Fünftägiger ASP.NET MVC Workshop

Ihr Trainer: Johannes Hoppe von der conplement AG

Tag 2 - Agenda

1. Besprechung Ergebnisse Refactoring, Unit Testing
2. Grundlagen ASP.NET MVC (Bundling, Sections, Partial Views)
3. Action Filter (MVC)
4. OData

1. Ein einfacher Test mit MSpec

```
using FluentAssertions;
using Machine.Specifications;

[Subject("Test")]
public class When_doing_a_simple_test
{
    static int value1;
    static int value2;
    static int result;

    private Establish context = () =>
                                {
                                    value1 = 1;
                                    value2 = 2;
                                };

    Because of = () => result = value1 + value2;

    It should_have_the_expected_result = () => result.Should().Be(3);
}
```

2. Grundlagen

Sections

Die Methode RenderSection zum Rendern von benannten Bereichen, die ebenfalls in den einzelnen Views zu hinterlegen sind, verwendet werden.

```
<!-- im Index-View -->
@section scripts {
    <script>
        console.log("test");
    </script>
}

<!-- Layout.cshtml -->
@RenderSection("scripts", required: false)
```

Partielle Views

Partielle Views sind Views, welche in andere Views eingebunden werden.

```
<!-- Test.cshtml --->
@model string

<h1>@Model</h1>
<hr />
```

```
@Html.Partial("Test", "Dies ist Text")
```

3. Action Filter (MVC)

Action Filter bieten die Möglichkeit, vor und zwischen diesen Schritten benutzerdefinierte Logiken zur Ausführung zu bringen, wobei ein Filter auf beliebig viele Seiten angewandt werden kann.

IAuthorizationFilter / OnAuthorization: Wird ausgeführt, bevor die Anfrage abgearbeitet wird
IActionFilter / OnActionExecuting: Wird vor der Action-Methode ausgeführt
IActionFilter / OnActionExecuted: Wird nach der Action-Methode ausgeführt
IResultFilter / OnResultExecuting: Wird vor dem Action-Ergebnis (z. B. View) ausgeführt
IResultFilter / OnResultExecuted: Wird nach dem Action-Ergebnis (z. B. View) ausgeführt
IExceptionFilter / OnException: Wird ausgeführt, wenn eine Ausnahme ausgelöst wurde

Beispiel

```
using System.Diagnostics;
using System.Web.Mvc;
```

```
namespace AcTraining.Controllers
{
    /// <summary>
    /// Filter to display the execution time of both the action and result
    /// </summary>
    public class RequestTimingFilter : FilterAttribute, IActionFilter, IResultFilter
    {
        private static Stopwatch GetTimer(ControllerContext context, string name)
        {
            var key = string.Format("__timer__{0}", name);
            if (context.HttpContext.Items.Contains(key))
            {
                return (Stopwatch)context.HttpContext.Items[key];
            }

            var result = new Stopwatch();
            context.HttpContext.Items[key] = result;
            return result;
        }

        public void OnActionExecuting(ActionExecutingContext filterContext)
        {
            GetTimer(filterContext, "action").Start();
        }

        public void OnActionExecuted(ActionExecutedContext filterContext)
        {
            GetTimer(filterContext, "action").Stop();
        }

        public void OnResultExecuting(ResultExecutingContext filterContext)
        {
            GetTimer(filterContext, "render").Start();
        }

        public void OnResultExecuted(ResultExecutedContext filterContext)
        {
            var renderTimer = GetTimer(filterContext, "render");
            renderTimer.Stop();

            var actionTimer = GetTimer(filterContext, "action");
            var response = filterContext.HttpContext.Response;

            if (response.ContentType == "text/html")
            {
                response.Write(
                    string.Format(
                    "<b>Action '{0} :: {1}'<br /> Execute: {2}ms, Render: {3}ms.</b>",
                    filterContext.RouteData.Values["controller"],
                    filterContext.RouteData.Values["action"],
                    actionTimer.ElapsedMilliseconds,
                    renderTimer.ElapsedMilliseconds));
            }
        }
    }
}
```

4. OData

Es bietet sich für eine JavaScript-Anwendung eine Architektur nach dem **Re**presentational **S**tate **T**ransfer (REST [1]) an. Doch hinsichtlich der einzusetzenden Protokolle, Formate und Konventionen bleiben diverse Fragen für die praktische Umsetzung von REST offen. Wie sollen etwa die Query-Parameter heißen? Welchem Format soll eine Antwort genügen? Wie lassen sich die Schnittstellen maschinenlesbar definieren? Microsoft gibt hier mit dem Open Data Protocol (OData) eine ausführliche und standardisierte Antwort.

