25 Appendix B : Oplossing oefeningen

25.1 Fibonacci-reeks (p. 37)

```
@{
    var i = 0;
    var j = 1;
    var tussen = 0;

    <h2>Rij van Fibonacci</h2>
    <p>@i, @j
    @while (tussen < 100)
    {
       tussen = i + j;
       if (tussen < 100) {
         @:, @tussen
       }
       i=j;
       j=tussen;
    }
    </p>
}
```

25.2 Bieren (p. 44)

De actionmethod:

```
public ActionResult Index()
{
    var bieren = new List<Bier>();
    bieren.Add(new Bier
    {
        ID = 15,
        Naam = "Felix",
        Alcohol = 7
    });
    bieren.Add(new Bier
    {
        ID = 17,
        Naam = "Roman",
        Alcohol = 7.5F
    });
    return View(bieren);
}
```

De view:

```
@model List<MVCBierenApplication.Models.Bier>
<h2>Alle bieren</h2>
@foreach (var bier in Model)
{
    @bier.ID@:. @bier.Naam
    @: (@bier.Alcohol
    @:%)
    <br/>
}
```

25.3 Logo Biertempel en adresgegevens (p.57)

_Layout.cshtml :

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>@ViewBag.Title</title>
    @Styles.Render("~/Content/css")
    @Scripts.Render("~/bundles/modernizr")
</head>
<body>
    <img src="@Url.Content("~/Images/biertempel.png")" alt="logo"/>
    @RenderBody()
    <hr />
    <p>- De Biertempel - Hoppestraat 103 - 3360 Bierbeek - tel. 016123456 - email:
info@@biertempel.be -</p>

    @Scripts.Render("~/bundles/jquery")
    @Scripts.Render("~/bundles/bootstrap")
    @RenderSection("scripts", required: false)
</body>
</html>
```

25.4 Extra opmaak (p.67)

In een nieuwe map *Views/Shared/DisplayTemplates* voeg je een view *kleuren.cshtml* toe :

```
@model float
@{
   if (Model < 2) {
     <span class="groen">@Model %</span>
   }
   else if (Model < 5) {
    <span class="geel">@Model %</span>
   }
   else if (Model < 8)
   {
   <span class="oranje">@Model %</span>
   }
   else
   {
   <span class="rood">@Model %</span>
   }
}
```

Voeg in *Site.css* (folder *Content*) de volgende css-code toe :

```
.groen { color: green; }
.geel { color: yellow; }
.oranje { color: orange; }
.rood { color: red; }
```

Wijzig de class *Bier.cs* als volgt :

```
using System.ComponentModel.DataAnnotations;
…
public class Bier
{
    [DisplayFormat(DataFormatString = "{0:000}")]
    public int ID { get; set; }
    public String Naam { get; set; }
    [UIHint("kleuren")]
    public float Alcohol { get; set; }
}
```

In *Bier/Index.cshtml* kan je de opmaak nu gebruiken :

```
@model List<MVCBierenApplication.Models.Bier>
<h2>Alle bieren</h2>

@foreach (var bier in Model)
{
  <p>@Html.DisplayFor(m=>bier.ID) @bier.Naam @Html.DisplayFor(m=>bier.Alcohol)</p>
}
```

25.5 Een bier verwijderen (p.81)

We wijzigen eerst de webapplicatie zodat de bieren via een bierenservice worden opgehaald.

We voegen een class *BierenService.cs* toe :

```
using MVCBierenApplication.Models;
…
public class BierenService
{
  private static Dictionary<int, Bier> bieren =
    new Dictionary<int, Bier>();

  static BierenService()
  {
    bieren[1] = new Bier { ID = 1, Naam = "Romy pils", Alcohol = 4.5F};
    bieren[2] = new Bier { ID = 2, Naam = "Leffe blond", Alcohol = 2.5F };
    bieren[3] = new Bier { ID = 3, Naam = "Rodenbach", Alcohol = 3.5F };
    bieren[4] = new Bier { ID = 4, Naam = "Liefmans goudenband", Alcohol = 6F };
    bieren[5] = new Bier { ID = 5, Naam = "Duvel", Alcohol = 7F };
  }

  public List<Bier> FindAll() {
    return bieren.Values.ToList();
  }

  public Bier Read(int id)
  {
    return bieren[id];
  }

  public void Delete(int id)
  {
    bieren.Remove(id);
  }
}
```

We voegen in de BierController een private variabele toe voor de bierenservice en wijzigen de Index-action :

