Oefening 2

Doelstellingen:

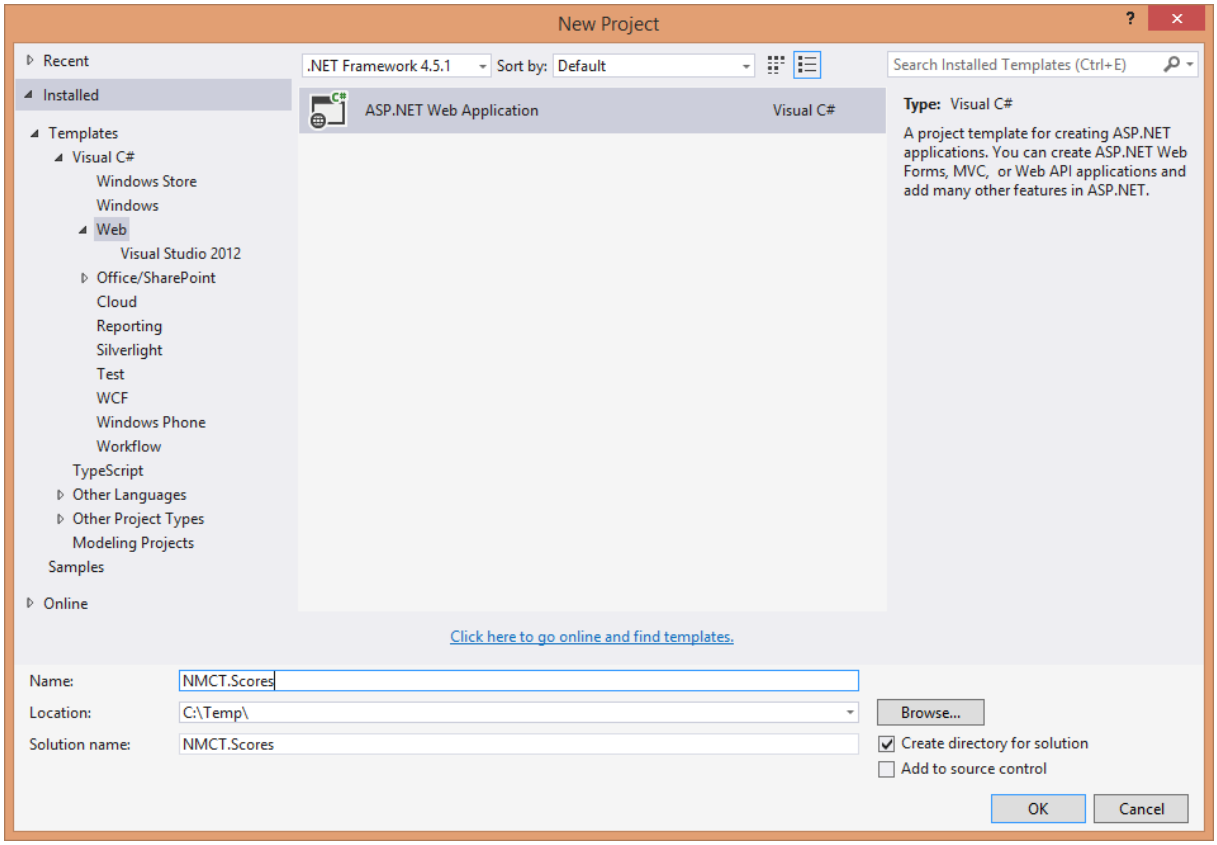
- Herhaling ASP.NET MVC semester 3
- Leren werken met Entity Framework 6 (EF6)
- Basis LINQ query' schrijven.

