

M183 Applikationssicherheit

Implementieren

Tutorial zur Übung 2-Factor-Authentication with TOTP Token
(Google Authenticator)

Version 1	26.10.2017	Jürg Nietlispach
-----------	------------	------------------

Contents

Idee	3
Setup Google Authenticator	3
Setup ASP.NET MVC Applikation	3
Herangehensweise Login-Prozedere	3
Setup Google Authenticator	4
Setup eines neuen ASP.NET MVC Projekt.....	5
Login-Prozedere	7

Idee

Two Factor Authentication kommt in verschiedenen Varianten vor. In der vorliegenden Übung soll ein Login-Prozedere (ASP.NET-MVC-Applikation) durch die Abfrage eines Timebased One Time Passwords (TOTP) erweitert werden. Das TOTP wird – nach Austauschen eines Secret-Keys (z.B. mittels QR-Code) zwischen Backendsystem und dem Client - z.B. von Google Authenticator generiert (jeweils frisch in regelmässigen zeitlichen Abständen).

Setup Google Authenticator

1. Im App- oder Android-Store nach Google Authenticator suchen und installieren

Generierung und Austausch eines Secret Keys

1. Generierung Secret Key und Austausch via QR-Code

Setup ASP.NET MVC Applikation

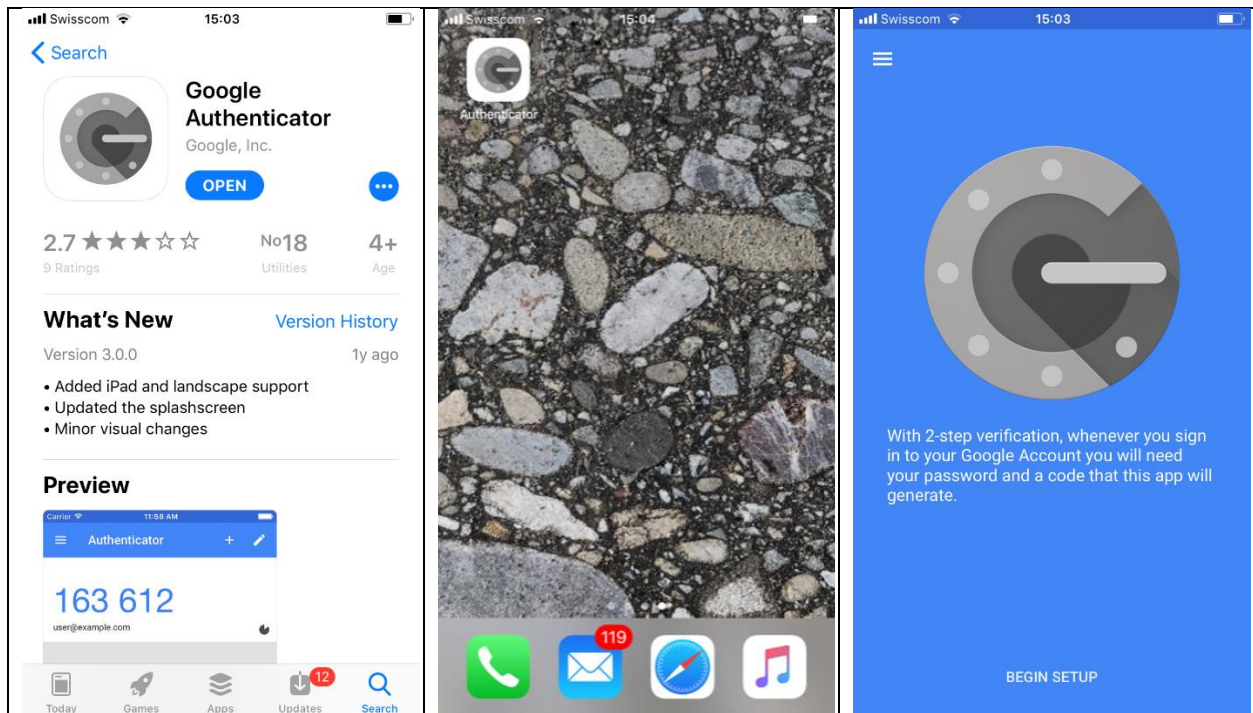
1. Erstellen eines neuen ASP.NET Projektes

Herangehensweise Login-Prozedere

1. Loginmaske mit Benutzernamen und Passwort erstellen
2. Business-Logik erstellen, welches einfach Benutzernamen und Passwort prüft
3. Sind die Eingaben korrekt, soll ein drittes bzw. ein neues Eingabefeld für den Token angezeigt werden.
4. Bei Eingabe des Tokens soll die Business-Logik der Applikation das Token prüfen und eine entsprechende Antwort senden.

