Volledige controle over je webapplicatie

Microsoft ASP.NET MVC framework

Geert van der Cruijsen

De webontwikkelwereld biedt veel verschillende frameworks die voornamelijk gebouwd zijn op het Model View Controller designpattern. Voor PHP is er het Zend framework, Java heeft Struts en Ruby on Rails is ook een bekend en veelgebruikt MVC framework. Microsoft had tot op heden geen vergelijkbaar framework en begon daarom in december 2007 met de ontwikkeling van het Microsoft ASP.NET MVC framework.

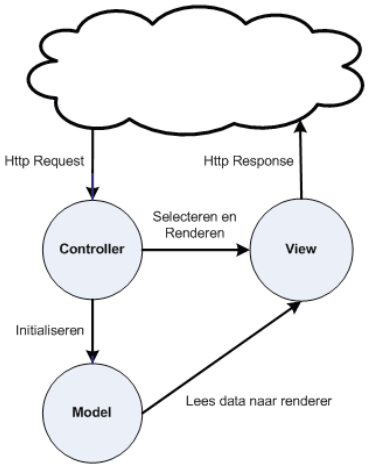
In april is versie 1.0 van dit framework uitgekomen. Zijn ASP.NET Webforms nu verleden tijd en wat biedt dit nieuwe webframework voor ontwikkelaars?
Het Model View Controller (MVC) Design Pattern is een van de bekendste designpatterns. Het gebruik van het MVC pattern zorgt voor een scheiding tussen business-logica en de user interface-logica. Toepassing van het MVC pattern resulteert in een onderhoudbare en testbare applicatie waar verschillende onderdelen los van elkaar staan, zodat ze eenvoudig verwisseld kunnen worden met onderdelen van een andere implementatie. Omdat bijvoorbeeld de data-access losgekoppeld is, kan deze vervangen of gemockt worden: Dit maakt de applicatie onderhoudbaar en beter testbaar. Daarnaast gaat het MVC framework uit van 'Conventions over Configuration'. Dit houdt in dat er standaard een afgesproken patroon wordt gevolgd. Een voorbeeld hiervan is dat de views die bij een controller horen automatisch gevonden worden als ze in een directory gezet worden met de naam van de controller. Wil je hiervan afwijken, dan kun je dat nog wel configureren.

- Model: Het model is een representatie van de business objecten en rules en zorgt voor het bijhouden van de 'State' in de applicatie.
- View: Views zorgen voor het renderen van het model naar een User Interface in bijvoorbeeld HTML. Er kunnen meerdere views bestaan voor een model.
- Controller: De Controllers zorgen voor het verwerken van events en kunnen aanpassingen op het model aanroepen.

Voordelen
Het MVC framework biedt veel voordelen ten opzichte van de standaard ASP.NET Webforms. Doordat je veel meer controle hebt over de output is het eenvoudiger om leesbare nette HTML te genereren. Het MVC framework maakt gebruik van de nieuwe URL Routing functionaliteit die in het .NET 3.5 framework is toegevoegd sinds SP1, waardoor je meer controle hebt over hoe URLs worden opgebouwd. Nettere URLs zijn niet alleen eenvoudiger te onthouden door gebruikers, maar helpen ook bij Search Engine Optimization waardoor de site beter doorzocht kan worden door zoekmachines.

Verschil MVC framework/ASP.NET Webforms

Binnen het MVC framework is de pageflow anders dan bij ASP.NET webforms. In figuur 1 staat een schematische weergave van de pageflow die gebruikt wordt binnen het ASP.NET MVC framework. Als eerste komt een request van een client binnen op de controller. De controller voert daarna business operaties uit op het model. Het model geeft het resultaat van deze operaties terug aan de controller, de controller beslist welke view getoond moet worden en geeft deze view de benodigde data uit het model. De view genereert de output en stuurt deze terug naar de client.



FIGUUR 1: PAGEFLOW VAN EEN MVC APPLICATIE.

Waar bij het gebruik van ASP.NET webforms de http request binnenkomt op de aspx file (View) en deze view zorgt voor zowel de input als de output, is dit helemaal gescheiden in het MVC framework.