Manier van werken

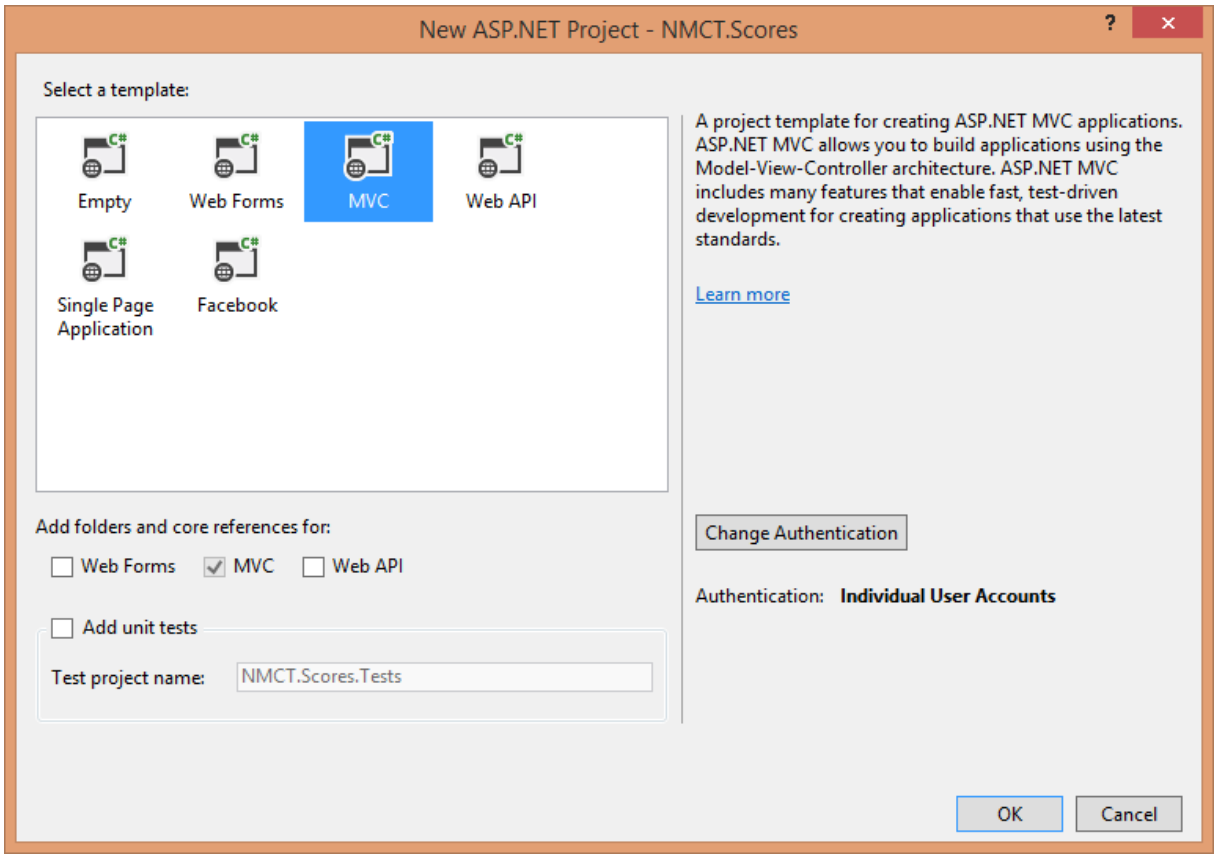
In dit labo zijn een aantal zaken volledig uitgeschreven en moet u een aantal zaken deels zelf zoeken of op het einde volledig zelf zoeken. Maak hiervoor gebruik van de theorie en het internet.

