Maarten Balliauw

Weg met postbacks!

ASP.NET MVC FRAMEWORK ALS ALTERNATIEF VOOR WEBFORMS

In .NET Magazine #19 kon je al iets lezen over Monorail, een MVC Framework voor ASP.NET. Sinds december 2007 heeft Microsoft een eigen oplossing, die op het moment van schrijven nog volop in ontwikkeling is. In dit artikel beschrijft de auteur hoe je ASP.NET. MVC Framework kunt gebruiken om een gastenboekapplicatie te bouwen.

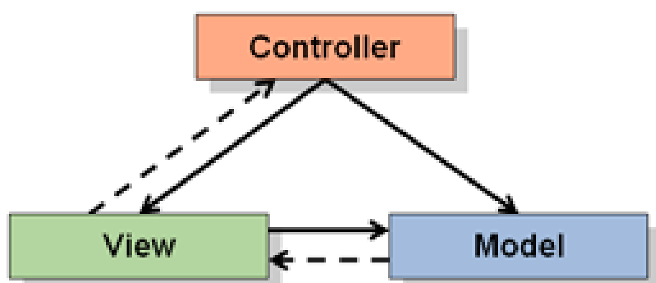
Als je vertrouwd bent met design patterns, heb je vast al wel gehoord van het MVC-pattern (model-view-controller). MVC is een volwassen architectuur voor het bouwen van de presentatielaag in een (web)applicatie en kent drie componenten, elk met hun eigen verantwoordelijkheden:

- Model: deze component is verantwoordelijk voor de 'state', die ook vaak in een database wordt bewaard. Een klasse Product in het model kan zo mappen naar de tabel Producten in een database.
- View: deze component is verantwoordelijk voor de weergave van de user-interface. Meestal mapt deze op het model: voor de klasse Product is er in de view een aantal labels en textboxes dat de data weergeeft.
- Controller: deze component is verantwoordelijk voor alle interactie. De controller bepaalt of het model aangepast moet worden (bijvoorbeeld een nieuwe naam), en of er eventueel een andere view voor moet worden geladen en weergegeven.

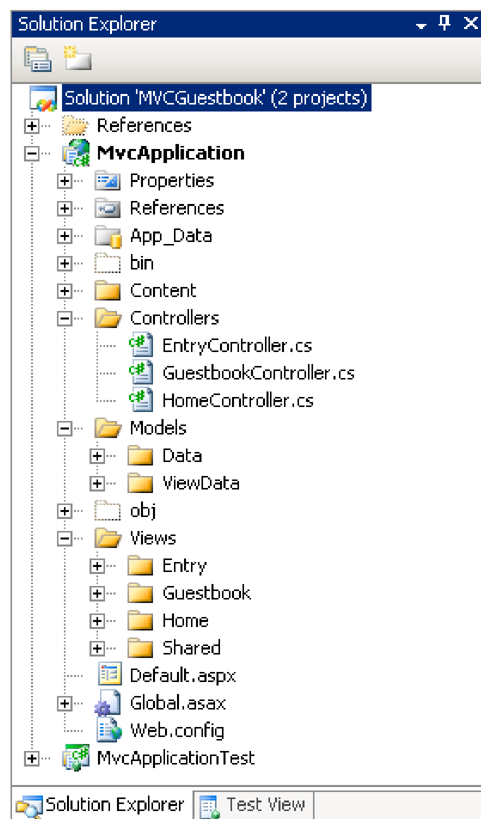
De controller handelt dus alle interactie met de gebruiker af en zorgt eventueel voor het inladen van een nieuwe view (routing). De view wordt enkel gebruikt voor de rendering van gegevens. Het handige aan deze manier van werken is dat je gemakkelijk nieuwe views kunt toevoegen, zonder het model aan te passen. Iedere view kan zich registreren bij het model, dat op zijn beurt alle views waarschuwt wanneer er een update van het model is geweest. Zo kan elke view zelf bepalen wat er moet gebeuren als er een update binnenkomt, zie afbeelding 1. Bijkomend voordeel is dat unit-tests kunnen worden geschreven op de controller. Zo kun je perfect testen hoe de controller en het model reageren op bepaalde interactie met de gebruiker via een unit-test.

