

M183 Applikationssicherheit Implementieren

Tutorial zum Lab Databases, XSS & SQL Injections

Version 1	23.11.2017	Jürg Nietlispach
-----------	------------	------------------

Contents

Idee	3
Setup ASP.NET MVC Applikation	3
Setup Datenbank.....	3
Herangehensweise Login- und Feedback-Formular.....	3
Setup eines ASP.NET MVC Projekts	4
Setup Datenbank.....	6

Idee

In diesem Lab soll eine .NET MVC Applikation erstellt werden, welche über eine Login-Form und ein Feedback-Formular verfügt.

Die Login-Daten sollen in der Datenbank gegengeprüft werden – SQL-Injections sind hier möglich und sollen in einer zweiten Phase verhindert werden. Die Inhalte des Feedback-Formulars werden ebenfalls in der Datenbank gespeichert – XSS-Attacken wären hier möglich und sollen ebenfalls in einer zweiten Phase verunmöglicht werden.

XSS- und SQL-Injection Attacken können auf mehreren Ebenen verhindert werden. Es sollen in diesem Lab entsprechende Gegenmassnahmen implementiert werden.

Zuerst sollen für die Erkennung von XSS und SQL-Injections Regular Expressions gefunden werden, um die unerlaubten Zeichen später dann herauszufiltern.

In einem zweiten Schritt (falls Zeit) sollen dann die Standardmethoden vom .NET Framework selber verwendet werden, um die Attacken zu verhindern (z.B. die Verwendung des Entity Frameworks)

Setup ASP.NET MVC Applikation

1. Erstellen einer ASP.NET MVC Applikation gemäss Anleitung.

Setup Datenbank

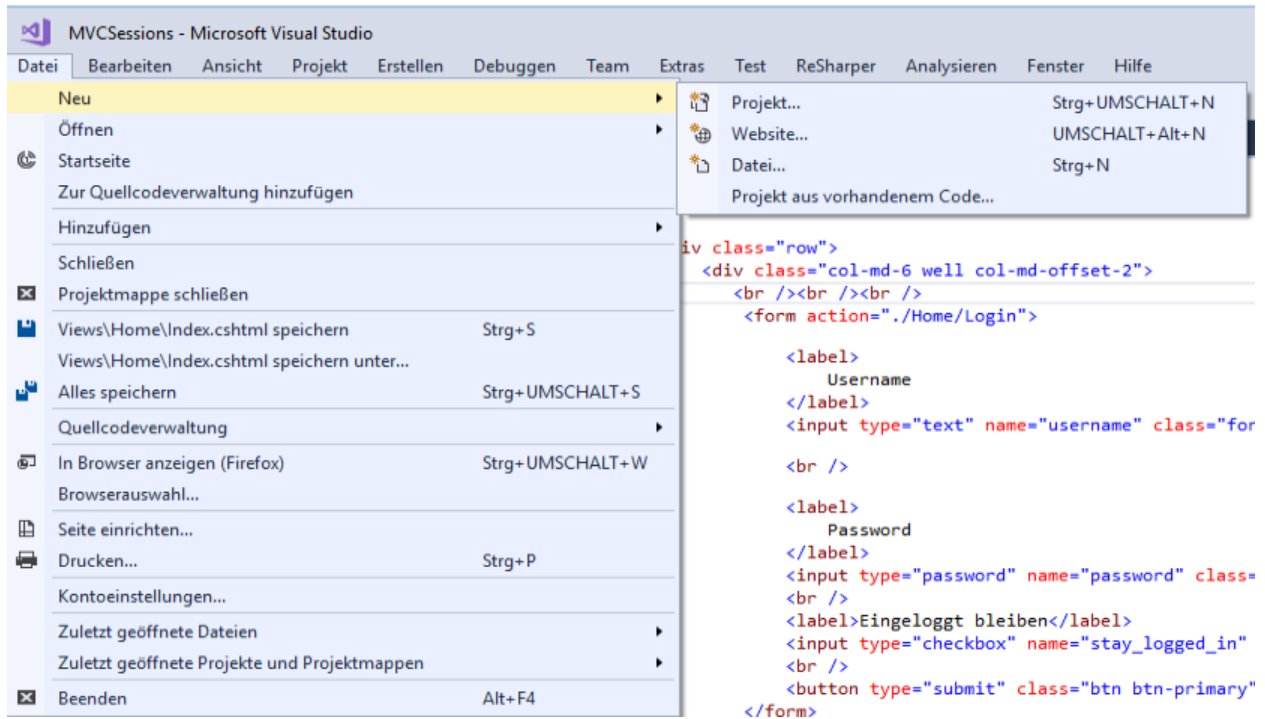
1. Erstellen einer Datenbank, welche im Applikationskontext verwendet werden kann.

Herangehensweise Login- and Feedback-Formular

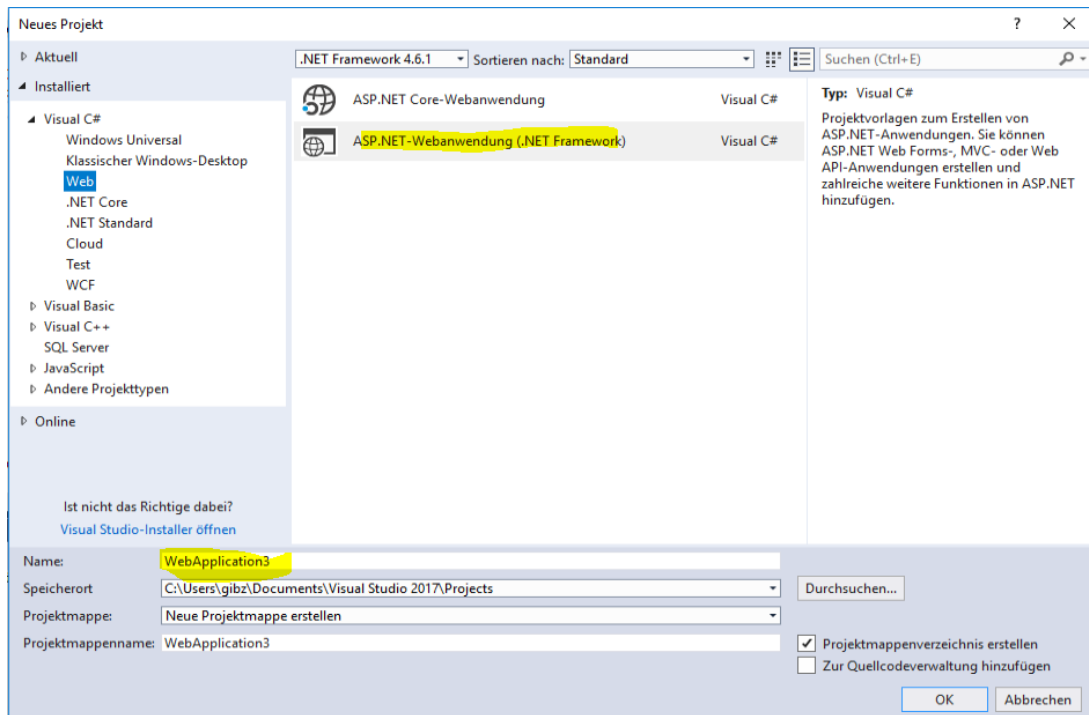
1. Loginmaske mit Username, Passwort erstellen
2. Login-Prozedere realisieren (Benutzernamen und Passwort aus der Datenbank gegenprüfen)
3. Feedbackformular erstellen, welche beim Posten an den Server in die DB gespeichert wird
4. System erweitern, damit XSS- und SQL-Injections nicht mehr möglich sind