Aanmaken project

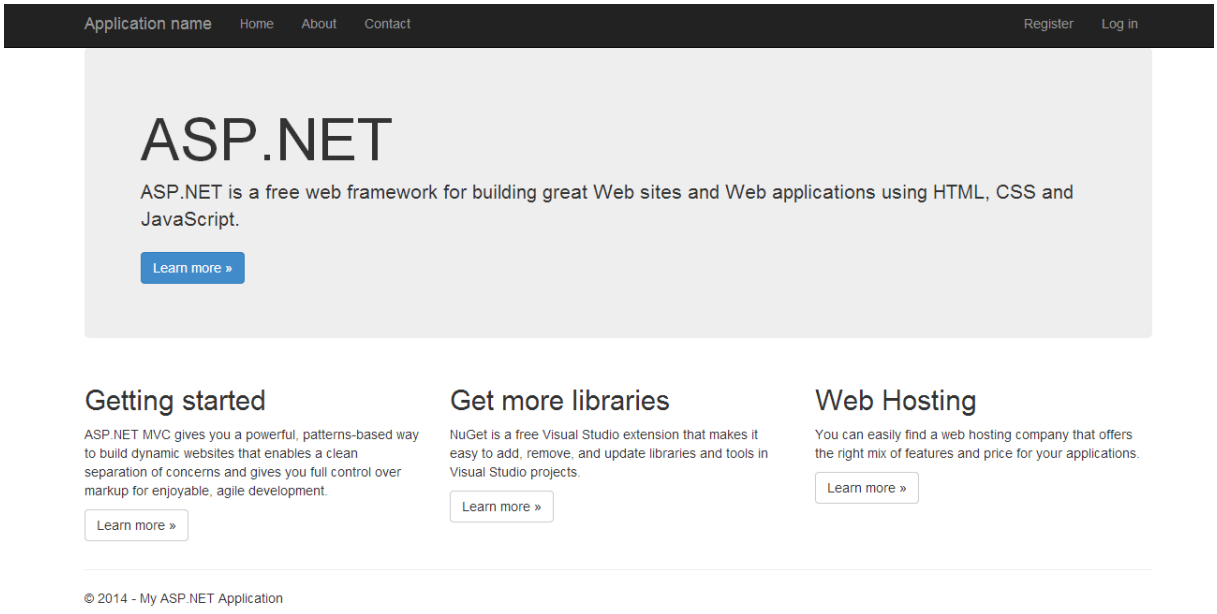
We maken een nieuw ASP.NET MVC 5 project aan vanuit Visual Studio 2013 met als naam **NMCT.Scores**.



In het volgende scherm kiezen we voor MVC



Na het aanmaken van het project controleer je het project of het werkt door het te starten. Je zou onderstaande scherm moeten zien:

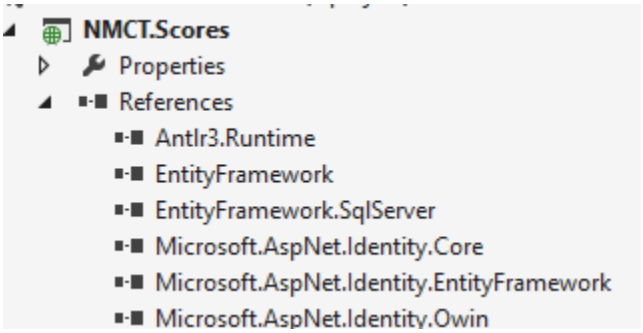


Models

In de opgave zitten de models die je nodig hebt in het project. Sleep deze models in het project in de map Models. Controleer de namespace van de models of deze wel hetzelfde is als de rest van de applicatie.

Entity Framework

In deze applicatie maken we gebruik van EF 6 code first. Aangezien we een ASP.NET MVC 5 toepassing aanmaken is Entity Framework standaard aanwezig. Je kan dit zien in de solution explorer bij references (controleer dit).



EF gebruikt een context om te communiceren met de databank. Maak een folder DAL aan onder Models. Daarin maken we een file “ScoreContext.cs” aan. De klasse ScoreContext moet **erven** van DbContext. Daarnaast moeten we ook opgeven welke tabellen we wensen in onze database. Hiervoor moeten we public DbSet<T> gebruiken en op deze manier onze tabellen publiek maken.