Setup Google Authenticator

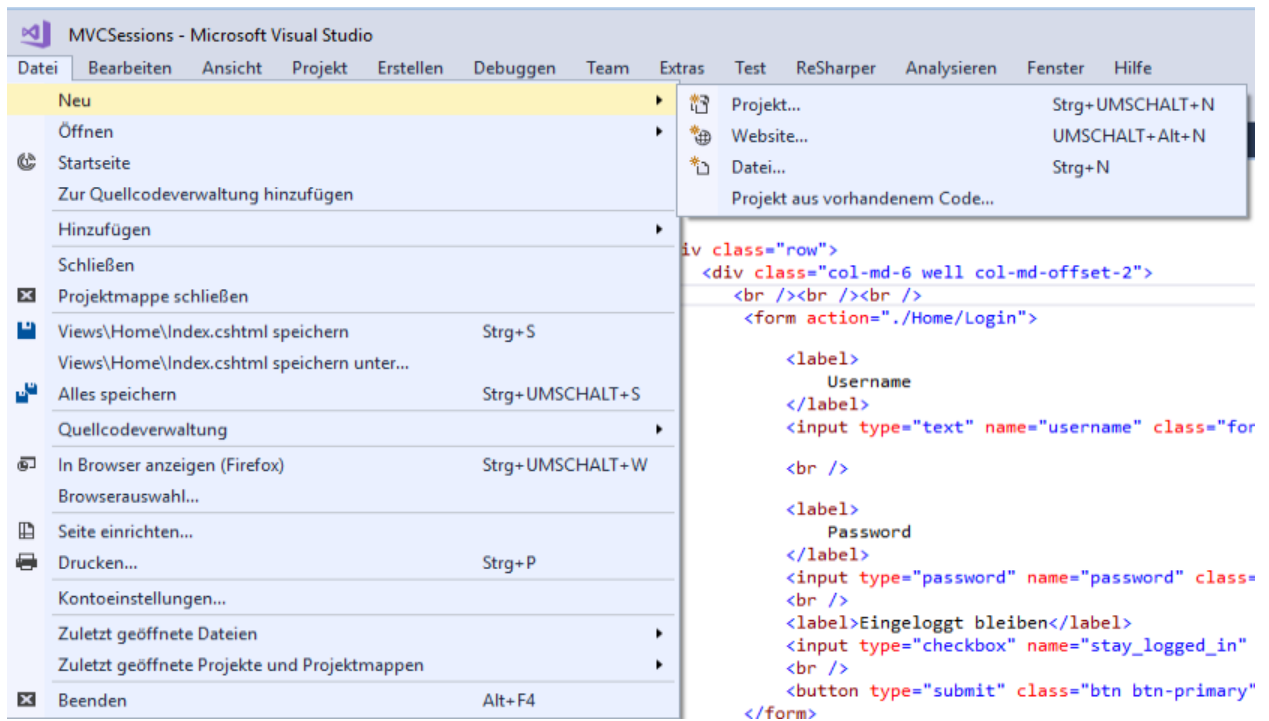
Z.B. im App-Store nach Google Authenticator suchen und installieren:



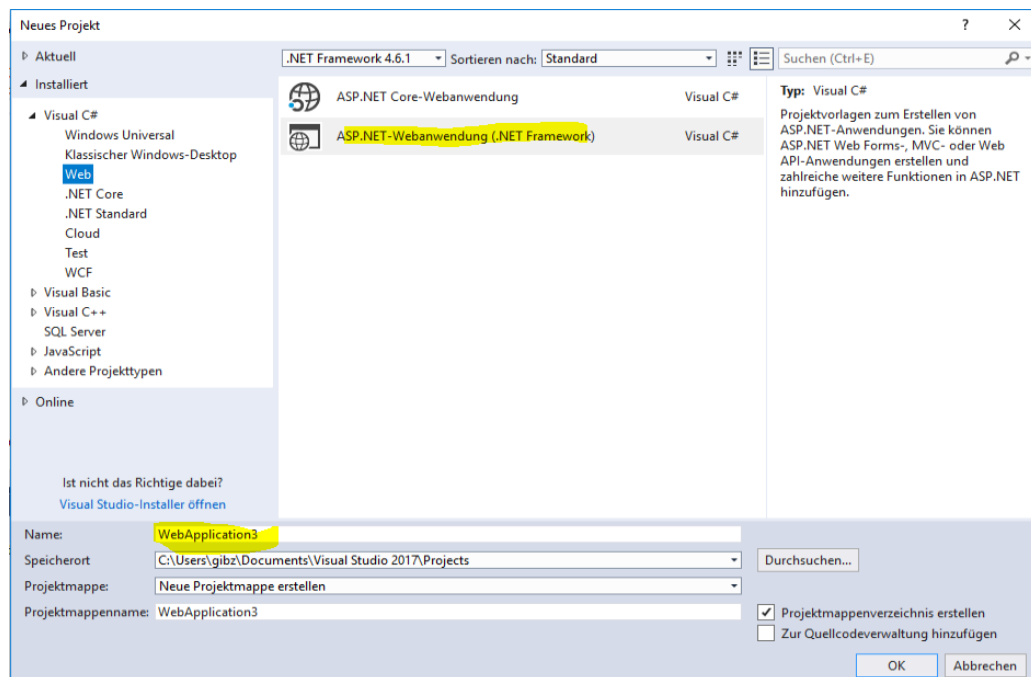
Der Initialisierungs-Schritt des Apps (Scannen des QR-Codes) erfolgt weiter unten im Tutorial