Test Driven Development

De testbaarheid van moderne webapplicaties is erg belangrijk en door het ontwikkelen van applicaties volgens Test Driven Development (TDD) is het mogelijk om goede testbare code te schrijven. Het MVC framework Is uitermate geschikt voor het bouwen van applicaties volgens TDD. Bij TDD schrijf je eerst je unittest en vanuit de unittest gaan we de functionaliteit die je in de test beschrijft ook daadwerkelijk implementeren.

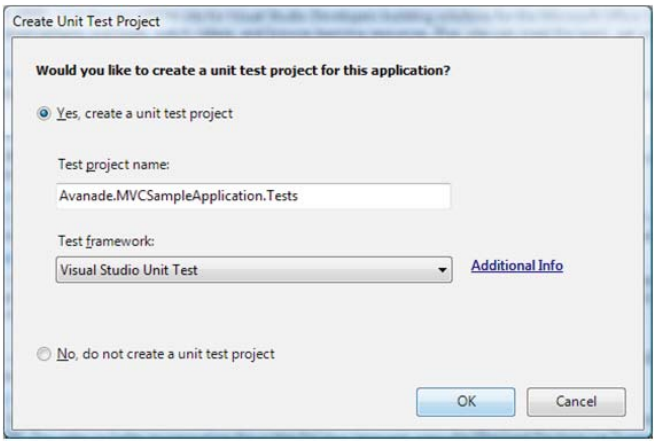
Het motto van TDD is: 'Red, Green Refactor' wat staat voor:

- Red: schrijf je unittest en zorg ervoor dat als je deze uitvoert de unittest faalt, omdat de functionaliteit nog niet is toegevoegd;
- Green: schrijf de classes in je unittests zo eenvoudig mogelijk, zodat de test slaagt;
- Refactor: Pas de code in je classes aan, zodat je nettere code gebruikt. Pas eventuele designpatterns toe of maak functions aan van stukken code die je op meerdere plekken gebruikt. Zorg er natuurlijk wel voor dat je testen blijven slagen.

Doordat er alleen functionaliteit gebouwd wordt die in de testen voorkomt, ben je er zeker van dat je nooit code schrijft die uiteindelijk niet in de rest van de applicatie gebruikt wordt. Een bijkomend voordeel van het schrijven van de tests voor het bouwen van de applicatie is, dat je er zeker van bent dat alle functionaliteit later ook getest wordt. Hierdoor kun je eenvoudig controleren of alles nog werkt nadat je wijzigingen hebt gemaakt in de bestaande code.

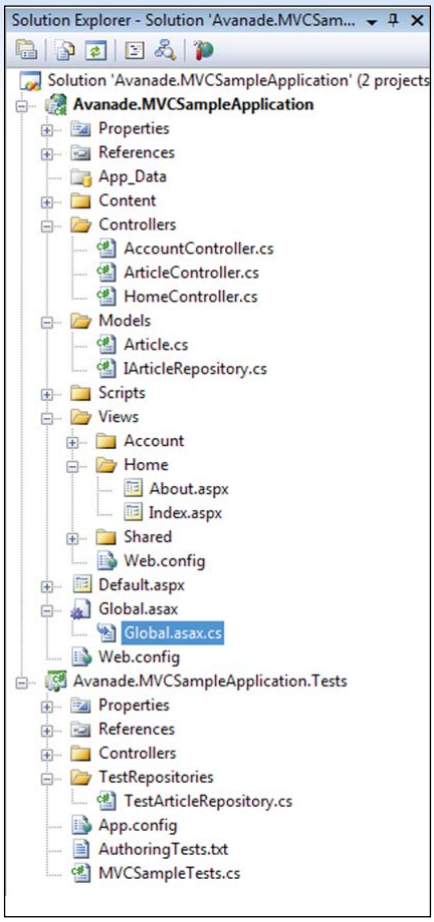
Het bouwen van een MVC webapplicatie

Bij het aanmaken van een nieuw MVC project worden er standaard een aantal files automatisch aangemaakt. Als eerste wordt na het aanmaken van een project altijd de vraag gesteld of er ook een unittest project moet worden aangemaakt. Hierbij kan je standaard kiezen voor het gebruik van het Visual Studio Unit-Test framework, maar als er andere unittest frameworks geïnstalleerd zijn zoals Nunit, dan komt deze in ook in de keuze lijst voor zoals zichtbaar in figuur 2.



FIGUUR 2: HET AANMAKEN VAN EEN UNITTEST PROJECT.