Setup eines ASP.NET MVC Projekts

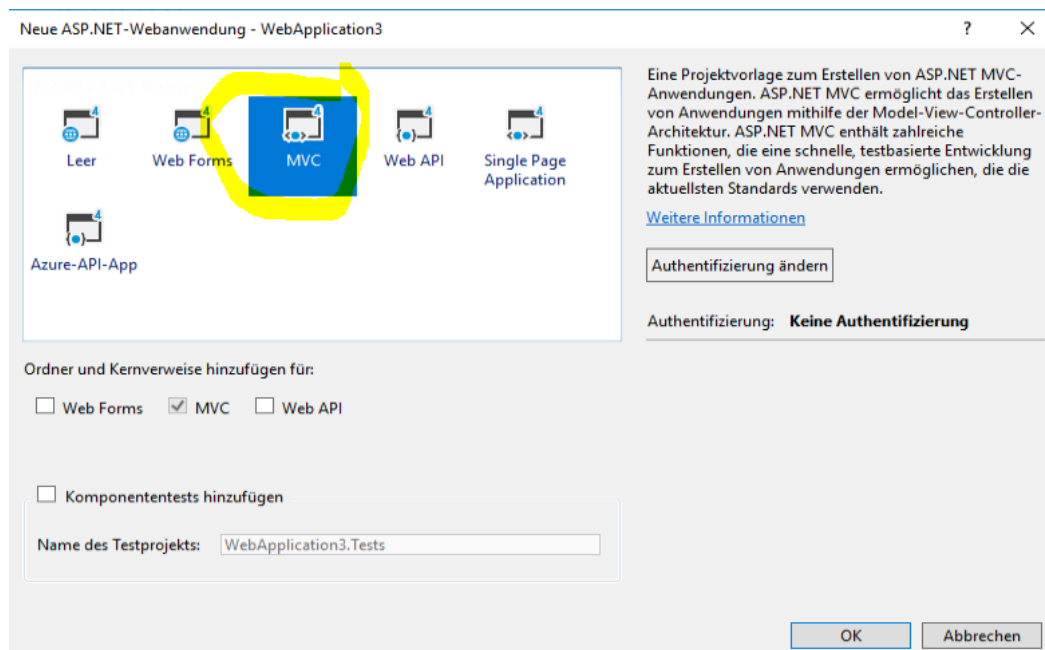
In Visual Studio ein neues Projekt erstellen:



Entsprechendes Projekt auswählen: C# Web-Projekt, ASP.NET Projekt. Namen für Projekt angeben:

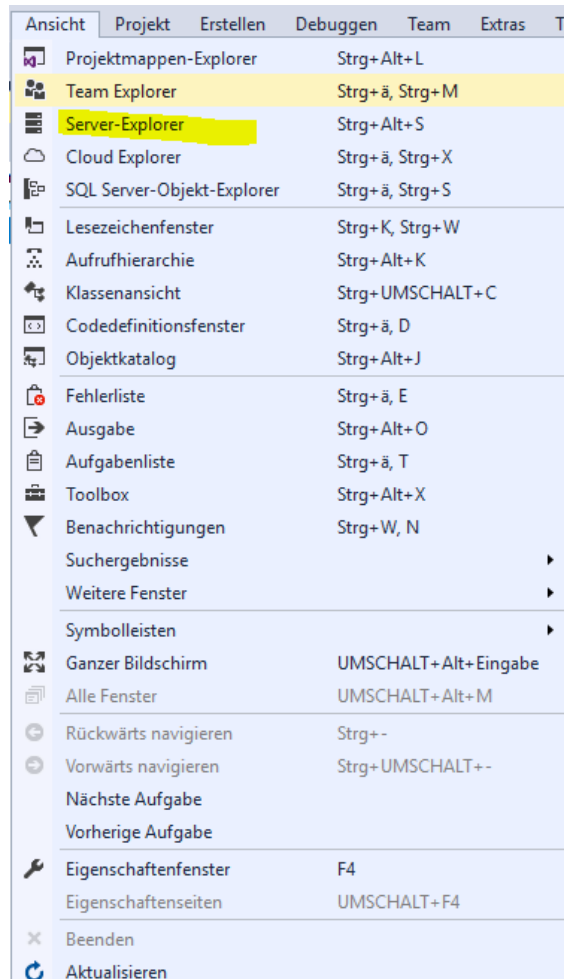


MVC-Projekt auswählen (erstellt ein MVC-Boilerplate-Projekt inkl. Bootstrap Layout Engine)

