVC und ASP.NET Web Forms für alle Größen von Anwendungen geeignet und bietet dank seiner plattformübergreifenden Fähigkeiten den Vorteil, auf mehr als einem System zu laufen. Auch dem Einsatz in Docker-Containern steht nichts im Weg. So kann ASP.NET Core ideal in Micro-Service-Architekturen zum Einsatz kommen.

ASP.NET Web API

Dieser und der nächste Abschnitt behandeln keine ASP.NET Frameworks mehr, die mit grafischen UIs arbeiten. Vielmehr geht es um Schnittstellen zum Datenaustausch. ASP.NET Web API ist eine davon. Web API bietet die Möglichkeit, REST- oder REST-ähnliche Schnittstellen zu entwerfen sowie Daten in verschiedenen Varianten auszugeben oder zu empfangen; die Standardformate sind XML und JSON.

Wie auch ASP.NET MVC setzt ASP.NET Web API auf das MVC-Muster. Daher sind beide Frameworks sehr ähnlich aufgebaut. Das ist auch der Grund, warum ASP.NET Core beide Frameworks zu einem einzigen zusammenfasst.

Eine andere Parallele zu ASP.NET MVC liegt darin, dass Web API sehr stark mit dem HTTP-Protokoll und dessen Verben arbeitet:

- *GET*: fordert Daten an
- *POST*: aktualisiert Daten
- *PUT*: legt Daten an
- *DELETE*: löscht Daten
- *PATCH*: aktualisiert Teildaten
- *PROPPATCH*: bearbeitet Eigenschaften
- etc.

Web API ist sehr flexibel und erlaubt es, sehr schnell einfache Schnittstellen bereitzustellen. Aber es ist ebenso möglich, mit etwas mehr Aufwand echte REST-Schnittstellen anzubieten. Auch Authentifizierung und Validierung sind möglich.

ASP.NET Web API lässt sich sowohl in MVC- als auch in Web-Forms-Projekten einbauen und verwenden, so können Nutzeroberfläche und Datenschnittstelle das gleiche Webprojekt und intern die gleichen Logiken nutzen für den Fall, dass dies sinnvoll sein sollte.

WCF Services

Die Dienste der Windows Communication Foundation (WCF) sind eine relativ alte, aber immer noch weit verbreitete Technik für den Datenaustausch. WCF nutzt in der Regel Remote Procedure Call als Verfahren. Dabei werden entfernte Methoden aufgerufen, die entweder Aktionen ausführen oder Daten liefern.

WCF Services sind in der Anbindung und in der Bereitstellung recht einfach zu erstellen und zu verstehen. Erst beim Anpassen wird es etwas aufwendiger. WCF Services liefern SOAP-basierte HTTP-Dienste aus, unterstützen aber auch andere Protokolle und Transportebenen. Allerdings ist es etwas komplexer, andere Protokolle bereitzustellen, und es sind umfangreiche Konfigurationen erforderlich.

REST- und JSON-basierte Dienste lösen die klassischen SOAP-basierten Dienste mehr und mehr ab. Dennoch gibt es nach wie vor viele Systeme, die noch SOAP konsumieren oder SOAP-Services anbieten. Vom Tisch ist das Thema also nicht, auch wenn sich das viele Entwickler wünschen und die wesentlich einfacheren REST-basierten Dienste vorziehen.

Vergleich

Tabelle 1 enthält einen kompakten Vergleich der hier vorgestellten Webtechnologien. Anhand dieser Tabelle können Sie relativ schnell entscheiden, welche Technologie für welches Projekt geeignet ist. Die Sternbewertungen sind so zu lesen, dass mehr Sterne „besser“ bedeuten und weniger Sterne eher „schlechter“. Die Bewertungen der Technologien und Tools berücksichtigen dabei immer auch, wie groß und aktiv die jeweiligen Communities sind. Das ist in der Regel ein Indikator dafür, wie viel Unterstützung man auf StackOverflow, GitHub, in Foren oder auch in Form von Blog-Artikeln erhält. Dies ist speziell für Anfänger wichtig und empfehlenswert. Gerade bei Open-Source-Projekten ist eine aktive Community ein Garant für schnelles Bugfixing und stetige Weiterentwicklung. ■

[1]Knockout, http://knockoutjs.com

Jürgen Gutsch
ist Software Developer, Berater, Trainer und freier Autor. Er betreibt einen Blog auf http://asp.net-hacker.rocks/, engagiert sich in der .NET-Community und wurde mehrfach als MVP ausgezeichnet. Sie erreichen ihn unter **juergen@gutsch-online.de**

dnpCode	A1707WebVergleich