```
using MVCBierenApplication.Services;
...
private BierenService bierenService = new BierenService();

public ActionResult Index()
{
  var bieren = bierenService.FindAll();
  return View(bieren);
}
```

In de Index-view die de bieren uitlijst voegen we een hyperlink naar een verwijderactie toe.

```
@model List<MVCBierenApplication.Models.Bier>
<h2>Alle bieren</h2>

@foreach (var bier in Model)
{
    var url = "Verwijderen/" + bier.ID;
    <p>@Html.DisplayFor(m=>bier.ID) @bier.Naam @Html.DisplayFor(m=>bier.Alcohol) <a
href="@url"><img src="~/Images/delete.png" alt="verwijderen" /></a></p>
}
```

In de BierController voegen we een aantal actions toe :

```
      public ActionResult Verwijderen(int ID)
      {
          var bier = bierenService.Read(ID);
          return View(bier);
      }

      [HttpPost]
      public ActionResult Delete(int ID)
      {
          var bier = bierenService.Read(ID);
          this.TempData["bier"] = bier;
          bierenService.Delete(ID);
          return Redirect("~/Bier/Verwijderd");
      }

      public ActionResult Verwijderd()
      {
          var bier = (Bier)this.TempData["bier"];
          return View(bier);
      }
```

De views :

Verwijderen.cshtml :

```
@model MVCBierenApplication.Models.Bier
@{
    ViewBag.Title = "Verwijderen";
}
<h2>@Model.Naam verwijderen?</h2>
<form method="post" action="~/Bier/Delete/@Model.ID">
    <input type="submit" value="Delete" />
</form>
```

Verwijderd.cshtml :

```
@model MVCBierenApplication.Models.Bier
@{
    ViewBag.Title = "Verwijderd";
}
<h2>Verwijdering van @Model.Naam is doorgevoerd.</h2>
<h3>Terug naar de <a href="Index">lijst</a></h3>
```

25.6 Hyperlinks (p. 85)

De index-pagina :

```
@model List<MVCBierenApplication.Models.Bier>
<h2>Alle bieren</h2>

@foreach (var bier in Model)
{
    var url = Url.Action("Verwijderen", "Bieren", new {ID=bier.ID});
    <p>@Html.DisplayFor(m=>bier.ID) @bier.Naam @Html.DisplayFor(m=>bier.Alcohol)
      <a href="@url"><img src="~/Images/delete.png" alt="verwijderen" /></a></p>
}
```

De delete-action in de controller :

```
[HttpPost]
public ActionResult Delete(int id)
{
    var bier = bierenService.Read(id);
    this.TempData["bier"] = bier;
    bierenService.Delete(id);
    return RedirectToAction("Verwijderd");
}
```

25.7 Nieuw bier toevoegen en layout verzorgen (p. 121)

In de bierenservice voeg je volgende method toe :

```
public void Add(Bier b)
{
    b.ID = bieren.Keys.Max() + 1;
    bieren.Add(b.ID, b);
}
```

In Index.cshtml voeg je onderaan een hyperlink toe naar een nog aan te maken action :

```
<p>@Html.ActionLink("Bier toevoegen", "Toevoegen")</p>
```

Voeg in BierController.cs onderstaande actionmethod toe waarmee het toevoegformulier wordt getoond :

```
[HttpGet]
public ActionResult Toevoegen()
{
    var bier = new Bier();
    return View(bier);
}
```

De bijhorende view :