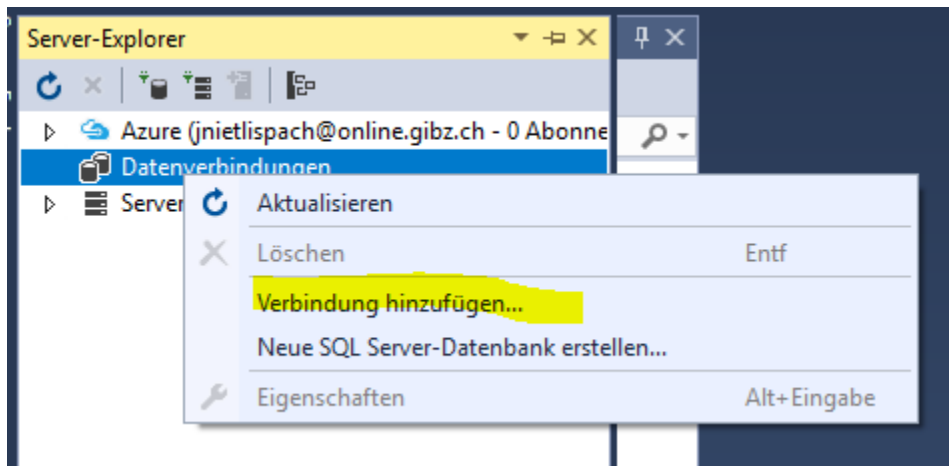


Setup Datenbank

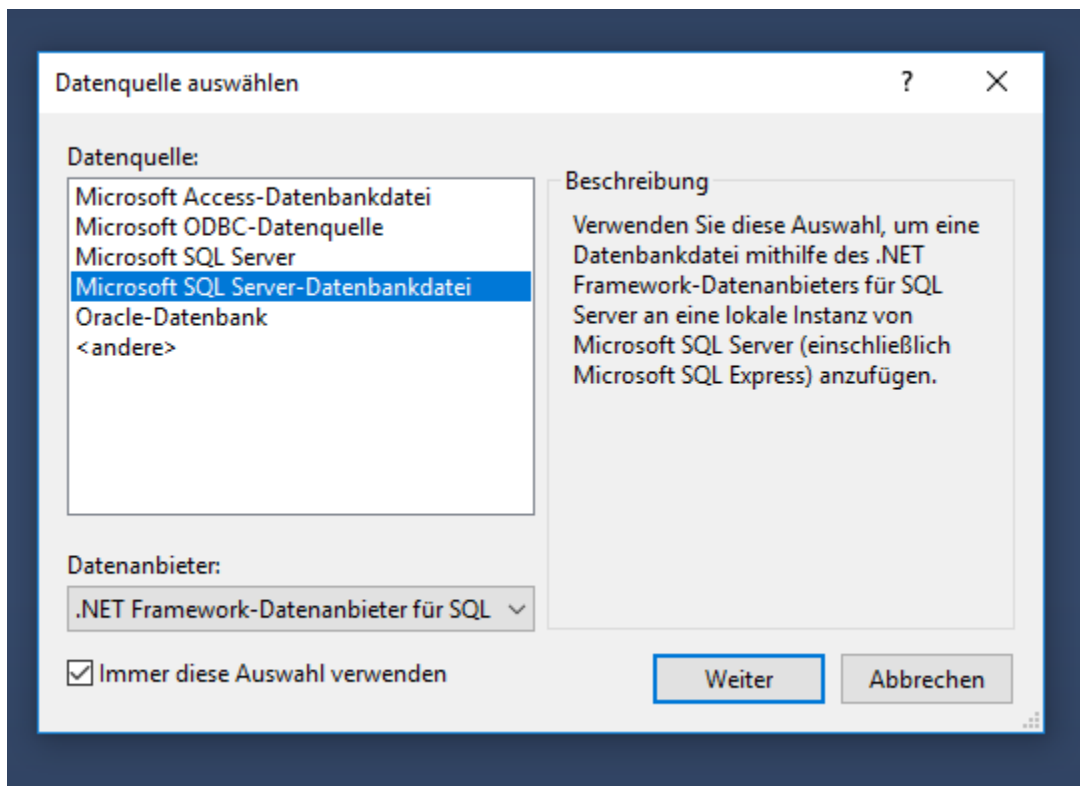
Damit die Daten korrekt in einer Datenbank abgefüllt werden können, müssen wir noch ein paar Vorkehrungen treffen. Im Server Explorer ...



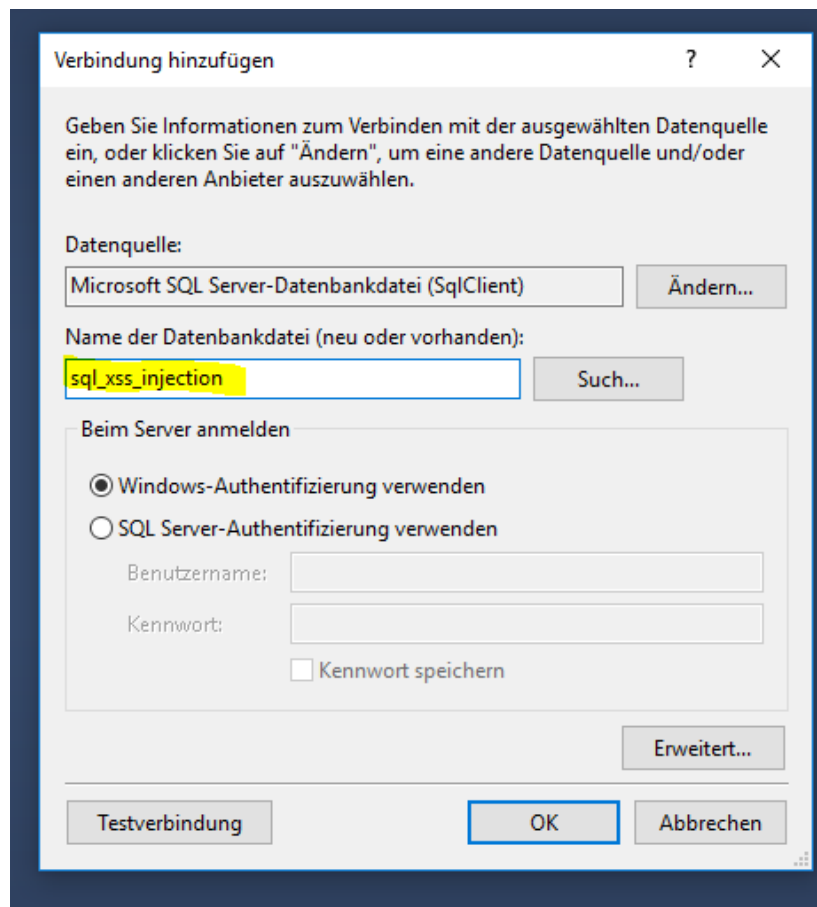
... werden dann die Datenverbindungen angezeigt.



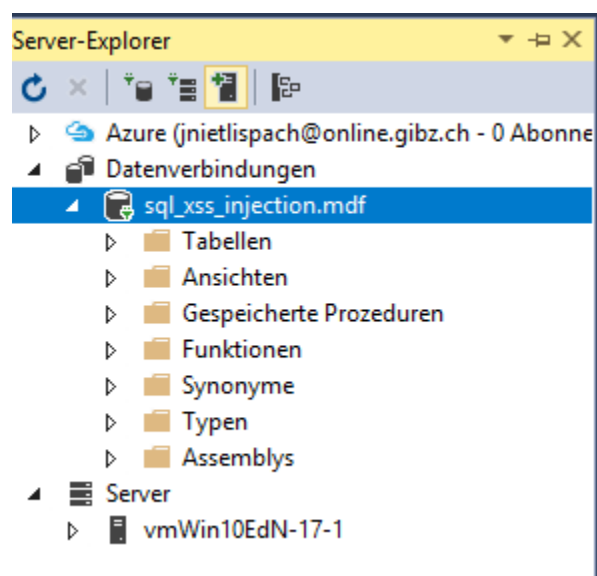
Man kann nun via Visual Studio, ein Datenbankfile erstellen.



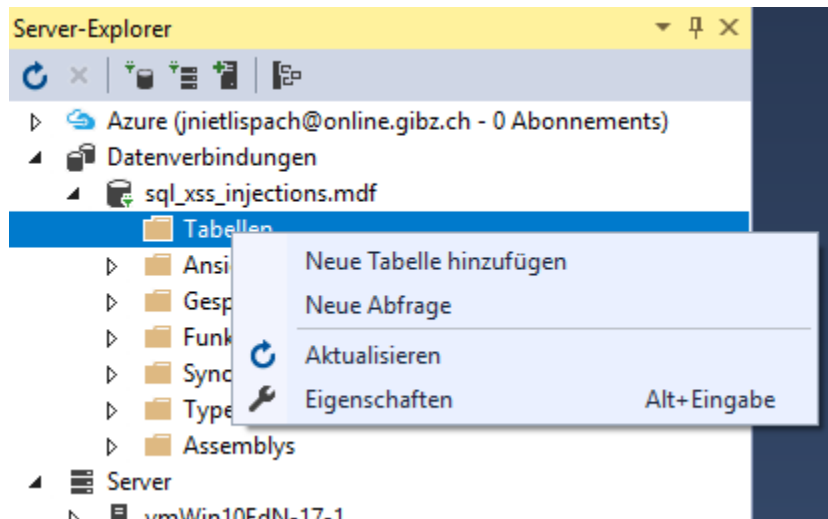
Hierfür muss für die Datenbank noch einen Namen angegeben werden.