Naast het unittest project wordt een aantal mappen met specifieke bestanden aangemaakt om een start te kunnen maken met de applicatie. In de controller directory komen alle Controllers die volgens de standaard conventies altijd eindigen op Controller.cs. De views die bij een bepaalde controller horen staan altijd in de View\controllernaam directory. In figuur 3 zie je een voorbeeld van de bestandstructuur van een MVC applicatie.



FIGUUR 3: PROJECT STRUCTUUR VAN EEN MVC PROJECT.

Als we op een TDD manier een MVC Applicatie bouwen, beginnen we met het aanmaken van een testmethode. In codevoorbeeld 1 staat een testmethode beschreven voor het bouwen van een webapplicatie met daarin artikelen. De testmethode beschrijft het ophalen van een bepaald artikel.

```
[TestMethod]
public void Does _ ArticleRepository _ Return _ Expected _ Article _
Title _ Test()
{
    // Arrange
    Article article = new Article();
    article.Title = "TestArtikel";

    IArticleRepository repository = new TestArticleRepository();
    ArticleController articleController = new
ArticleController(repository);

    // Action
    ActionResult result = articleController.GetArticle(article.Tit-
le);

    // Assert
    Assert.IsInstanceOfType(result, typeof(ViewResult));
    ViewResult view = (ViewResult)result;
    Article resultArticle = (Article)view.ViewData.Model;
    Assert.AreEqual(article.Title, resultArticle.Title);
}
```

CODEVOORBEELD 1: TESTMETHODE VOOR HET TESTEN VAN EEN GET-ARTICLE.

Een goede unittest bestaat altijd uit drie onderdelen die worden beschreven als arrange, action en assert. Arrange staat voor het maken en initiëren van de variabelen. Bij action voeren we de te testen functionaliteit uit en bij assert controleren we de uitkomst met wat we verwachten.
Bij het schrijven van de testmethode moet een keuze gemaakt

worden over de opbouw van de webapplicatie. We hebben een entiteit nodig die het artikel object gaat bevatten (Article class) en we hebben iets nodig waar we de artikelen uit kunnen halen (IArticleRepository). Op de IArticleRepository maken we een methode genaamd GetArticle om een artikel mee uit de database te kunnen halen.
Bij het openen van 'http://localhost/Article/[Titel van artikel]' moet een artikel worden weergegeven. Hiervoor maak je een ArticleController met een ShowArticle Action Methode. Deze roepen we aan in het Action gedeelte van de test. De ArticleController geeft een View terug met daarin een Article object en deze kunnen we in de Assert fase controleren met de waarde die we verwachten.

Het model

In de applicatie wordt een Article object gebruikt dat in het MVC framework ons Model is. De Article class bevat een aantal properties die het business object Article beschrijven (codevoorbeeld 2).

```
public class Article
{
    public Guid ID { get; set; }
    public string Title { get; set; }
    public string Content { get; set; }

    public Article() { }
}
```

CODEVOORBEELD 2: ARTICLE CLASS.

De IArticleRepository is een interface die door verschillende classes geïmplementeerd kan worden. Sla je de gegevens bijvoorbeeld op in MS-SQL dan kun je een SQLArticleRepository bouwen, maar voor het voorbeeld en de test gebruiken we hier een TestArticleRepository die hardcoded waardes teruggeeft. Zie codevoorbeeld 3 voor de code van IArticleRepository en TestArticleRepository. IArticleRepository heeft een GetArticle methode die een Article teruggeeft. In de TestArticleRepository wordt deze functie geïmplementeerd en wordt het artikel opgehaald uit de bron waar die specifieke repository voor is gemaakt.

```
public interface IArticleRepository
{
    Article GetArticle(string Title);
}

class TestArticleRepository : IArticleRepository
{
    #region IArticleRepository Members

    public Article GetArticle(string Title)
    {
        Article article = new Article();
        article.Title = Title;
        article.Content = "Dit is een test Artikel uit de TestArticleRepository";
        article.ID = Guid.NewGuid();

        return article;
    }

    #endregion
}
```

CODEVOORBEELD 3: IARTICLEREPOSITORY EN TESTARTICLEREPOSITORY.