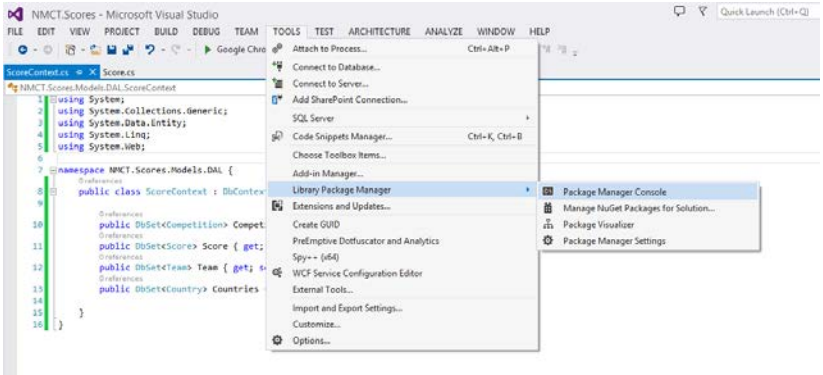
```
0 references
public class ScoreContext : DbContext {

    0 references
    public DbSet<Competition> Competition { get; set; }
    0 references
    public DbSet<Score> Score { get; set; }
    0 references
    public DbSet<Team> Team { get; set; }
    0 references
    public DbSet<Country> Countries { get; set; }

}
```

Als laatste moeten we in web.config opgeven naar welke database EF moet connecteren. De naam van de connectionstring moet hetzelfde zijn als de naam van de context klasse. In dit voorbeeld ScoreContext.

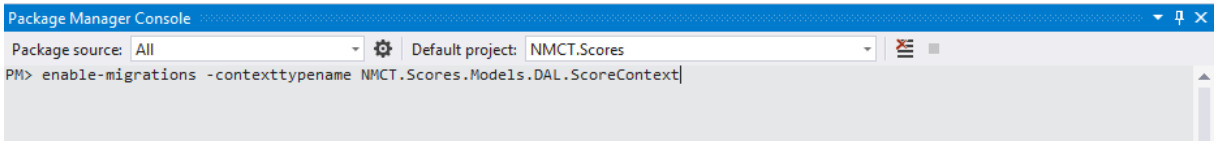
```
<connectionStrings>
  <add name="ScoreContext" connectionString="data source=.;Initial Catalog=ScoreApp;Integrated Security=SSPI" providerName="System.Data.SqlClient"/>
</connectionStrings>
```

Zorg er nu voor dat alles compileert.

De laatste stap is het aanmaken van de database zelf. Dit gaan we doen via code first migrations. Via “Tools” => “Library Package Manager” => “Package Manager Console” openen we de package manager command line. Deze console werkt op basis van powershell commands.

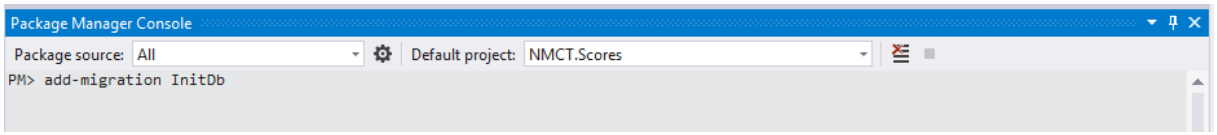


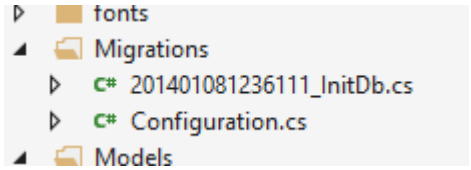
Via het powershell commando “enable-migrations” activeren we EF migrations in het project. We moeten wel opgeven over welke context het gaat. Dit doen we via het argument –contexttypename gevolgd door de volledige naam (Fully Qualified Name) van de context (inclusief namespace).