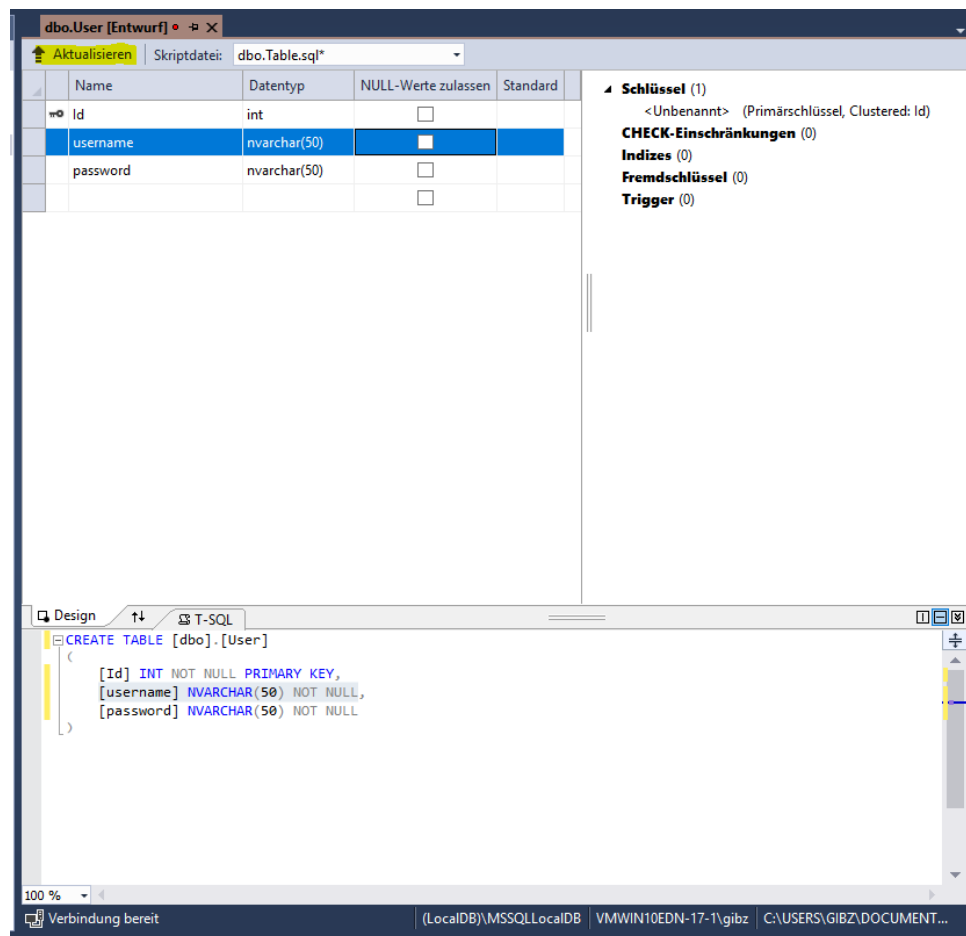
Schlussendlich wird die DB dann erstellt.



Nun können Tabellen für User und Feedback erstellt werden:



Die Felder können so gewählt werden.

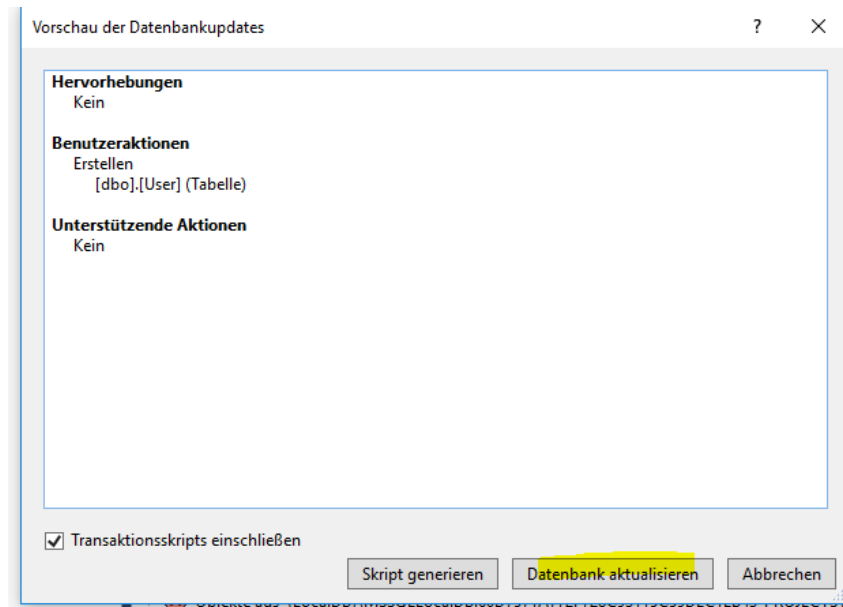


. Wichtig ist, dass beide Tabellen bei der ID über ein Autoincrement verfügen:

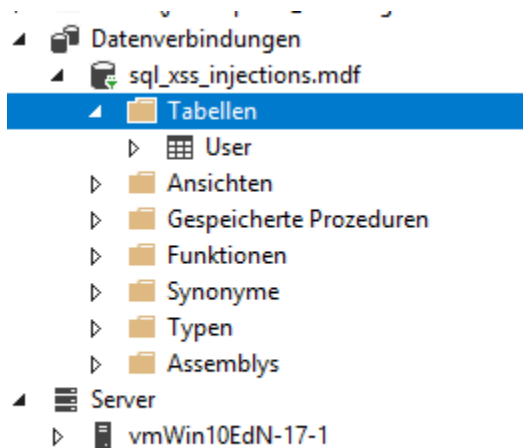
Dies kann mit dem Identity-Parameter angegeben werden.

```
[Id] INT IDENTITY(1,1) NOT NULL,
```