Setup eines neuen ASP.NET MVC Projekt

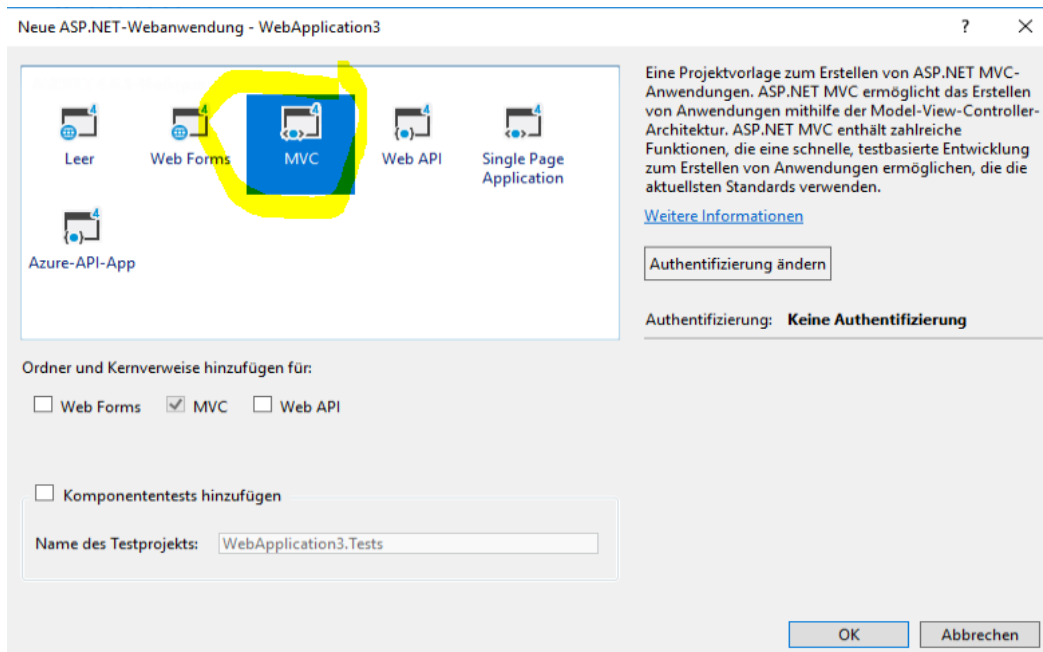
1. In Visual Studio ein neues Projekt erstellen:



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



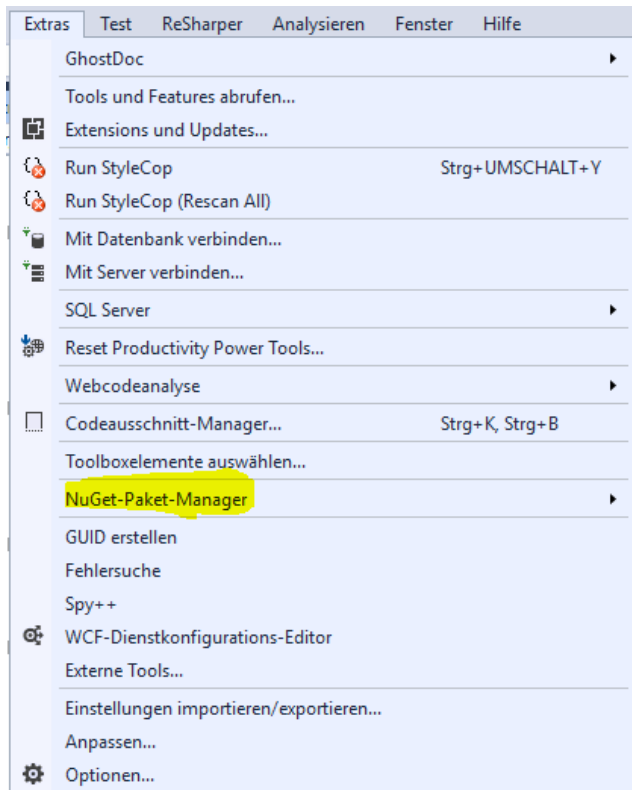
3. MVC-Projekt auswählen (erstellt ein MVC-Boilerplate-Projekt inkl. Bootstrap Layout Engine)



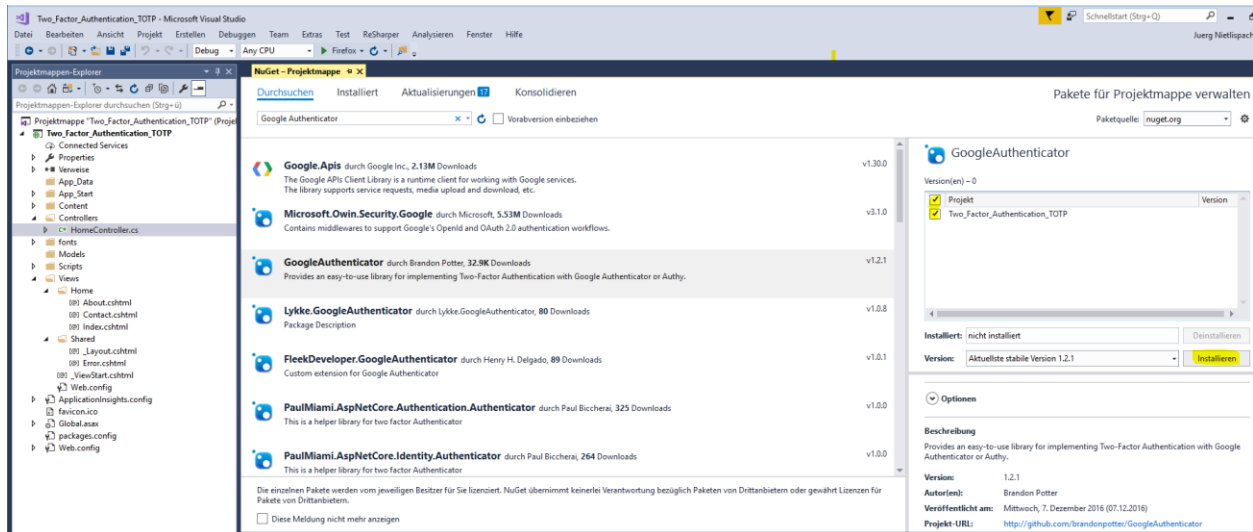
Generierung und Austausch eines Secret Keys