ASP.NET MVC versus ASP.NET WebForms

Een webapplicatie op zich is steeds 'stateless'. Dit houdt in dat een client een request naar de server stuurt en een response terugkrijgt. Iedere volgende request is steeds volledig nieuw: tussen de requests wordt geen state bijgehouden. Microsoft heeft met ASP.NET WebForms een goede poging gedaan om dit aan te pakken. Door gebruik te maken van ViewState en postbacks, wordt dit stateless karakter van een webapplicatie voor de ontwikkelaar verborgen. Helaas brengt dit vaak problemen en ergernis met zich mee. De complexiteit en regels waaraan dit WebForms-model is gebonden (page life-cycle), hebben vaak als resultaat dat een ontwikkelaar met veel zaken rekening moet houden en vaak allerlei omwegen moet zoeken om een bepaalde actie te kunnen uitvoeren. Ter illustratie: als je op het internet zoekt naar 'viewstate-probleem', krijg je al snel meer dan 200.000 resultaten. Het ASP.NET MVC Framework ondersteunt geen ViewState en postbacks, en beschouwt een url ook niet als 'endpoint' van een webpagina die gecompileerd moet worden. Waar je bij WebForms een postback doet naar een url, doe je in het ASP.NET MVC Framework eigenlijk een post naar een centrale handler. Deze handler zorgt voor de verdere verwijzing naar de juiste controller



Afbeelding 1. Schematisch overzicht van het MVC pattern



Afbeelding 2. Overzicht van een ASP.NET MVC-project in Visual Studio

in het MVC-model; zie afbeelding 2. Het idee achter deze centrale handler is het REST-pattern (REpresentational State Transfer). Volgens dit pattern dient een applicatie te bestaan uit resources, die via een url-syntax worden aangesproken. Op die manier zal een url als http://www.mijnsite.com/products/list/hardware/ in het ASP.NET MVC Framework 'routen' naar de *List*-methode in een *ProductsController*, waar als parameter 'hardware' wordt meegegeven.
Met het MVC-model, dat overigens geen vervanger wordt voor het WebForms-model, wil Microsoft het volgende bereiken:

- Scheiding van verantwoordelijkheden
- Eenvoudig testen van webapplicaties
- Uitbreidbaarheid: je kunt bijvoorbeeld een eigen view-engine of routing-policy implementeren.
- Ondersteuning voor SEO (Search Engine Optimalization) wordt beter dankzij een krachtige rewrite-engine. Zo kun je bijvoorbeeld een url als http://www.mijnsite.com/products/list/hardware/ mappen naar een bepaalde view die door de *ProductenController* wordt geladen.
- De huidige ASP.NET-aspx, -ascx en -masterpages worden ondersteund als view template. Het idee van page life-cycle, postbacks en viewstate wordt overboord gegooid: iedere view post naar een controller die bepaalt wat er verder moet gebeuren.
- Alle huidige ASP.NET-providers worden ondersteund: authentication, authorization, profiles, output caching, sessies, configuratiebeheer.

Aan de slag!

In dit artikel wordt een gastenboekapplicatie gebouwd. Deze applicatie ondersteunt meerdere gastenboeken waarin bezoekers een korte reactie kunnen schrijven. Een gebruiker 'Administrator' kan eventueel deze reacties verwijderen. De voorbeeldcode is gebaseerd op CTP 2 van het ASP.NET MVC Framework (maart 2008). Bij publicatie van dit artikel is er mogelijk al een nieuwere versie beschikbaar waardoor code kan afwijken. De laatste broncode kun je, indien aangepast, terugvinden op mijn blog: http://blog.maartenballiauw.be/page/ASPNET-MVC-Guestbook.aspx. Na installatie van de nodige componenten en Visual Studio-templates kun je een nieuwe solution aanmaken, op basis van het ASP.NET MVC Web Application-template. Dit template maakt een solution met daarin twee projecten: de applicatie zelf en een testproject.

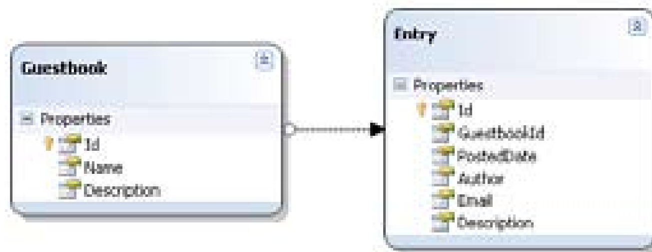
In het webapplication-project zie je enkele automatisch aangemaakte mappen:

- Views: hierin kun je de views aanmaken. Deze bestaan onder meer uit .aspx-, .ascx-, en .master-bestanden zoals je dit gewend bent van WebForms. Je ziet ook enkele subfolders. Het ASP.NET MVC Framework probeert telkens de juiste view bij iedere controller te zoeken op basis van deze subfolders. De *HelloController* zal zijn views bijvoorbeeld gaan zoeken in de map /Views/Hello/. De map Common die je ziet, kan gebruikt worden voor opslag van bijvoorbeeld CSS-bestanden en afbeeldingen.
- Controllers: hier worden controllers geplaatst.
- Models: alle klassen die het applicatiemodel van de applicatie ondersteunen. Dit zal vaak code zijn die met de businesslaag in jouw applicatie gaat communiceren.
- In het webapplication-project wordt ook een aangepaste web.config geplaatst die de nodige modules registreert en een Global.asax, waarin de routes beschreven worden.

```
routes.Add(new Route("{controller}/{action}/{id}",
new MvcRouteHandler())
{
  Defaults = new RouteValueDictionary(new { action = "Index", id = ""
}),
});

routes.Add(new Route("Default.aspx", new MvcRouteHandler())
{
  Defaults = new RouteValueDictionary(new { controller = "Home", a
ction = "Index", id = "" }),
});
```

Codevoorbeeld 1.



Afbeelding 3. Het ontsluiten van data met behulp van LINQ to SQL

Routing

In Global.asax wordt een routetabel opgebouwd, waarmee het ASP.NET MVC Framework requests zal routeren naar de juiste controller.
In codevoorbeeld 1 zie je dat er twee standaard routes zijn aangemaakt. In de eerste zie je dat een url wordt opgesplitst in drie delen: {controller}/{action}/{id}. De variabelen tussen { en } worden als parameters gebruikt voor de geconfigureerde action op de controller. Als een van deze variabelen niet is ingevuld, worden de waarden uit *Defaults* gebruikt.
Een tweede standaard route is de url 'Default.aspx', die gerouteerd wordt naar de *Index*-methode van *HomeController*.
In codevoorbeeld 2 zie je een stuk code uit de klasse *HomeController*. Deze klasse bevat een methode *About* die niets meer doet dan de view voor About aanroepen. Door de configuratie in Global.asax wordt een gebruiker die naar de URL /Home/About surft, gerouteerd naar deze *About*-methode van *HomeController*.

Databinding

Het voorgaande voorbeeld is eenvoudig. We doen niet aan databinding, form posts, enzovoort, wat toch zaken zijn die in bijna iedere applicatie voorkomen. We maken gebruik van LINQ to SQL om onze data te ontsluiten via het volgende model; zie afbeelding 3.
De eenvoudigste manier om data van model naar view te ontsluiten via de controller, is door gebruikmaken van het *ViewData*-object, een *Dictionary* die in iedere controller en

```
using System;
using System.Web;
using System.Web.Mvc;
using MvcApplication.Models.Data;
using MvcApplication.Models.ViewData;

namespace MvcApplication.Controllers
{
  public class HomeController : Controller
  {
    public void About()
    {
      RenderView("About");
    }
  }
}
```

Codevoorbeeld 2.

view beschikbaar is. In codevoorbeeld 3 zie je hoe deze wordt gebruikt binnen de controller. Aan de kant van de view kon je deze *ViewData* eenvoudig uitlezen:

```
<% if (ViewData["ErrorMessage"] != null) { %>
<h3><%=ViewData["ErrorMessage"].ToString()%></h3>
<% } %>
```

In dit voorbeeld gaan we er van uit dat deze ErrorMessage een string is. Het mooie aan het ASP.NET MVC Framework is, dat het mogelijk is om van *ViewData* een strong-typed object te maken. Het volstaat om in de codebehind van de view een generic toe te voegen:

```
public partial class Show : ViewPage<GuestbookDetails> { }
```

Het *ViewData*-object is nu van het type *GuestbookDetails*. In codevoorbeeld 4 zie je hoe we een *Guestbook*-object doorgeven aan de view waar we via *ViewData.Guestbook* kunnen werken met het *Guestbook*-object. Visual Studio's IntelliSense speelt dit spel overigens mee, wat erg handig is bij het ontwikkelen van een view. Als je ontwikkelt in het ASP.NET MVC Framework kun je uiteraard ook de standaard ASP.NET-controls gebruiken op voorwaarde dat ze niet afhankelijk zijn van ViewState. Het ASP.NET-team werkt op dit moment volop aan een set controls voor het MVC Framework. Deze controls worden enkel ontwikkeld als het nodig is. De quote "We've got a repeater control, it's called a foreach loop" is het bewijs van deze gedachtegang.