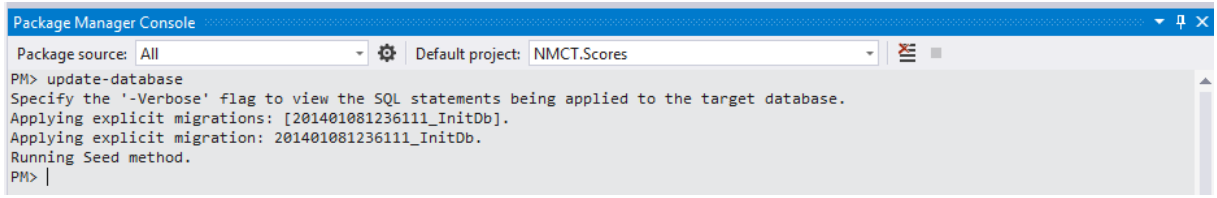
Als alles goed verloopt zal EF migrations een configuration file aanmaken. In deze file is een Seed() methode aanwezig. Deze methode zal worden uitgevoerd als we een migratie uitvoeren. We kunnen dan bvb test data toevoegen aan de database. In de opgave kan je deze methode vinden. Voeg deze toe en zorg ervoor dat alles compileert. **Bestudeer deze methode. Voeg eventueel zelf een nieuwe competitie toe en wat nieuwe ploegen.**

Nu moeten we nog een migratie toevoegen. De eerst keer dat we dit doen zal ook de database aangemaakt worden. We noemen dit ook soms de startupmigratie. Via het powershell commando “Add-migration” gevolgd door een zelf gekozen naam zal een migratie file worden aangemaakt. Deze file zal ook een timestamp bevatten. Als je deze file opent zal je zien dat EF via C# functions de volledige database zal aanmaken.

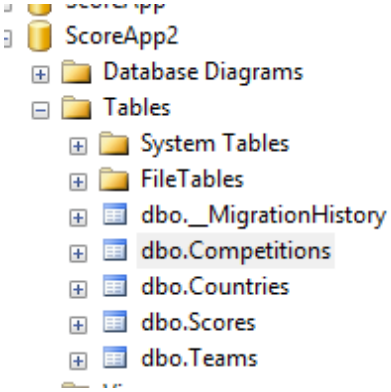


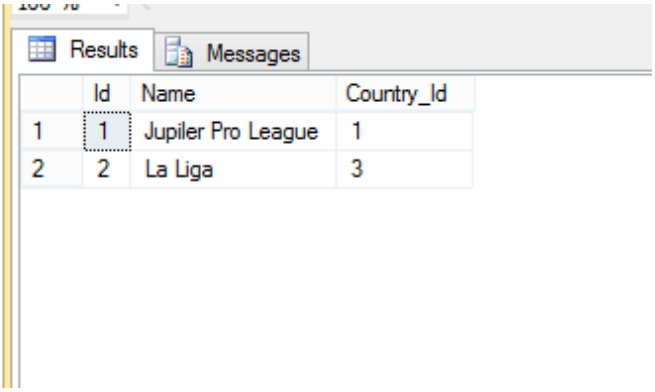


De laatste stap is het uitvoeren van een migration. Via het powershell commando “Update-database” zal EF migrations de laatst aangemaakte migratie uitvoeren.



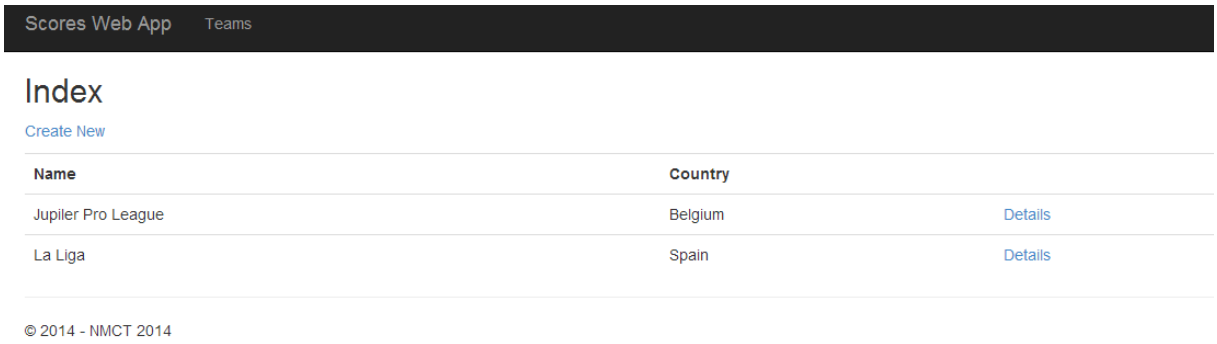
Controleer in SQL Server of de database werd aangemaakt met de juiste tabellen en of er data aanwezig is.