Damit das Backend und der Device identische Tokens aufgrund des aktuellen Zeitstempels generieren können, müssen diese zuerst einen Secret-Key austauschen.

Da wir Google Authenticator Verwenden, können wir hierfür eine bestehende Library für den Austausch des Secret-Keys sowie die Tokengenerierung verwenden. Der NuGet-Paketmanager kann hierfür zur Hilfe genommen werden:



Beim Packetmanager kann nach Google Authenticator gesucht werden und die entsprechende Library installiert werden.



Auf der Projekt-URL sind Zusatzinformationen zur Library und der Implementierung erhältlich:
<https://github.com/brandonpotter/GoogleAuthenticator>

Beim MVC-Template kann nun eine Setup-Routine erstellt werden, welche einerseits einen QR-Code generiert bzw. andererseits ein Code für die manuelle Eingabe beim Google Authenticator zur Verfügung stellt.

Im HomeController kann nun diese Setupfunktion ergänzt werden und die Library entsprechend angesprochen werden:

```
public ActionResult SetupAuthentication()
{
    // Replace all UPPERCASE Variables with custom variables

    TwoFactorAuthenticator tfa = new TwoFactorAuthenticator();
    var setupInfo = tfa.GenerateSetupCode("MY_MVC_APP", "MY_EMAIL_ADDRESS", "MY_SECRET_KEY", 300, 300);

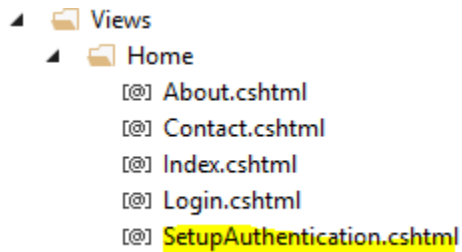
    string qrCodeImageUrl = setupInfo.QrCodeSetupImageUrl;
    string manualEntrySetupCode = setupInfo.ManualEntryKey;

    ViewBag.Message = "<h2>QR-Code:</h2> <br><br> <img src='" + qrCodeImageUrl +
        "' /> <br><br><h2>Token for manual entry</h2> <br>" + manualEntrySetupCode;

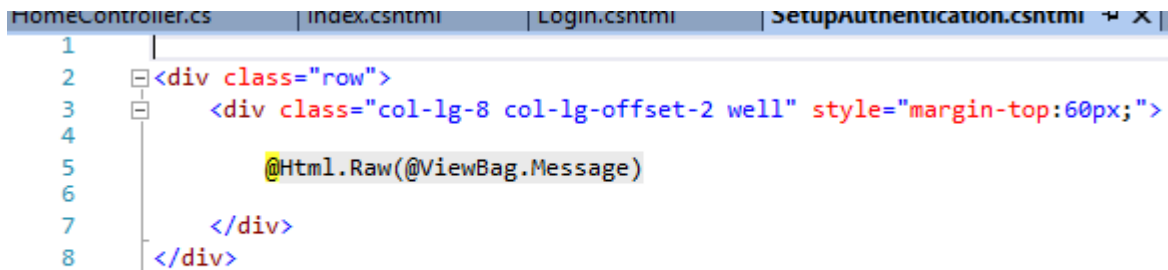
    return View();
}
```

Die Informationen in Grossbuchstaben müssen entsprechend dem aktuellen Kontext angepasst werden. Der Secret-Key hat eigene Kryptographische Eigenschaften, der Einfachheit halber beschränken wir uns einmal auf „MY_SECRET_KEY“.