```
@model MVCBierenApplication.Models.Bier
@{
    ViewBag.Title = "Toevoegen";
}
<h2>Bier toevoegen</h2>
@using (Html.BeginForm("Toevoegen"))
{
    @Html.LabelFor(m => m.Naam)
    @Html.EditorFor(m => m.Naam)
    @Html.LabelFor(m => m.Alcohol)
    @Html.EditorFor(m => m.Alcohol)
    <input type="submit" value="Toevoegen" />
}
```

In de BierController voegen we dan nog een Post-method toe die het bier effectief toevoegt :

```
        [HttpPost]
        public ActionResult Toevoegen(Bier b)
        {
            bierenService.Add(b);
            return RedirectToAction("Index");
        }
```

Om de layout aan te passen voegen we de "zebra"-class toe aan de tabel waarin de bieren worden getoond. De bijhorende css-code voegen we toe aan Site.css (zie cursus) alsook de css-code die de input in een block zet. De index-view wordt nu :

```
@model List<MVCBierenApplication.Models.Bier>
<h2>Alle bieren</h2>
<table class="zebra">
<thead>
<tr>
<th>ID</th>
<th>Naam</th>
<th>Alcohol</th>
<th></th>
</tr>
</thead>
<tbody>
@foreach (var bier in Model)
{
    var url = Url.Action("Verwijderen", "Bier", new { ID = bier.ID });
    <tr>
    <td>@Html.DisplayFor(m=>bier.ID)</td>
    <td>@bier.Naam</td>
    <td>@Html.DisplayFor(m=>bier.Alcohol)</td>
    <td>
        <a href="@url"><img src="~/Images/delete.png" alt="verwijderen" /></a>
    </td>
    </tr>
}
</tbody>
</table>
@Html.ActionLink("Bier toevoegen", "Toevoegen")
```

Voor de validatie voegen we aan de class *bier.cs* een aantal data-annotations toe :

```
public class Bier
{
    [DisplayFormat(DataFormatString = "{0:000}")]
    public int ID { get; set; }
    [Required]
    [StringLength(20, ErrorMessage="Max. {1} tekens voor {0}")]
    public string Naam { get; set; }
    [UIHint("kleuren")]
    [AlcoholGrenzen(ErrorMessage="{0} heeft een ongeldige waarde")]
    public float Alcohol { get; set; }
}
```

In de BierController activeren we de validatie :

```
        [HttpPost]
        public ActionResult Toevoegen(Bier b)
        {
            if (this.ModelState.IsValid)
            {
                bierenService.Add(b);
                return RedirectToAction("Index");
            }
            else
                return View(b);
        }
```

En in de view voegen we validationmessages toe en een validationsummary :

```
<h2>Bier toevoegen</h2>
@using (Html.BeginForm("Toevoegen"))
{
    @Html.LabelFor(m => m.Naam)
    @Html.ValidationMessageFor(m => m.Naam, "*")
    @Html.EditorFor(m => m.Naam)
    @Html.LabelFor(m => m.Alcohol)
    @Html.ValidationMessageFor(m => m.Alcohol, "*")
    @Html.EditorFor(m => m.Alcohol)
    <input type="submit" value="Toevoegen" />
    @Html.ValidationSummary()
}
```

Vergeet in Site.css de validationstyles niet toe te voegen.

Om het alcoholpercentage te kunnen valideren voegen we een class `AlcoholGrenzenAttribute` toe :

```
public class AlcoholGrenzenAttribute : ValidationAttribute
{
    public override bool IsValid(object value)
    {
        if (value == null)
            return true;
        if (!(value is float))
            return false;
        var alcoholwaarde = (float)value;
        return ((alcoholwaarde < 15) && (alcoholwaarde > 0));
    }
}
```

25.8 Bierendata uit databank (p. 131)

Voeg in de folder *Models* een Entity Data Model toe met de naam *MVCBieren.edmx*. Je gebruikt best niet de naam *Bieren.edmx* omdat dit ook de naam is van één van de tabellen in de database. Verwijder de oude class Bier.cs en breng daarna de nodige correcties aan aan de entitynamen en navigation properties in het Entity Data Model.

In de BierenService haal je de static variabele bieren weg en ook de static constructor.

Wijzig de methods in BierenService.cs als volgt :