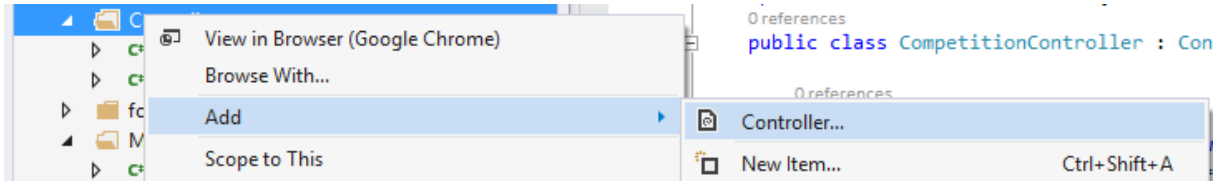


Overzicht scherm

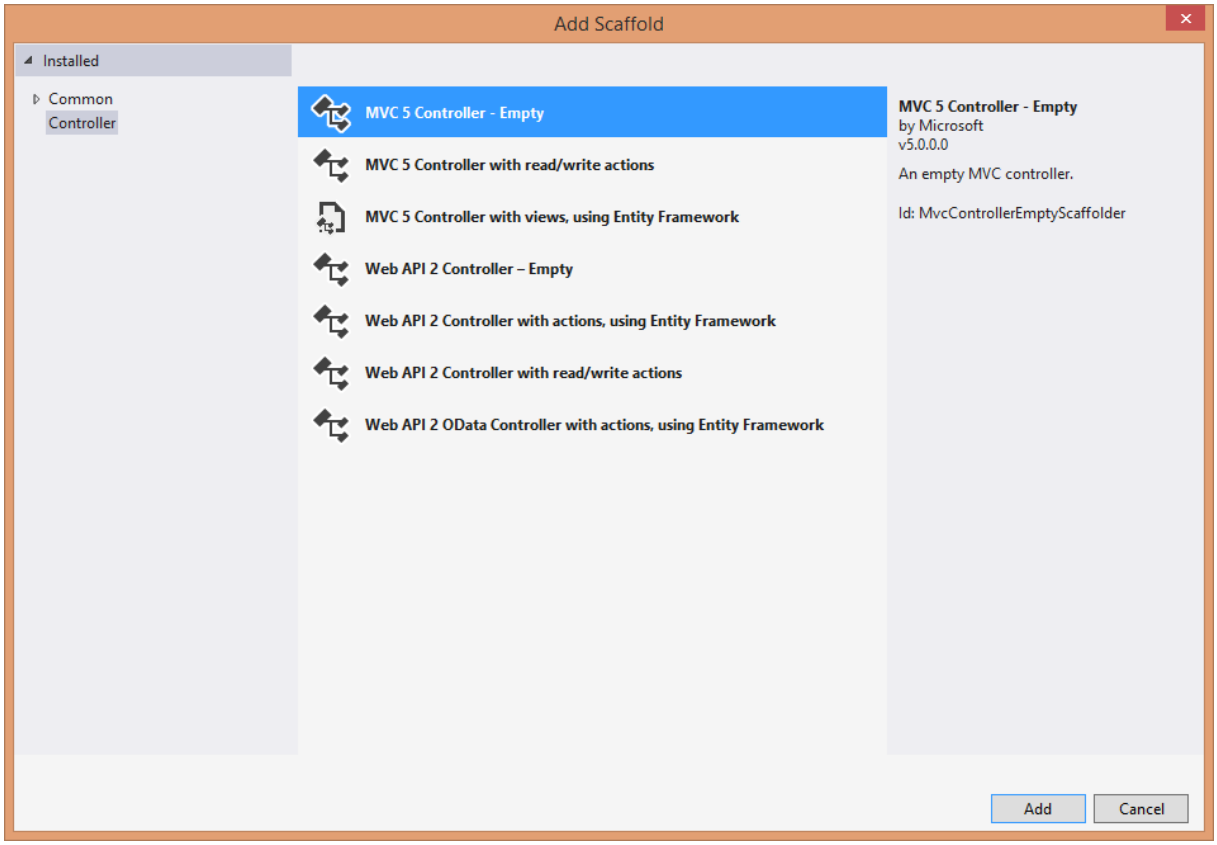
Het eerste scherm die we wensen te ontwikkelen ziet er als volgt uit:



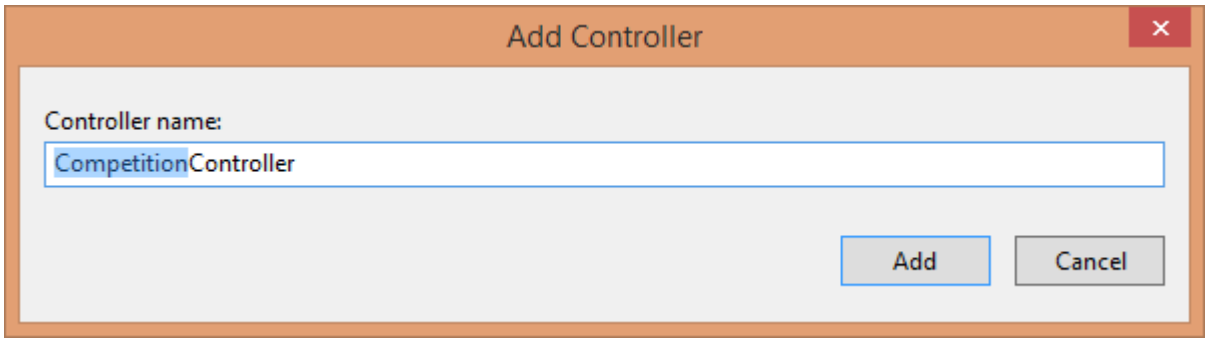
We starten met het toevoegen van een “CompetitionController”. Via rechtermuis op de map “Controllers” kiezen we “Add” “Controller”



Daarna verschijnt het onderstaande scherm waar we kiezen voor “MVC 5 Controller – Empty” en kiezen “Add”



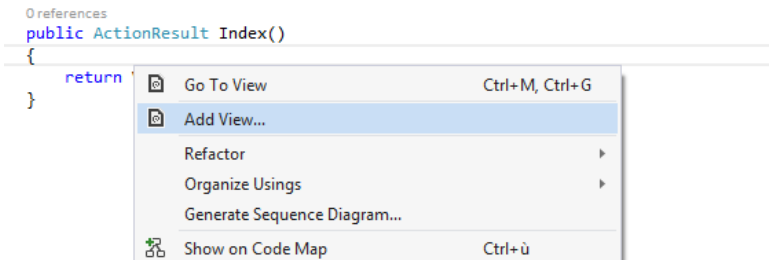
Daarna vullen we de naam van de controller in en drukken we op “Add”.



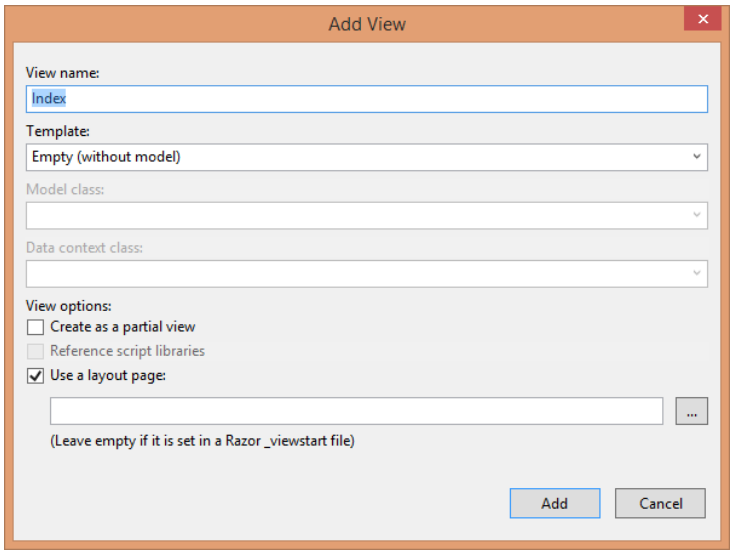
Voeg nu in de “CompetitionController” een actie (methode) Index() toe.

```
0 references
public class CompetitionController : Controller
{
    0 references
    public ActionResult Index()
    {
        return View();
    }
}
```