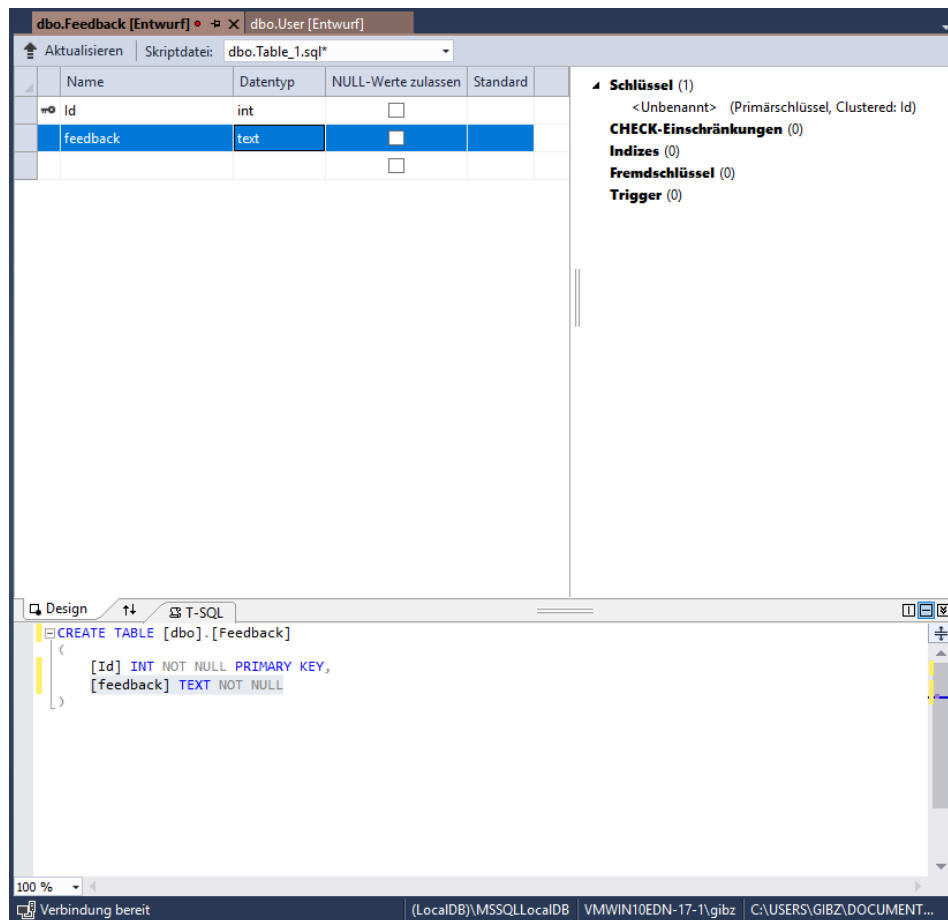
Mit „Aktualisieren“ und Bestätigung werden die Anpassungen übernommen



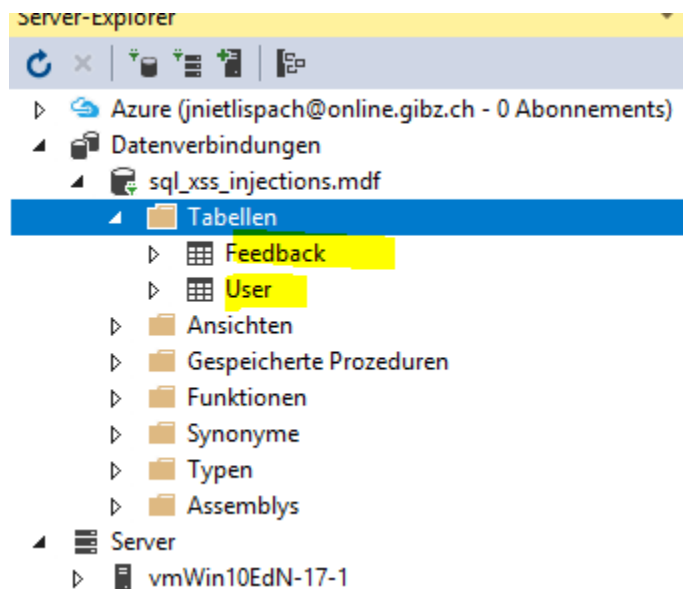
Aktualisiert man die Anzeige, kommt die User-Tabelle zum Vorschein:

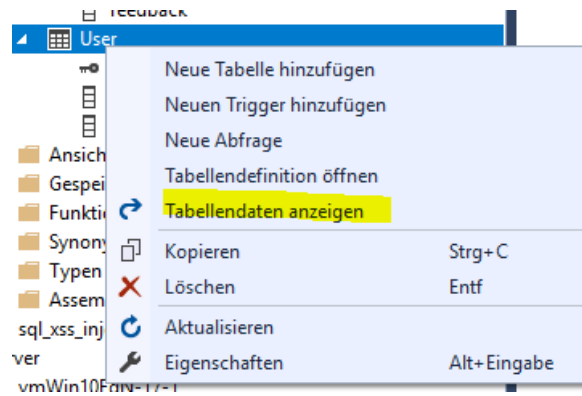


Analog dazu kann die Tabelle Feedback erstellt werden:



Im Table-Browser sind nun beide Tabellen vorhanden:





Es können nun so Daten direkt eingegeben werden:

HomeController.cs dbo.Feedback [Entwurf] dbo.Feedback [Daten] + X		
Max. Zeilen: 1000		
Id	feedback	
1	Hallo Zusammen	
2	<script></scrip...	
NULL	NULL	

Bei der User Tabelle analog:

HomeController.cs dbo.Feedback [Entwurf] dbo.Feedback [Daten] dbo.User [Daten] + X			
Max. Zeilen: Alle			
Id	username	password	
1	admin	test	
2	user	test	
NULL	NULL	NULL	

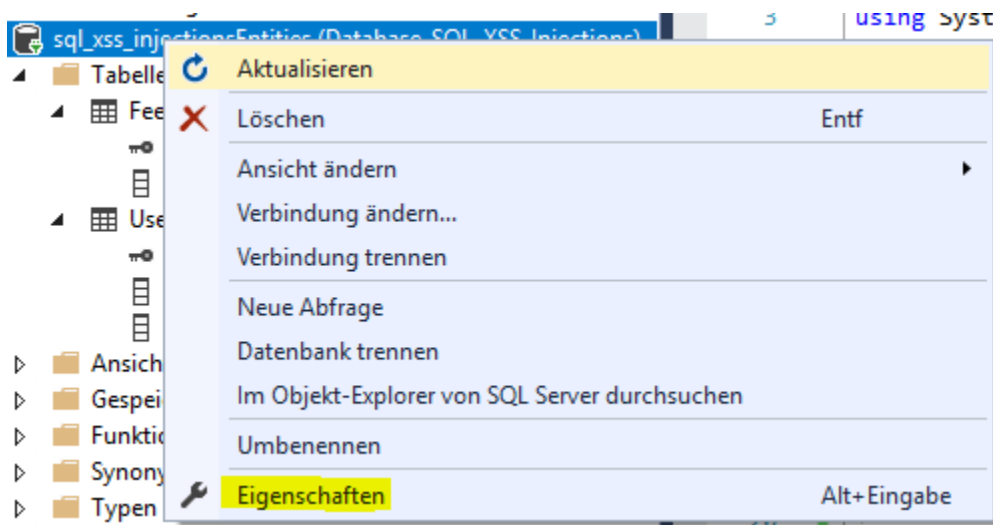
Nun kann aus dem Code die Datenbank angesprochen werden.

```
SqlConnection con = new SqlConnection();
con.ConnectionString = "Data Source=(LocalDB)\\MSSQLLocalDB;AttachDbFilename=\\\"C:\\\\Use

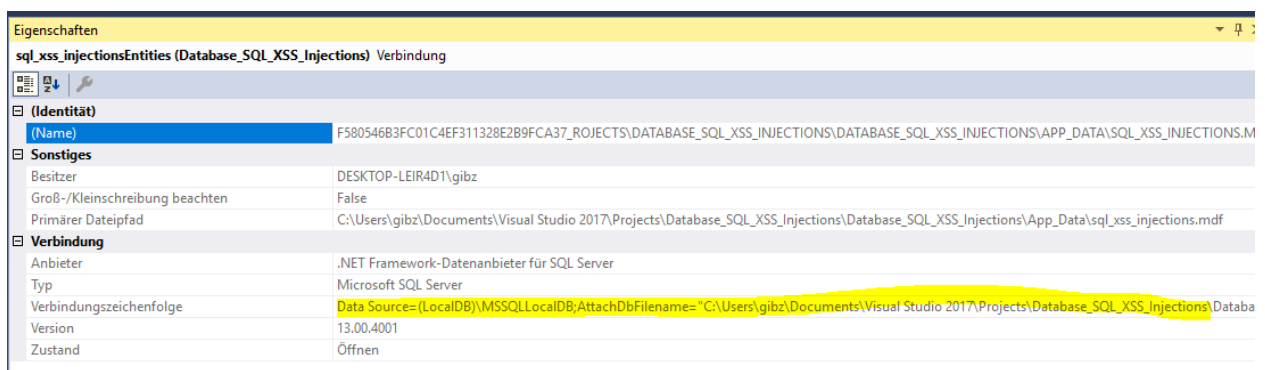
SqlCommand cmd = new SqlCommand();
SqlDataReader reader;

cmd.CommandText = "SELECT [Id] ,[username] ,[password] FROM [dbo].[User]";
//cmd.CommandType = CommandType.Text;
cmd.Connection = con;
```

