LB3 M183
DOKUMENTATION FLEXLINE

Severin Gafner & Aris Javet

Inhalt

Übersicht

Das Projekt wurde mit der Webtechnologie ASP.NET MVC und der dementsprechenden Programmiersprache C# entwickelt. Dabei wurden ausserdem auf die Technologien Entity Framework, Identity Framework, HTML, CSS, JavaScript und JQuery zurückgegriffen. Als Datenbank wurde eine MSSQL Datenbank erstellt.

Login

Da das Projekt mit dem Identity Framework realisiert wurde, mussten diverse Konfigurationen gemacht werden. Zu aller erst wurde ein Controller erstellt, welcher die Login- und Registrierungsanfragen entgegennimmt. Dieser Controller wurde «AccountController» genannt.

Anschliessend musste die Applikation konfiguriert werden, so dass alle Funktionen, welche ein Login benötigen, als geschützte Ressource markiert werden. Wenn nun eine nicht angemeldete Anfrage auf eine geschützte Ressource stösst, werden diese auf die Login Seite weitergeleitet.

Danach wurde das Identity konfiguriert. Hierbei wurde definiert, welche Klasse durch das Identity verwaltet werden soll.

Neuer Benutzer Registrieren

Da beim Schritt Login bereits viele Konfigurationen im Zusammenhang mit dem Identity Framework gemacht wurden, musste hier nicht mehr viel gemacht werden.

Was noch fehlte waren die Funktionen, welche die Registrierungsanfragen entgegennehmen. Dies wurde ebenfalls im «AccountController» ergänzt.

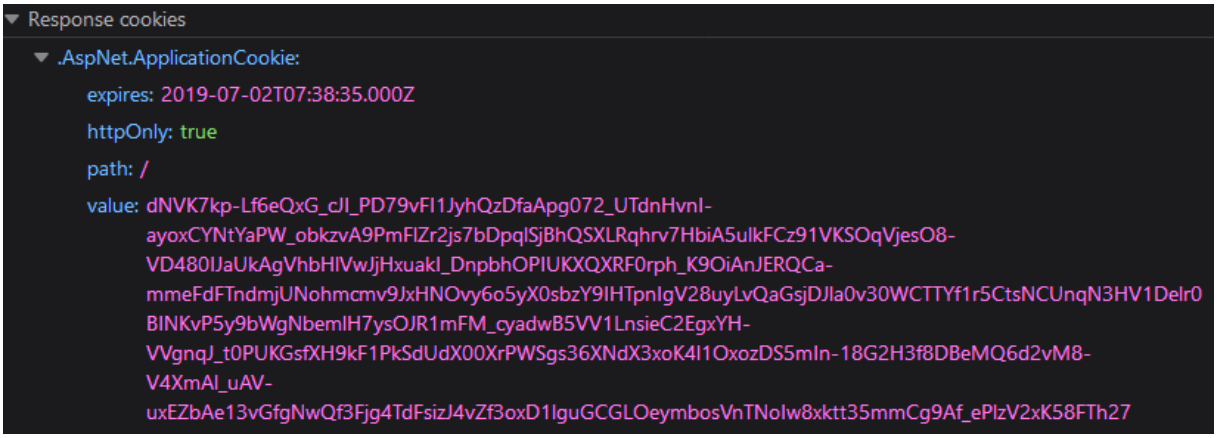
Schlussendlich wurden noch die Anforderungen an ein Passwort definiert also wie lange muss das Passwort sein, welche Zeichen müssen enthalten sein, usw. Dies waren ebenfalls Konfigurationen auf Seiten Identity Framework.

```
// Configure validation logic for passwords
manager.PasswordValidator = new PasswordValidator
{
    RequiredLength = 6,
    RequireNonLetterOrDigit = true,
    RequireDigit = true,
    RequireLowercase = true,
    RequireUppercase = true,
};
```

Session Handling

Das Identity Framework regelt das Session Handling. Wenn eine erfolgreiche Registration gemacht wird, wird ein Cookie im Response Header gesetzt. Wird nun auf eine geschützte Ressource zugegriffen prüft das Framework, ob das Cookie ein gültiges Cookie ist und macht so die Verbindung von Cookie zu Benutzer.

Im untenstehenden Screenshot wird das Response Cookie angezeigt, welches nach einer erfolgreichen Registration geschickt wird.