So wie der Web API Controller bislang implementiert wurde, wird ein Aufruf der Ressource ohne weitere Parameter eine Liste aller Entitäten zurückgeben. Es wird hierbei tatsächlich der gesamte Inhalt der Datenbank-Tabelle verwendet! Je mehr Daten vorhanden sind, desto unbrauchbarer wird dieser Ansatz. Es fehlt eine seitenweise Einschränkung der Ergebnismenge. An diesem Punkt stellt sich die Frage, wie die notwendigen Query-Parameter in der URL benannt werden sollten. Man könnte etwa "page" und "pageSize" verwenden. Man könnte sich auch von LINQ inspirieren lassen und auf "skip" und "take" setzen.

Die Entscheidungsmatrix lässt sich beliebig weiterführen und auf weitere Probleme ausweiten. Klärungsbedarf innerhalb eines Teams ist quasi vorprogrammiert. Eine zähe Entscheidungsfindung lässt sich gänzlich vermeiden, wenn man auf das OData Protokoll setzt. OData gibt die Namen der Parameter mit einer Sammlung von Konventionen exakt vor, so dass die Verwendung eindeutig wird. Die notwendigen Parameter heißen `$top` und `$skip`. `$top` gibt *n* Elemente der Ergebnismenge zurück. `$skip` überspringt *n* Elemente in der Ergebnismenge. Möchte man z.B. die Kunden mit der fortlaufenden Nummer 3 bis 7 abrufen, so verwendet man folgenden Aufruf:

```
GET http://example.org/odata/Customers?$top=5&$skip=2
```

Weitere Query-Parameter sind unter anderem `$filter`, `$orderby`, `$count` oder `$search`. Der bestehende Web API Controller kann durch ein paar Änderungen um die Funktionalität von OData ergänzt werden. Der Controller muss hierzu vom "ODataController" erben. Weiterhin ist es notwendig, dass die Funktionalität per `[EnableQuery]` explizit freigeschaltet wird.

```
public class CustomersController : ODataController
{
    private DataContext db = new DataContext();
```

```
    // GET: odata/Customers
    [EnableQuery]
    public IQueryable<Customer> GetCustomers()
    {
        return db.Customers;
    }

    /* [...] */
}
```

Anschließend ist es erforderlich die Klasse `WebApiConfig` zu konfigurieren. Mittels `config.Routes.MapODataRoute` legt man fest, unter welcher Adresse der "root" des OData Service zu finden ist. Alle Beispiele von Microsoft verwenden die Adresse "/odata", welche sich von der Adresse "/api" für normale ASP.NET Web API Aufrufe unterscheidet.

```
public static class WebApiConfig
{
    public static void Register(HttpConfiguration config)
    {
        ODataConventionModelBuilder builder = new ODataConventionModelBuilder();
        builder.EntitySet<Customer>("Customers");
        builder.EntitySet<Invoice>("Invoices");
        config.Routes.MapODataServiceRoute("odata", "odata", builder.GetEdmModel());
    }
}
```

Der Controller unterstützt nun eine seitenweise Ausgabe, Sortierung und Filterung. Diese Fähigkeiten direkt mit AngularJS umzusetzen wäre ein großer Aufwand. Es bietet sich an, ein fertiges Tabellen-Control ("Grid") zu verwenden. Auf dem Markt finden sich eine Reihe von freien und proprietären Grids, welche mit AngularJS kompatibel sind. Ein bekanntes und weit verbreitetes Framework ist Kendo UI von Telerik:

http://demos.telerik.com/kendo-ui/grid/index

Hinweis, bei einem OData v3 Endpunkt muss die Datasource wie folgt angepasst werden:

```
var dataSource = new kendo.data.DataSource({
    type: 'odata',
    transport: {
        read: {
            type: 'GET',
            url: '/odata/Customers',
            dataType: 'json'
        }
    },
    schema: {
        data: function (data) { return data.value; },
        total: function (data) { return data['odata.count']; },
        model: {
            id: 'Id',
            fields: {
                Id: { type: 'number' },
                FirstName: { type: 'string' },
                LastName: { type: 'string' },
                Mail: { type: 'string' },
                DateOfBirth: { type: 'date' }
            }
        }
    },
    serverPaging: true,
    serverSorting: true,
    serverFiltering: true,
    pageSize: 10
});
```

Aufgaben:

1. Fügen Sie Kendo UI per Nuget zum Projekt hinzu
2. Implementieren Sie ein Grid mit Kendo UI
3. Implementieren Sie ein Chart mit Kendo UI