Den Connection – String kann man aus dem Server-Manager in den Datenbank-Eigenschaften herauslesen:



Connection – String = Verbindungszeichenfolge



d

Wurde der String hinterlegt, können Queries gemacht werden.

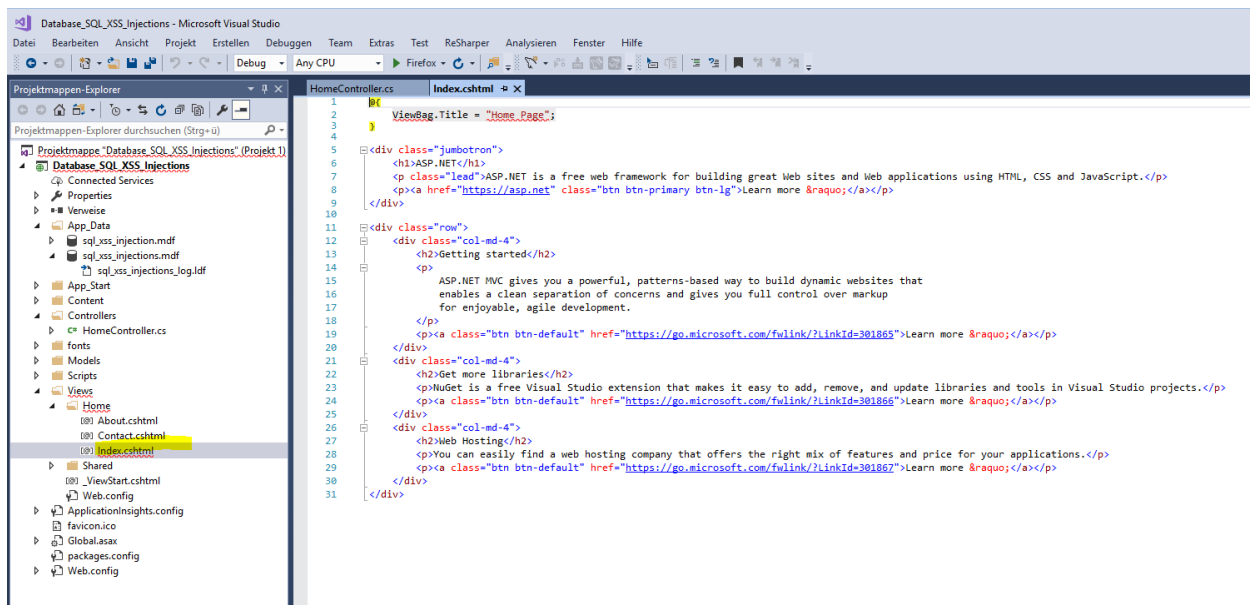
```
cmd.CommandText = "SELECT [Id] ,[username] ,[password] FROM [dbo].[User]";
//cmd.CommandType = CommandType.Text;
```

Diese werden mit dem ExecuteReader (retourniert eine DataReader Klasse) ausgelesen.

```
reader = cmd.ExecuteReader();
// Data is accessible through the DataReader object here.
if (reader.HasRows)
{
    while (reader.Read())
    {
        Console.WriteLine("{0}\t{1}", reader.GetInt32(0), reader.GetString(1));
    }
}
else
{
    Console.WriteLine("No rows found.");
}
```

Es muss nun ein Formular für die Login-Form mit entsprechendem Login-Controller erstellt werden und analog dazu das Feedback-Formular (Template und Controller). Dies wurde in den Übungen nun schon ein paarmal gemacht und sollte nun selbständig funktionieren können. Hier trotzdem noch ein paar Hinweise:

Das File Index.cshtml



Entsprechend so anpassen:

```

<div class="row">
  <form action="./Home/DoLogin" method="post" class="well col-lg-8 col-lg-offset-2" style="margin-top:60px;">

    <label>Username</label>
    <input type="text" name="username" value="" class="form-control"/>

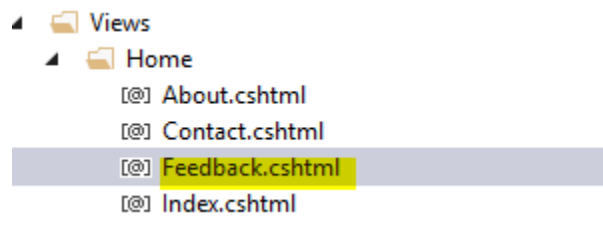
    <label>Password</label>
    <input type="password" name="password" value="" class="form-control" />

    <input type="submit" name="" value="Log In" class="btn btn-primary"/>

  </form>
</div>

```

Nun wird das Feedback Template erstellt:



Folgendes HTML wird benötigt

```

<div class="row">
  <form action="./Home/DoFeedback" method="post" class="well col-lg-8 col-lg-offset-2" style="margin-top:60px;">

    <label>Feedback</label>
    <textarea name="feedback" class="form-control"></textarea>

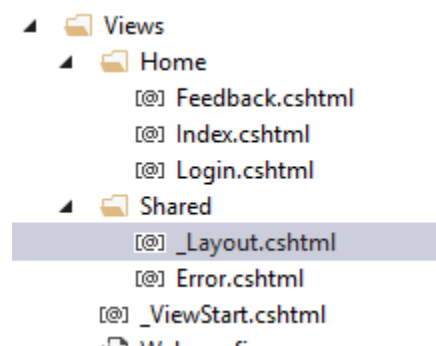
    <label>Password</label>

    <input type="submit" name="" value="Log In" class="btn btn-primary"/>

  </form>
</div>

```

Wenn man im _Layout.cshtml ...