Formulieren

Gegevens op een formulier moeten vaak gebruikt kunnen worden in programmacode. Binnen het ASP.NET MVC Framework bouw je een formulier het beste op aan de hand van enkele standaarden die allerlei acties vereenvoudigen. Allereerst moet je aan de client-kant een formulier opbouwen:

```
<form action="<%=Html.BuildUrlFromExpression<EntryController>(c =>
c.Preview(ViewData.Guestbook.Id))%>" method="post">
```

Dit formulier zal worden gepost naar de *Preview*-methode van *EntryController*. De *html*-klasse bouwt deze koppeling op aan de hand van een LINQ expression: *c => c.Preview(ViewData.Guestbook.Id)*. Het voelt erg natuurlijk aan om op deze manier een koppeling te creëren. Op het formulier zelf kunnen we velden aanmaken: *<%= Html.TextBox("Entry.Author", "") %>*. De *html*-klasse bouwt een HTML TextBox met de naam *Entry.Author*. Deze naamgeving is van belang, omdat we daarmee in de controller geposte gegevens kunnen inlezen; zie codevoorbeeld 5. De methode *BindingHelperExtensions.UpdateFrom()* kopieert gegevens uit overeenkomende velden naar het *Entry*-object.

Unit-testing

Voor het testen van een webapplicatie maak je normaal grotendeels gebruik van webtests. Deze geautomatiseerde tests doen requests op een server en valideren er of bepaalde data foutloos terugkomen. Het testen van een MVC-webapplicatie is eenvoudiger. Hier kun je simpelweg unit-tests schrijven op de methodes van jouw controller en op die manier controleren of alle broncode aan de gestelde eisen voldoet. Andere voordelen zijn dat ook code-coverage kan worden bepaald op controllers. Ook de snelheid van testen neemt toe! Er hoeft immers geen webtest gestart te worden (server starten, deployen, uitvoeren, ...). Uiteraard kun je deze unit-tests nog steeds aanvullen met een klassieke webtest. In codevoorbeeld

```
using System;
using System.Web;
using System.Web.Mvc;
using System.Web.Security;

namespace MvcApplication.Controllers
{
  public class LoginController : Controller
  {
    // ...
    public void Authenticate()
    {
      // ...
      // Authenticate
      if (MembershipProviderInstance.Authenticate(userName, password))
      {
        // ...
      }
      else
      {
        // Set error...
        ViewData.Add("ErrorMessage", "Invalid credentials! Please verify
          your username and password.");
        // Render index page
        RenderView("Index");
      }
    }
  }
}
```

Codevoorbeeld 3.

```
using System;
using System.Web;
using System.Web.Mvc;
using System.Linq;
using MvcApplication.Models.Data;
using MvcApplication.Models.ViewData;

namespace MvcApplication.Controllers
{
  public class GuestbookController : Controller
  {
    public void Show(int id)
    {
      GuestbookDataContext ctx = new GuestbookDataContext();

      GuestbookDetails viewData = new GuestbookDetails
      {
        Guestbook = ctx.Guestbooks.Single(g => g.Id == id)
      };

      RenderView("Show", viewData);
    }
  }
}
```

Codevoorbeeld 4.

```
public void Preview(int guestbookId)
{
  // ...

  Entry entry = new Entry();
  BindingHelperExtensions.UpdateFrom(entry, Request.Form);
  entry.PostedDate = DateTime.Now;
  entry.GuestbookId = guestbookId;

  // ...
}
```

Codevoorbeeld 5.

```
[TestMethod]
public void TestRouteDefault()
{
   MockRepository mocks = new MockRepository();
   HttpContextBase context;

   using (mocks.Record())
   {
      context = mocks.FakeHttpContext();
      context.Request.SetupRequestUrl("~/Default.aspx");
   }

   using (mocks.Playback())
   {
      RouteData routeData = routes.GetRouteData(context);
      Assert.AreEqual("Home", routeData.Values["controller"]);
      Assert.AreEqual("Index", routeData.Values["action"]);
      Assert.AreEqual(typeof(MvcRouteHandler),
         routeData.RouteHandler.GetType());
   }
}
```