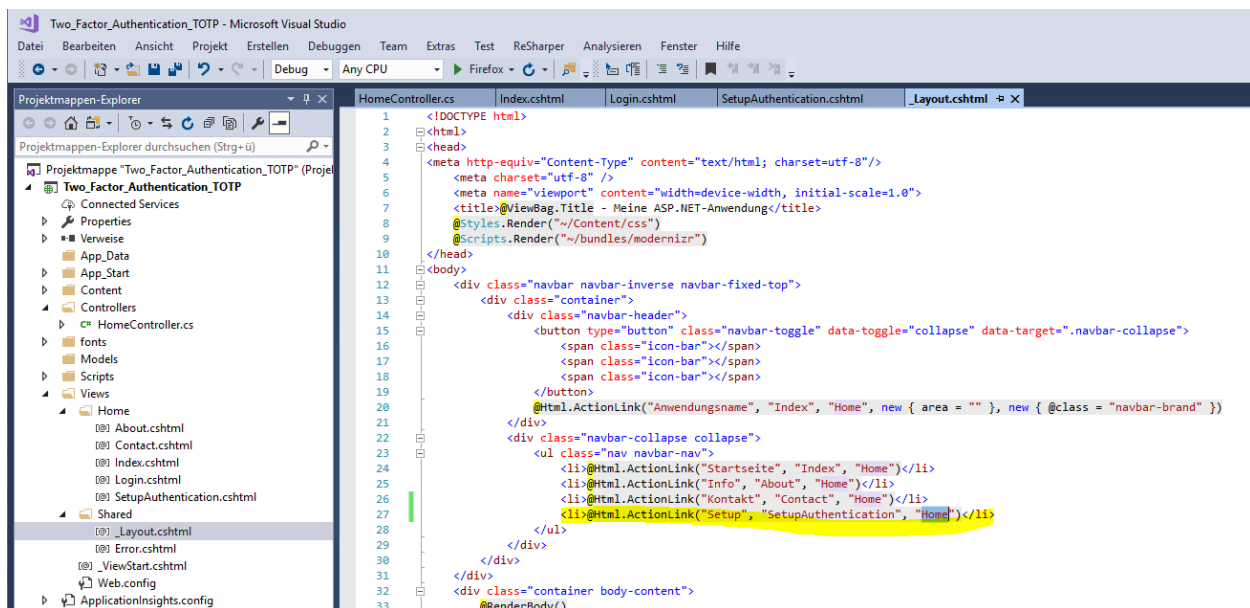
Es muss nun noch ein Template erstellt werden, welches denselben Namen trägt, wie der Methodennamen:



Der Inhalt des Templates kann so gesetzt werden:



Im Browser sieht das dann so aus, wenn man den Setup-Link noch im _Layout.cshtml ergänzt:



QR-Code:



Token for manual entry

JVMV6U2FINJEKVC7JNCVS

Der Austausch dieses Keys erfolgt entweder durch manuelle Eingabe oder durch scannen eines QR-Codes.

Der QR-Code kann alternativ auch mit dieser Webseite generiert werden und vom Device da gescannt werden: <http://www.qr-code-generator.com/>

www.qr-code-generator.com

zieren Sie Ihre Lesezeichen hier in der Lesezeichenleiste, um schnell auf sie zugreifen zu können. Lesezeichen jetzt importieren...

GET QR CODE GENERATOR PRO

LOG IN SIGN UP

Create your QR Code for free

URL VCard Text E-mail SMS Facebook PDF MP3

App stores Images Multi URL

Website (URL)

MY_SECRET_KEY

☒ Static ☐ Dynamic [What does dynamic mean?](#)

Create QR code

JPG EPS SVG

Download

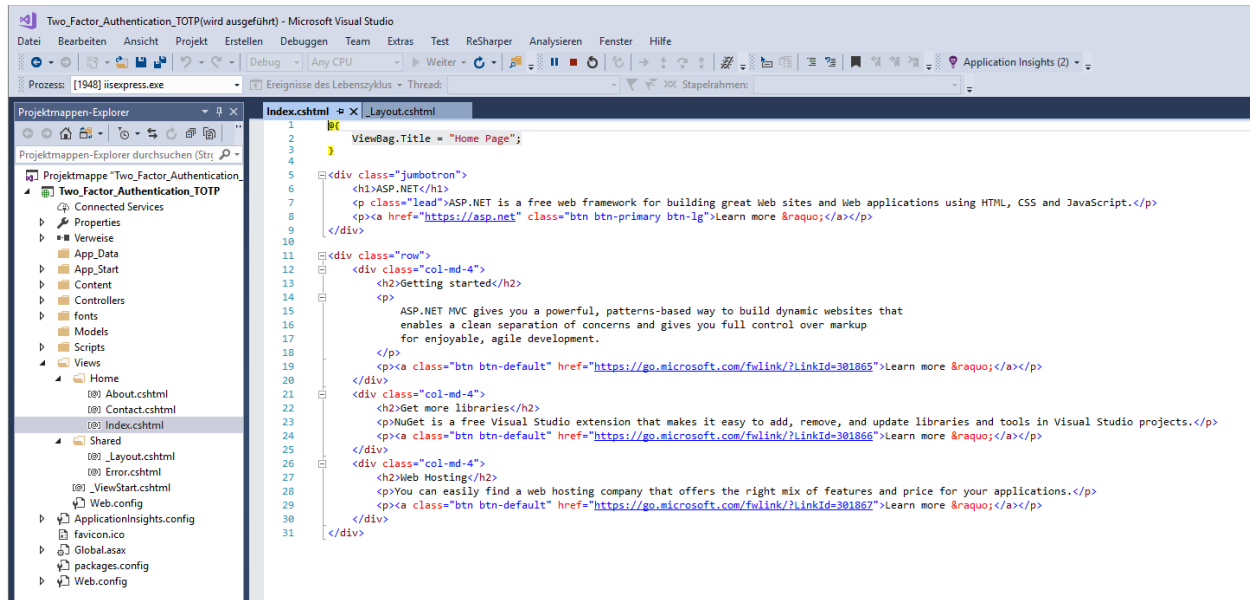
Embed QR code

With Logo?

Dieser Secret Key ist nun auf dem Device gespeichert und kann entsprechende Tokens generieren – wir müssen dann später noch dafür sorgen, dass dieser Key auch noch in irgend einer Form im Backend hinterlegt wird.