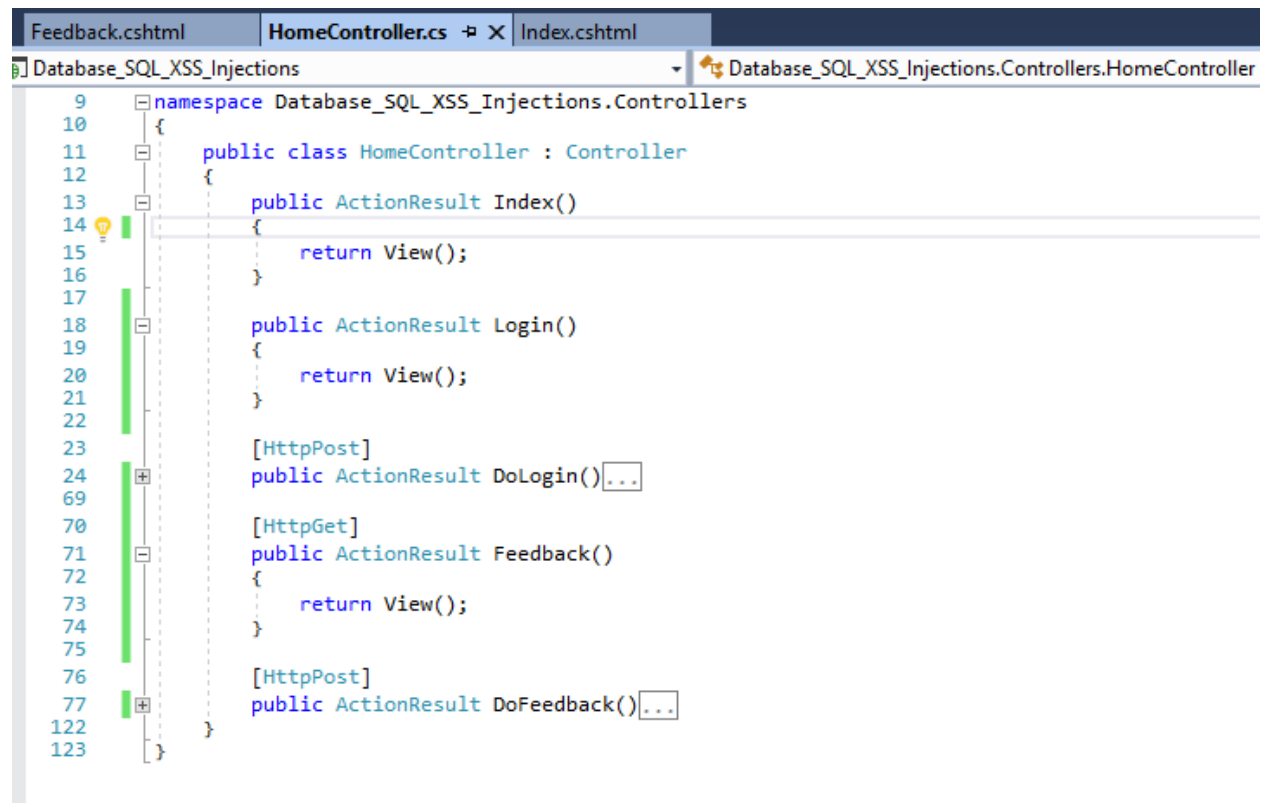
... noch die entsprechenden Links umbenennt:

```

:body>
  <div class="navbar navbar-inverse navbar-fixed-top">
    <div class="container">
      <div class="navbar-header">
        <button type="button" class="navbar-toggle" data-toggle="collapse" data-target=".navbar-collapse">
          <span class="icon-bar"></span>
          <span class="icon-bar"></span>
          <span class="icon-bar"></span>
        </button>
        @Html.ActionLink("Anwendungsname", "Index", "Home", new { area = "" }, new { @class = "navbar-brand" })
      </div>
      <div class="navbar-collapse collapse">
        <ul class="nav navbar-nav">
          <li>@Html.ActionLink("Login", "Index", "Home")</li>
          <li>@Html.ActionLink("Feedback", "Feedback", "Home")</li>
        </ul>
      </div>
    </div>
  </div>
  <div class="container body-content">
    @RenderBody()
    <hr />
    <footer>
      <p>&copy; @DateTime.Now.Year - Meine ASP.NET-Anwendung</p>
    </footer>
  </div>

```

Nun müssen noch die entsprechenden GET und POST-Routen im Home-Controller definiert werden:



```

9 namespace Database_SQL_XSS_Injections.Controllers
10 {
11     public class HomeController : Controller
12     {
13         public ActionResult Index()
14         {
15             return View();
16         }
17
18         public ActionResult Login()
19         {
20             return View();
21         }
22
23         [HttpPost]
24         public ActionResult DoLogin()...
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70         [HttpGet]
71         public ActionResult Feedback()
72         {
73             return View();
74         }
75
76         [HttpPost]
77         public ActionResult DoFeedback()...
78
79     }
80 }

```

Nun sieht die Applikation folgendermassen aus:

Username

Password

Log In

Bzw.

localhost:49818/Home/Feedback

Anwendungsname Login Feedback

Feedback

Submit

© 2017 - Meine ASP.NET-Anwendung

Beim Absenden der Formulare muss nun Controller Logik bei den entsprechenden Routen hinterlegt werden. Einerseits das Auslesen der Parameter aus dem Request und andererseits die Validierung/Filterung und die Speicherung der Daten in der DB.

Beim Login-Controller (POST) könnte das z.B. so aussehen:

```

[HttpPost]
public ActionResult DoLogin()
{
    var username = Request["username"];
    var password = Request["password"];

    SqlConnection con = new SqlConnection();
    con.ConnectionString = "Data Source=(LocalDB)\\MSSQLLocalDB;AttachDbFilename=\\\"C:\\Users\\gibz\\Documents\\Visual Studio 2017\\Projects\\

    SqlCommand cmd = new SqlCommand();
    SqlDataReader reader;

    cmd.CommandText = "SELECT [Id], [username], [password] FROM [dbo].[User] WHERE [username] = '"+username+"' AND [password] = '" + password
    cmd.Connection = con;

    con.Open();

    reader = cmd.ExecuteReader();

    if (reader.HasRows)
    {
        ViewBag.Message = "success";

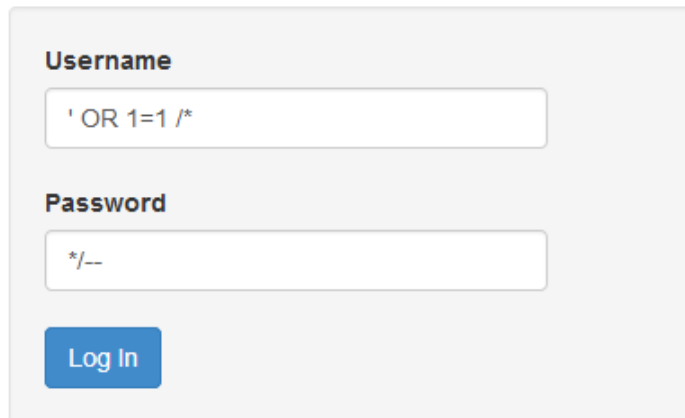
        while (reader.Read())
        {
            ViewBag.Message += reader.GetInt32(0) + " " + reader.GetString(1) + " " + reader.GetString(2); // id & value?
        }
    }
    else
    {
        ViewBag.Message = "nothing to read of";
    }
}

```

Wurde Benutzernamen und Passwort korrekt eingegeben, wird das in der View momentan so ausgegeben:

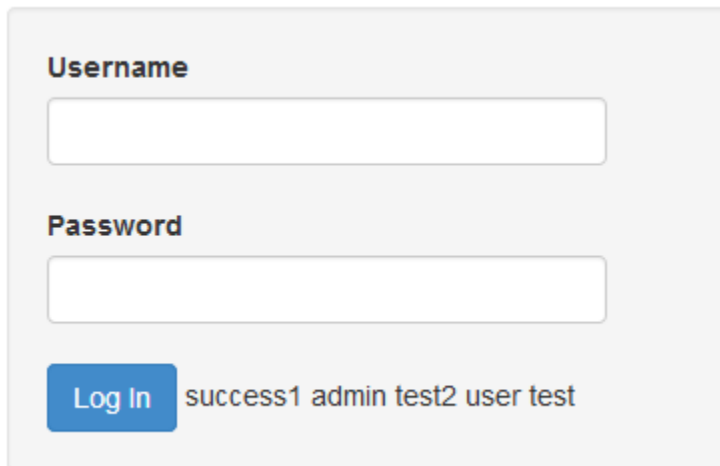
The screenshot shows a login form with two input fields: 'Username' containing 'admin' and 'Password' containing four dots. Below the fields is a blue 'Log In' button. To the right of the button, the text 'success1 admin test' is displayed, which is the output of the C# code shown in the previous block.

Nun ist es natürlich möglich, folgende Eingabe zu machen (Das Passwort-Feld wurde im Browser auf Text-Feld umgestellt).



A login form with two input fields and a button. The first field is labeled "Username" and contains the text "' OR 1=1 /*". The second field is labeled "Password" and contains the text "*/--". Below the fields is a blue button labeled "Log In".

Der Query liefert dann alle Datensätze (inkl. allen Feldern) zurück:



A login form with two empty input fields and a blue button labeled "Log In". Below the button, the text "success1 admin test2 user test" is displayed.

In einer zweiten Phase (innerhalb der POST Routinen) müssen Regular Expressions fürs Input Filtering erstellt werden oder Escape-Funktionen für Single Quotes

Beim Feedback Fomular dann analog.

Die Controller-Funktion sieht im Wesentlichen dann gleich aus.

```

[HttpPost]
public ActionResult DoFeedback()
{
    var feedback = Request["feedback"];

    SqlConnection con = new SqlConnection();
    con.ConnectionString = "Data Source=(LocalDB)\\MSSQLLocalDB;AttachDbFilename=\\\"C:\\Users\\gibz\\Documents\\Visual Studio 2017\\Projects\\Dat

    SqlCommand cmd = new SqlCommand();
    SqlDataReader reader;

    cmd.CommandText = "INSERT INTO [dbo].[Feedback] SET [feedback] = '" + feedback + "'";
    cmd.Connection = con;

    con.Open();

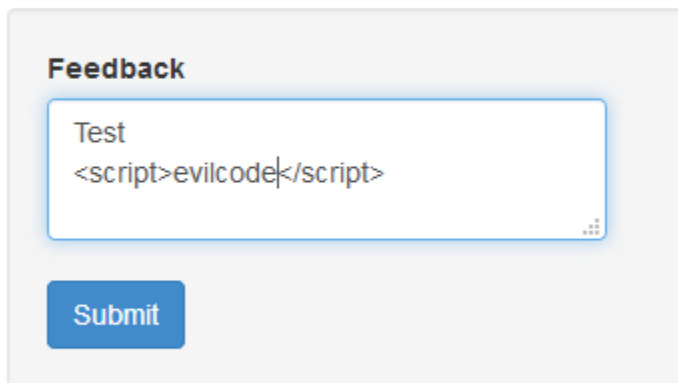
    reader = cmd.ExecuteReader();

    if (reader.HasRows)
    {
        ViewBag.Message = "success";

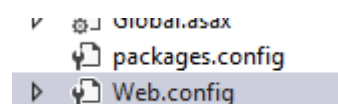
        while (reader.Read())
        {
            ViewBag.Message += reader.GetInt32(0) + " " + reader.GetString(1) + " " + reader.GetString(2); // id & value?
        }
    }
    else
    {
        ViewBag.Message = "nothing to read of";
    }
}

```

Die Attacke kann theoretisch so gestartet werden.



Ist in der web.config-Datei



aber bereits vordefiniert, dass gewisse Request-Elemente geprüft werden sollen (also die gelb markierten Elemente auf true gesetzt sind)...

```

</configSections>
<appSettings>
  <add key="webpages:Version" value="3.0.0.0" />
  <add key="webpages:Enabled" value="false" />
  <add key="ClientValidationEnabled" value="false" />
  <add key="UnobtrusiveJavaScriptEnabled" value="false" />
</appSettings>

```

Und

```

<system.web>
  <compilation debug="true" targetFramework="4.6.1" />
  <httpRuntime targetFramework="4.6.1" />
  <httpModules>
    <add name="ApplicationInsightsWebTrack" />
  </httpModules>
  <pages validateRequest="false" />
</system.web>

```

... wirft das .NET Framework sicherheitshalber bereits einen Fehler:

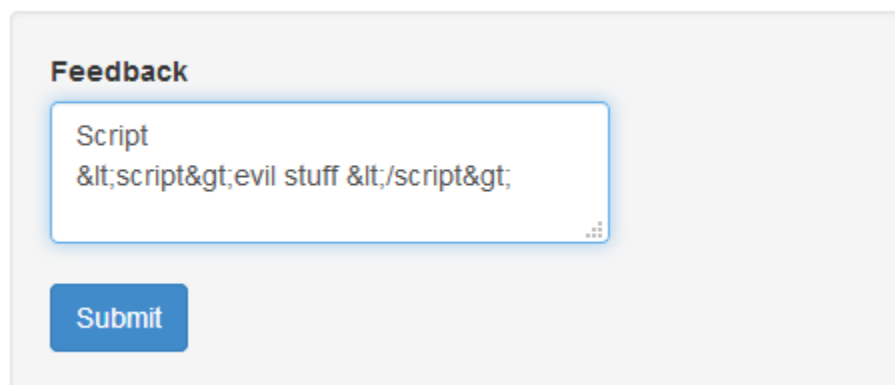
Serverfehler in der Anwendung /.

Ein möglicherweise gefährlicher Request.Form-Wert wurde vom Client (feedback="Test <script>evilcode</script>") entdeckt.

Beschreibung: ASP.NET hat potenziell schädliche Daten in der Anforderung festgestellt, die möglicherweise HTML-Code oder Skripts enthalten. Die Daten stellen möglicherweise Code in einer Webpage hinzufügen, mit dem dies explizit zugelassen wird. Weitere Informationen finden Sie unter <http://go.microsoft.com/fwlink/?LinkId=212874>.

Ausnahmedetails: System.Web.HttpRequestValidationException: Ein möglicherweise gefährlicher Request.Form-Wert wurde vom Client (feedback="Test <script>evilcode</script>") entdeckt.

Sind diese Werte aber auf false eingestellt, ist eine XSS-Attacke möglich. Sollte dies immer noch nicht der Fall sein, kann versucht werden, mit htmlentities encode die Sicherheitsmassnahmen zu umgehen:



Auf diese Weise kann bei Anzeigen der Feedbacks das Script auch aktiv werden.

Durch das .NET Framework wird bereits einiges gemacht, um Cross-Site-Scripting zu verhindern. Sollte dies nicht reichen, ist es möglich mit Regexes den Input zusätzlich zu Filtern.