Codevoorbeeld 6.

```
[TestMethod]
public void TestPreview()
{
   EntryController controller = new EntryController();
   var fakeViewEngine = new FakeViewEngine();
   controller.ViewEngine = fakeViewEngine;

   MockRepository mocks = new MockRepository();
   using (mocks.Record())
   {
      mocks.SetFakeControllerContext(controller);
      controller.HttpContext.Request.SetupFormParameters();
   }
   using (mocks.Playback())
   {
      controller.HttpContext.Request.Form.Add(
         "Entry.Author", "TestUser");
      controller.HttpContext.Request.Form.Add
         ("Entry.Email", "test@test.com");
      controller.HttpContext.Request.Form.Add
         ("Entry.Description", "This is a test");

      controller.Preview(1);
      Assert.AreEqual("New", fakeViewEngine.ViewContext.ViewName);
      Assert.AreEqual(1, ((CreateEntry)fakeViewEngine.ViewContext.
         ViewData).Guestbook.Id);
      Assert.AreEqual(1, ((CreateEntry)fakeViewEngine.ViewContext.
         ViewData).Entry.GuestbookId);
      Assert.AreEqual("TestUser", ((CreateEntry)fakeViewEngine.
         ViewContext.ViewData).Entry.Author);
      Assert.AreEqual("test@test.com", ((CreateEntry)fakeViewEngine.
         ViewContext.ViewData).Entry.Email);
      Assert.AreEqual("This is a test", ((CreateEntry)
fakeViewEngine.ViewContext.ViewData).Entry.Description);
   }
}
```

Codevoorbeeld 7.

6 heb ik een unit-test geschreven voor de route naar 'Default.aspx'. Deze unit-test maakt gebruik van een mocking-framework (Rhino mocks) waarmee on-the-fly complexe objecten als een HttpRequest kunnen worden gesimuleerd. Deze test *TestRouteDefault* start een (gesimuleerde) request naar de pagina Default.aspx en controleert daarna of de teruggegeven controller en action wel degelijk *HomeController* en *Index* zijn.

Uiteraard kan ook de werking van een pagina getest worden. In codevoorbeeld 7 wordt de werking van *EntryController.Preview()* getest aan de hand van 'fake' formuliergegevens.

What's next?

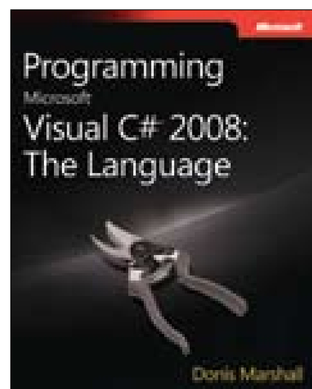
Hopelijk heb ik je warm kunnen maken voor deze veelbelovende technologie. In de voorbeeldapplicatie vind je nog allerlei codevoorbeelden rond formulieren, navigatie, security en testing, waarmee je een snelle start kunt maken. Verspreid over het internet zijn er nog veel extra tutorials en code-add-ons te vinden.

Maarten Balliauw is software-engineer bij Dolmen (www.dolmen.be) en houdt zich bezig met webontwikkeling in ASP.NET en PHP en application life-cycle management met Team Foundation Server. Zijn blog vind je op www.maartenballiauw.be.

Referenties

ASP.NET 3.5 extensions: http://asp.net/downloads/3.5-extensions/
ASP.NET MVC Framework Source Code: http://www.codeplex.com/aspnet
ASP.NET MVC Framework CTP 2: http://www.microsoft.com/downloads/details.aspx?FamilyId=38CC4CF1-773A-47E1-8125-BA3369BF54A3
Scott Hanselman video tutorials: http://www.asp.net/learn/3.5-extensions-videos/
Scott Guthrie's tutorials (CTP 1): http://weblogs.asp.net/scottgu/archive/2008/02/12/asp-net-mvc-framework-road-map-update.aspx
"We've got a repeater control, it's called a foreach loop.": http://theq ueue.net/blog//blog/archive/2008/02/14/rocknug-asp.net-mvc-asprepeater.aspx
MVC Framework – Security: http://blog.maartenballiauw.be/post/2007/12/ASPNET-MVC-framework---Security.aspx
Rhino mocks: http://www.ayende.com/projects/rhino-mocks.aspx
Laatste voorbeeldcode: http://blog.maartenballiauw.be/page/ASPNET-MVC-Guestbook.aspx

(advertentie MS Press)



Programming Microsoft Visual C# 2008: The Language
ISBN: 9780735625402
Auteur: Donis Marshall
Pagina's: 784

Programming Microsoft Office Business Applications
ISBN: 9780735625365
Auteurs: Steve Fox; Rob Barker; Paul Stubbs; Joanna Bichsel; Erika Ehrli Cabral
Pagina's: 256