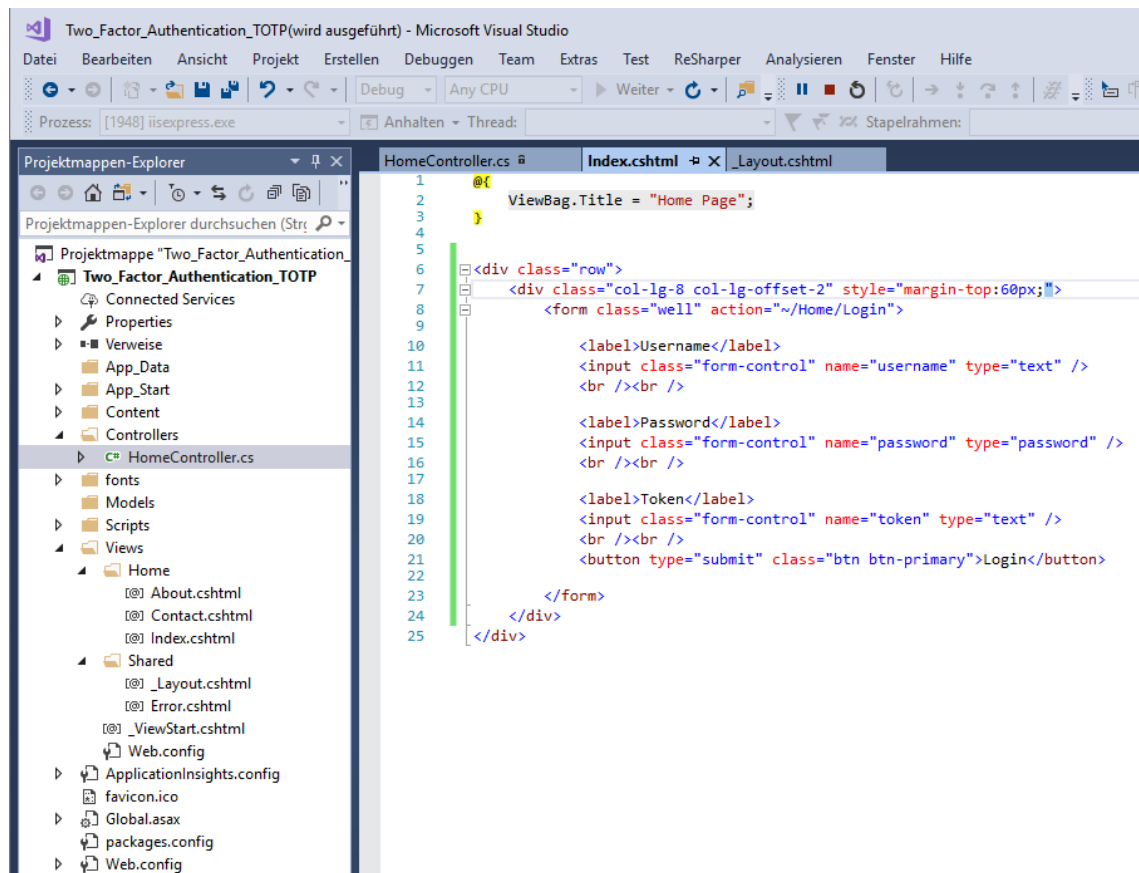
Login-Prozedere

Visual Studio hat aus dem Setup eine MVC-Vorlage mit Login-Maske erstellt – das index.cshtml in Views/Home sieht folgendermassen aus:

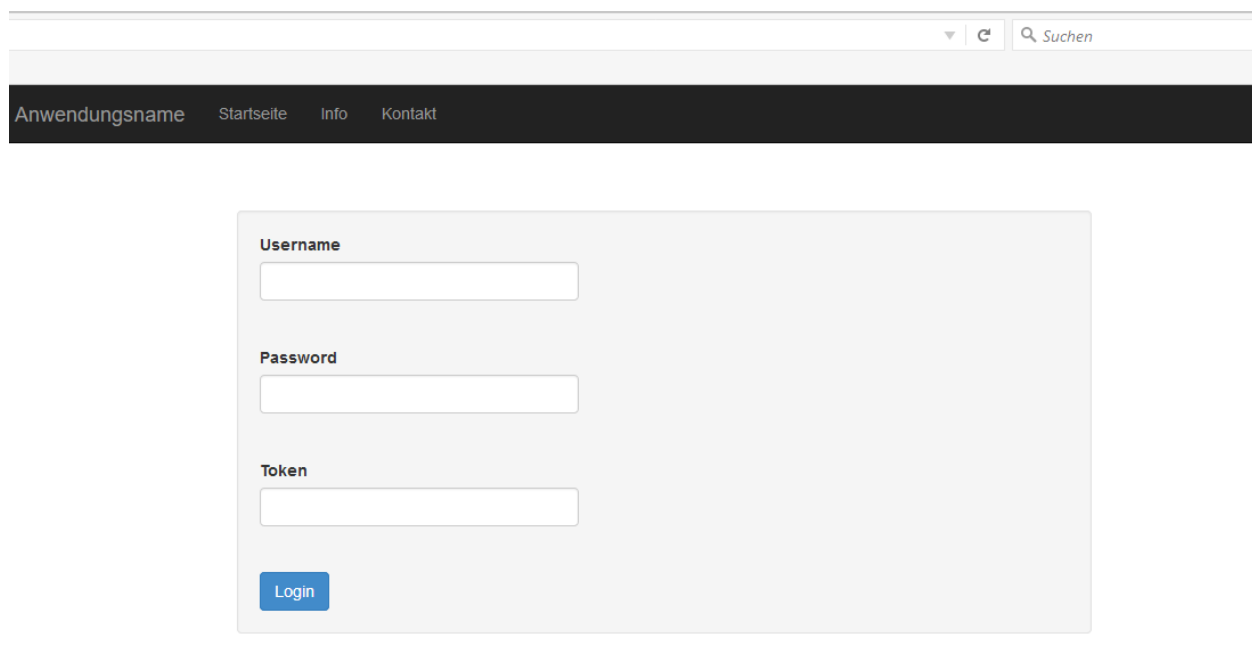


```
1  ViewBag.Title = "Home Page";
2
3
4
5  <div class="jumbotron">
6      <h1>ASP.NET</h1>
7      <p class="lead">ASP.NET is a free web framework for building great Web sites and Web applications using HTML, CSS and JavaScript.</p>
8      <p><a href="https://asp.net/" class="btn btn-primary btn-lg">Learn more &raquo;</a></p>
9  </div>
10
11  <div class="row">
12      <div class="col-md-4">
13          <h2>Getting started</h2>
14          <p>
15              ASP.NET MVC gives you a powerful, patterns-based way to build dynamic websites that
16              enables a clean separation of concerns and gives you full control over markup
17              for enjoyable, agile development.
18          </p>
19          <p><a class="btn btn-default" href="https://go.microsoft.com/fwlink/?LinkId=301865">Learn more &raquo;</a></p>
20      </div>
21      <div class="col-md-4">
22          <h2>Get more libraries</h2>
23          <p>NuGet is a free Visual Studio extension that makes it easy to add, remove, and update libraries and tools in Visual Studio projects.</p>
24          <p><a class="btn btn-default" href="https://go.microsoft.com/fwlink/?LinkId=301866">Learn more &raquo;</a></p>
25      </div>
26      <div class="col-md-4">
27          <h2>Web Hosting</h2>
28          <p>You can easily find a web hosting company that offers the right mix of features and price for your applications.</p>
29          <p><a class="btn btn-default" href="https://go.microsoft.com/fwlink/?LinkId=301867">Learn more &raquo;</a></p>
30      </div>
31  </div>
```

Dies ist der Ort, wo man die Loginmaske platzieren kann. Das Login-Formular sieht dann in HTML so aus:

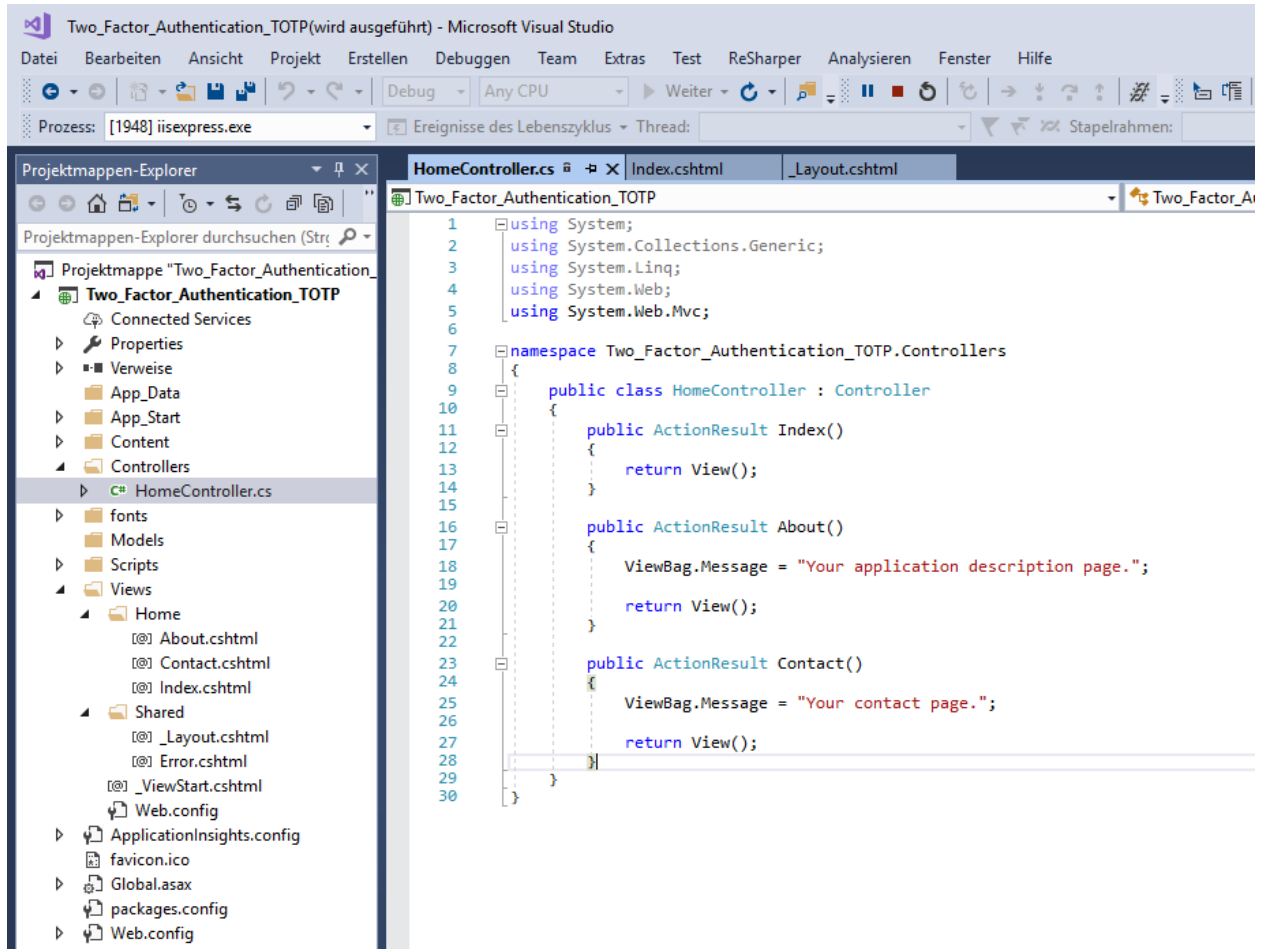


Lässt man die Applikation laufen im Browser, wird die Login-Maske entsprechend angezeigt:



Es gäbe auch hier die Möglichkeit, zuerst Benutzernamen und Passwort abzufragen. Der Einfachheit halber prüfen wird das Time-Based-Token aber gerade in diesem Schritt.

Beim Absenden des Formulars müssen diese Werte nun geprüft werden. Dies Passiert im HomeController.



Dieser muss nun noch mit einer Login-Funktion erweitert werden:

```
}
[HttpPost]
public ActionResult Login()
{
    return View();
}
```

In dieser Funktion werden nun die Parameter für Benutzernamen, Passwort und Token abgefangen und geprüft:

```

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

    if (username == "test" && password == "test")
    {
    }
    else
    {
        ViewBag.Message = "Wrong Credentials";
    }

    return View();
}

```

Typischerweise sind Benutzernamen und Passwort verschlüsselt in einer Datenbank hinterlegt. Der Einfachheit halber, werden diese hier im Code direkt hinterlegt und geprüft. Da die Methode eine View()-Instanz retourniert müssen wir noch ein Login.cshtml-File erstellen:

```

Views
└─ Home
    [@] About.cshtml
    [@] Contact.cshtml
    [@] Index.cshtml
    [@] Login.cshtml

```

Stimmen Benutzernamen und Passwort überein, muss das Token noch geprüft werden. Dies erfolgt nun mittels derselben Library, welche wir für Setup bereits verwendet haben. Die Methode wird dann folgendermassen ergänzt:

```

[HttpPost]
public ActionResult Login()
{
    // Replacee all UPPERCASE Variables with custom variables

    var username = Request["username"];
    var password = Request["password"];
    var token = Request["token"];

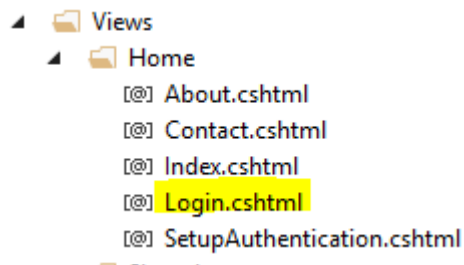
    if (username == "test" && password == "test")
    {
        TwoFactorAuthenticator tfa = new TwoFactorAuthenticator();
        bool isCorrectPIN = tfa.ValidateTwoFactorPIN("MY_SECRET_KEY", token);

        if (isCorrectPIN)
        {
            ViewBag.Message = "Login and Token Correct"; // and redirect for session handling
        }
        else
        {
            ViewBag.Message = "Wrong credentials and token";
        }
    }
    else
    {
        ViewBag.Message = "Wrong Credentials";
    }

    return View();
}

```

Da wiederum auch hier wieder eine View-Instanz retourniert wird, müssen wir der Vollständigkeit halber auch hier wieder ein Template erstellen.



Entweder wird beim erfolgreichen Login eine Weiterleitung im Code hinterlegt, oder die Fehlermeldung ins GUI geschrieben.

Beim Login werden nun Benutzernamen, Passwort und das auf dem Handy von Google Authenticator aktuell angezeigte Token eingegeben werden.

Username

Password

Token

Login

Mittels Debugger kann nun die Verifikation des Tokens überprüft werden:

```

33
34
35
36
37
38
39
40
41
42
43
44
45 [HttpPost]
46 public ActionResult Login()
47 {
48     // Replacee all UPPERCASE Variables with custom variables
49
50     var username = Request["username"];
51     var password = Request["password"];
52     var token = Request["token"];
53
54     if (username == "test" && password == "test")
55     {
56         TwoFactorAuthenticator tfa = new TwoFactorAuthenticator();
57         bool isCorrectPIN = tfa.ValidateTwoFactorPIN("MY_SECRET_KEY", token);
58
59         if (isCorrectPIN)
60         {
61             ViewBag.Message = "Login and Token Correct"; // and redirect for session handling
62         }
63         else
64         {
65             ViewBag.Message = "Wrong credentials and token";
66         }
67     }
68     else
69     {
70         ViewBag.Message = "Wrong Credentials";
71     }
72
73     return View();
74 }
75
76
77

```

ACHTUNG: die Systemzeit von Device und Backend muss übereinstimmen! Bitte synchronisieren, sonst wird die Tokenüberprüfung fehlschlagen.