Abfragen von Daten einer DB-Tabelle per Suchfeld

Unser Projekt ist dazu da Lieblingsfilme zu verwalten. Um dies auf Applikationsebene darzustellen, wurde der «MoviesController» erstellt. Da wir nicht wollen, dass jede Person Filme erstellen kann, wurde dieser Controller durch ein Login geschützt «[Authorize]». Die Funktionalität der Suche wurde dann in der Index Methode des Controllers geregelt.

```
[Authorize]
0 references | 2 requests
public class MoviesController : Controller
{
    private ApplicationDbContext db = new ApplicationDbContext();

    // GET: Movies
    [ValidateInput(false)]
    0 references | 2 requests | 0 exceptions
    public async Task<ActionResult> Index(string searchString)
    {
        IQueryable<Movie> movies = db.Movies.OrderBy(m => m.Name);

        if (!string.IsNullOrEmpty(searchString))
            movies = movies.Where(m => m.Name.Contains(searchString));

        var vm = new MoviesViewModel
        {
            SearchString = searchString,
            Movies = await movies.ToListAsync()
        };
        return View(vm);
    }
```

Sichere Passwortspeicherung mit Pepper

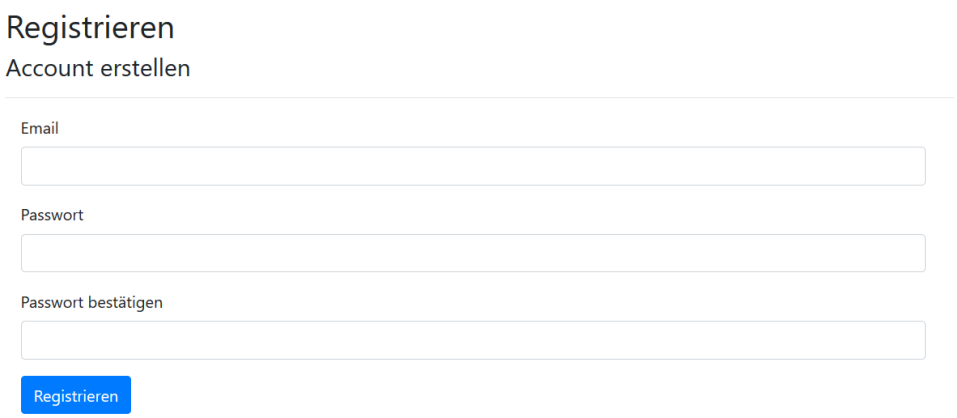
Das IdentityFramework sichert die Passwörter standardmässig nur mit einem indeviduellen Salt. Da dies für uns zu unsicher war, implementierten wir zusätzlich einen Pepper. Dazu wählten wir eine zufällig generierte GUID «Globally unique identifier». GUIDs sind Zeichenketten aus zufällig erstellten Kombinationen an 32 Zeichen plus den Bindestrichen. Unser Pepper lautet: «bf944798-31a5-470a-830c-0bc3c606f323». Eine solche Kombination herauszufinden grenzt an das unmögliche. Um ganz sicher zu gehen, dass das Passwort auch lange genug ist, hängten wir diese Zeichenkette am Anfang und zum Schluss des eingegebenen Passwortes an.

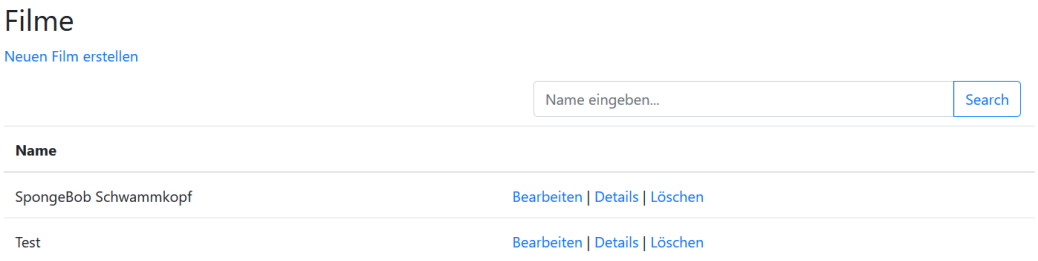
```
model.Password = $"{_pepper}{model.Password}{_pepper}";
var user = new ApplicationUser { UserName = model.Email, Email = model.Email };
var result = await UserManager.CreateAsync(user, model.Password);
```

Das heisst für unsere Applikation, dass die Mindestgrösse eines gehashten Passwortes eines Passwortes 78 Zeichen lang ist (6 (min. länge Identity Framework Konfiguration) + 36 (Pepper) + 36 (Pepper) = 78).

Sinnvolles und Ansprechendes GUI

Um das GUI ansprechend zu gestallten setzten wir das CSS-Framework Bootstrap ein. Folgendermassen sieht unsere Applikation aus:





Dies sind nur die wichtigsten Seiten. Damit das Dokument nicht zu gross wird wurden die kleineren ausgelassen.

XSS Injection vorbeugen

Das Entity Framework ist standardmässig gegen XSS Injection geschützt. Heisst, dass man keine Strings mit HTML Annotationen eintragen kann. Da wir aus Demozwecken aber wollten, dass man solche Strings speichern kann, mussten wir dies speziell kennzeichnen. Der Name der Klasse Movie musste die Annotation «AllowHtml» erhalten. Zusätzlich musste bei der Index Methode der Filme die Annotation «ValidateInput(false)» hinzugefügt werden. Nun war es möglich Namen mit Html Tags einzutragen und danach zu suchen. Diese werden aber in einer harmlosen Form angezeigt.

```
[AllowHtml]
2 references | 0 exceptions
public string Name { get; set; }
```

```
[ValidateInput(false)]
```