URL Routing

Een van de belangrijkste onderdelen van het MVC framework is URL routing. Hoe zorgen we ervoor dat we bij de aanroep van http://localhost/Article/ArticleTitle uitkomen bij de ArticleController?
Het Global.asax bestand bevat een routetabel in de static methode RegisterRoutes. De routetabel in RegisterRoutes beschrijft een aantal URLs met daarbij de controller die het request moet afvangen. In codevoorbeeld 4 staat de standaardroute die bij het aanmaken van een MVC project wordt gegenereerd.

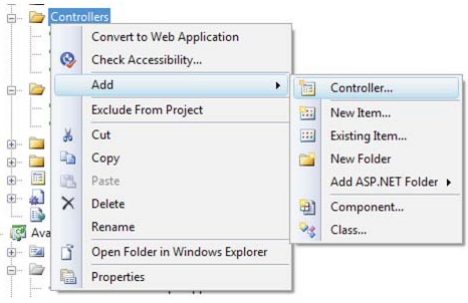
```
routes.MapRoute(
    "Default",
    "{controller}/{action}/{id}",
    new { controller = "Home", action = "Index", id = "" }
);

routes.MapRoute(
    "ArticleByTitle",
    "Article/{title}",
    new { controller = "Article", action = "GetArticle", id = ""}
);
```

CODEVOORBEELD 4: ROUTES VAN HET MVC FRAMEWORK.

Het MVC framework zal bij een aanroep proberen om de URL te matchen aan een patroon dat in de routetabel aanwezig is. Bij de aanroep van http://localhost/Article/GetArticle/Title zal deze routetabel de controller naar de ArticleController mappen, de action is GetArticle en id wordt Title. De laatste parameter van MapRoute zijn de default waardes. Is in de URL de action en id niet meegegeven dan zal met deze route de Index methode worden aangeroepen met een lege id.
De standaardroute voldoet in de meeste webapplicaties voor bijna alle gevallen. Om een URL als http://localhost/Article/Title te kunnen krijgen, moet een andere route worden toegevoegd. Een voorbeeld hiervan zie je in codevoorbeeld 4.

Het model gebruiken in de Controller

De article controller erft van de Controller class uit het MVC framework. De ShowArticle methode krijgt als parameter een string met daarin de titel mee en haalt het betreffende artikel uit de ArticleRepository. Dit artikel voegen we toe aan de viewdata zodat de view het artikel in HTML kan renderen.
Je maakt een nieuwe Controller aan door met de rechtermuisknop op de controllers directory te klikken en voor 'Add New Controller' te kiezen zoals te zien in figuur 4.



FIGUUR 4: HET TOEVOEGEN VAN EEN NIEUWE CONTROLLER.

In de unittest wordt de GetArticle methode van de ArticleController aangeroepen, dus deze kan nu automatisch worden aangemaakt door via de refactor tools van Visual Studio met de rechtermuisknop te klikken op de ArticleController.GetArticle aanroep in de unittest en te kiezen voor 'Generate Methode Stub'. Er wordt nu een methode aangemaakt in de ArticleController class. Om de code compileerbaar te maken, moet ook een nieuwe

constructor aangemaakt worden in de ArticleController waarin een IArticleRepository als parameter wordt meegegeven.
De code compileert nu en als de unittest uitgevoerd wordt, dan faalt de test. Dit is de 'Red'-fase van de TDD methode. Om van 'Red' naar 'Green' te gaan moet de code worden geschreven die de juiste gegevens uit de ArticleRepository haalt en deze meegeeft aan de view. In codevoorbeeld 5 is de code van de GetArticle methode uit de ArticleController te zien. Na het opnieuw uitvoeren van de test zal de test slagen zoals zichtbaar in figuur 5.

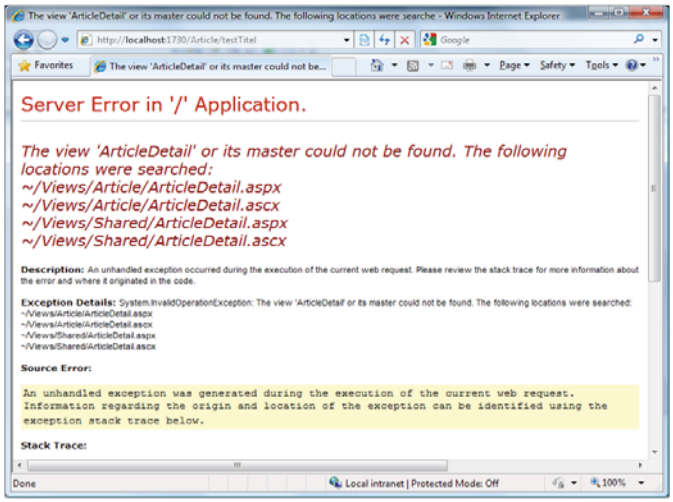
```
public ActionResult GetArticle(string title)
{
    Article article = _repository.GetArticle(title);
    return View("ArticleDetail", article);
}
```