```
public List<Bier> FindAll()
{
    using (var db = new MVCBierenEntities())
    {
        return db.Bieren.ToList();
    }
}

public Bier Read(int id)
{
    using (var db = new MVCBierenEntities())
    {
        return db.Bieren.Find(id);
    }
}

public void Delete(int id)
{
    using (var db = new MVCBierenEntities())
    {
        Bier bier = db.Bieren.Find(id);
        db.Bieren.Remove(bier);
        db.SaveChanges();
    }
}

public void Add(Bier b)
{
    using (var db = new MVCBierenEntities())
    {
        db.Bieren.Add(b);
        db.SaveChanges();
    }
}
```

Hier en daar moeten we nog enkele aanpassingen aanbrengen in views. In Views/Bier/Index.cshtml staat enkele keren bier.ID i.p.v. bier.BierNr. Bijvoorbeeld in de url naar de verwijderview een id die op bier.ID wordt gezet. Dit wordt bier.BierNr :

```
var url = Url.Action("Verwijderen", "Bier", new { ID = bier.BierNr });
```

Ook in *Verwijderen.cshtml* vervangen we ID door Biernr :

```
<form method="post" action="~/Bier/Delete/@Model.BierNr">
```

Het toevoegen van een bier zal nu niet meer lukken omdat er geen brouwernr en ook geen soortnr wordt opgegeven. Het toevoegen volledig uitwerken met keuzelijsten voor brouwer en soort, zou

ons te ver leiden. Je kan je beperken tot het toevoegen van twee extra textboxen in de view *Toevoegen.cshtml*.

Door een edmx-bestand te maken en de oorspronkelijke class Bier.cs te verwijderen is de opmaak van het biernr (DisplayFormat) en het alcoholpercentage (UIHint) verdwenen. We voegen dit opnieuw toe via het principe van de buddy classes.

Voeg in de folder *Models* een class *BierProperties* toe :

```
public class BierProperties
{
    [DisplayFormat(DataFormatString = "{0:000}")]
    public int BierNr { get; set; }
    [Required]
    [StringLength(20, ErrorMessage = "Max. {1} tekens voor {0}")]
    public string Naam { get; set; }
    [UIHint("kleuren")]
    [AlcoholGrenzen(ErrorMessage = "{0} heeft een ongeldige waarde")]
    public float Alcohol { get; set; }
}
```

We linken deze class aan de class Bier.cs via een extra partial class BierUitbreiding.cs :

```
[MetadataType(typeof(BierProperties))]
public partial class Bier
{
}
```

Als we de webapplicatie nu uitproberen krijgen we een fout omdat het alcoholpercentage van sommige bieren niet is ingevuld. We moeten dit opvangen in de view *Index.cshtml* :

```
…
@if (bier.Alcohol != null)
{
    @Html.DisplayFor(m=>bier.Alcohol)
}
else
{
    <span>Niet gekend</span>
}
…
```

25.9 Beveiligde bierenapplicatie (p. 203)

Maak een nieuwe MVC Internet Application en voeg de functionaliteiten uit de vorige oefening toe. Voeg een rol Administrators toe en zorg ervoor dat één van de users tot deze rol behoort zodat je kan uittesten. Aan de actionmethods die betrekking hebben tot het toevoegen en verwijderen van een bier voeg je de annotation `[Authorize(Roles = "Administrators")]` toe.

COLOFON

Domeinexpertisemanager	Jean Smits
Moduleverantwoordelijke	Steven Lucas
Auteurs	Steven Lucas
Versie	21/06/2017

Omschrijving module-inhoud

Abstract	Doelgroep	.NET ontwikkelaar met C#
	Aanpak	Begeleide zelfstudie
	Doelstelling	Webapplicaties leren maken met ASP.NET MVC
Trefwoorden		ASP.NET MVC C#
Bronnen/meer info		http://www.asp.net/mvc http://www.microsoftvirtualacademy.com/