CODEVOORBEELD 5: CODE VAN DE GETARTICLE METHODE UIT DE ARTICLECONTROLLER.

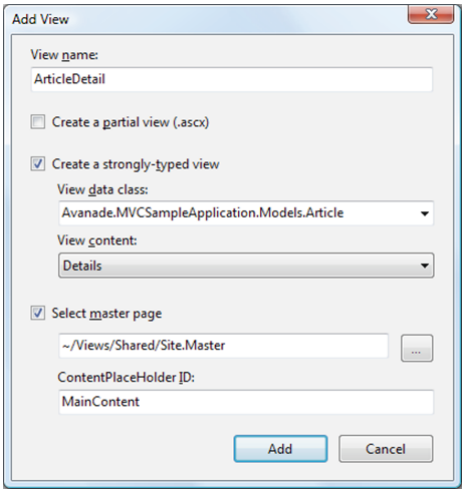
Result	Test Name
Passed	Does_ArticleRepository_Return_Expected_Article_Title_Test

FIGUUR 5: DOOR DE INVULLING VAN GETARTICLE SLAAGT DE TEST NU.

Als je http://localhost/Article/testtitel opent, krijg je in de browser de foutmelding zichtbaar zoals in figuur 6. Het MVC framework gaat op een aantal locaties zoeken naar 'ArticleDetail' view.



FIGUUR 6: FOUTMELDING BIJ NIET BESTAANDE VIEW.



FIGUUR 7: ADD VIEW WIZARD.

Aanmaken van views

Je maakt een view door in de code van de controller waar de view wordt aangeroepen rechts te klikken op de naam van de view en dan 'Add View' te selecteren. Een view kan aan een Model class worden gekoppeld waardoor je een stongly typed view krijgt. In figuur 7 zie je de wizard om een nieuwe view aan te maken.

De view is een aspx pagina zonder codebehind. In de aspx pagina kan via inline scripts de HTML worden gerenderd. Door in de wizard bij View Content in te stellen dat we een detail view willen aanmaken van de class Article worden automatisch alle velden van de Article class in de HTML gezet. In de code van de view kunnen via html helper functies verschillende standaard controls worden aangemaakt. Een voorbeeld hiervan is te zien in codevoorbeeld 6.

```
<asp:Content ID="Content2" ContentPlaceHolderID="MainContent"
runat="server">

    <h2>ArticleDetail</h2>

    <fieldset>
        <legend>Fields</legend>
        <p>
            ID:
            <%= Html.Encode(Model.ID) %>
        </p>
        <p>
            Title:
            <%= Html.Encode(Model.Title) %>
        </p>
        <p>
            Content:
            <%= Html.Encode(Model.Content) %>
        </p>
    </fieldset>

</asp:Content>
```

CODEVOORBEELD 6: CODE VAN DE ARTICLEDETAILS VIEW.

De toekomst van ASP.NET

Het ASP.NET MVC framework is niet gemaakt als vervanger van ASP.NET webforms. De twee manieren om webapplicaties te bouwen zullen naast elkaar blijven bestaan. Webforms hebben op dit moment een groot scala aan third-party controls waardoor webforms nog zeker blijven bestaan. Mocht je echter meer controle over je output HTML, pageflow of URLs willen dan is het MVC framework de oplossing.

Links

* Officiële ASP.NET MVC site
http://www.asp.net/mvc/
* Weblog van Scott Guthrie
http://weblogs.asp.net/scottgu/archive/2009/04/01/asp-net-mvc-1-0.aspx

Geert van der Cruijsen, is solution developer bij Avanade. Voor vragen of opmerkingen is hij te bereiken op g.van.der.cruijsen@avanade.com of via zijn weblog: www.vdcruijsen.net.