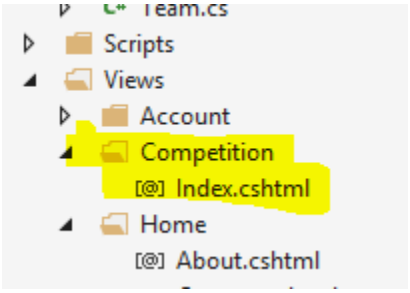
Als laatste voegen we een view toe via rechtermuisknop op de actie Index() en dan kies je voor “Add View”



Op het volgende scherm kiezen we gewoon “Add”.

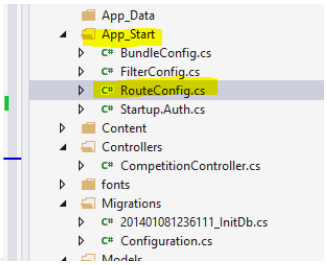


Alles alles goed verlopen is moet de map View er als volgt uitzien:

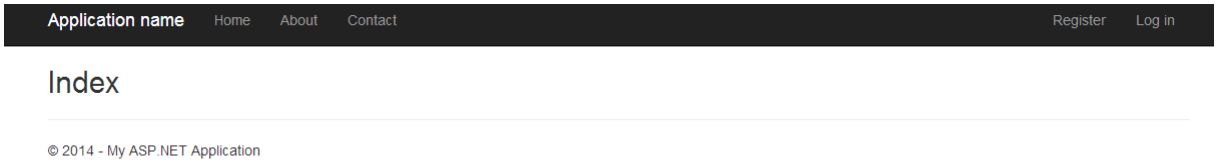


Als laatste passen we ook nog de default route aan zodat we bij het opstarten automatisch bij de CompetitionController terecht komen. Routes bepalen hoe je via de url bij een bepaalde contoller en action terecht komt.

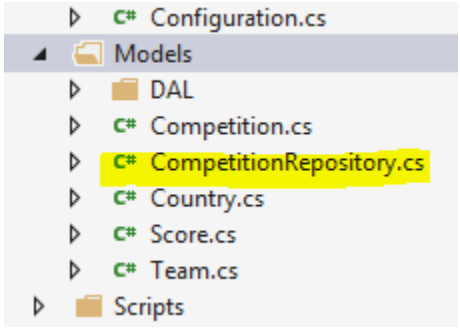
```
namespace NMCT.Scores {
    1 reference
    public class RouteConfig {
        1 reference
        public static void RegisterRoutes(RouteCollection routes) {
            routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

            routes.MapRoute(
                name: "Default",
                url: "{controller}/{action}/{id}",
                defaults: new { controller = "Competition", action = "Index", id = UrlParameter.Optional }
            );
        }
    }
}
```



Zorg er voor dat alles compileert. Het scherm moet er als volgt uit zien:



Op deze pagina gaan we alle competities weergeven. Vanuit de controller Index() methode moeten we alle competities ophalen en deze doorgeven aan de view “Index”. Om te communiceren met de database maken we eerst een “CompetitionRepository” aan. Dit concept kennen we nog vanuit semester 3 en heeft hier het zelfde doel, query’s uitvoeren op de database. Het verschil is echter dat we hier LINQ querys gaan uitvoeren. Voeg een nieuwe klasse toe onder models met naam “CompetitionRepository”.



Binnen deze klasse schrijven we onze methodes voor het ophalen, invoegen of wijzigen van data. De eerste methode die schrijven is GetCompetitions(). We maken eerst het de context aan met behulp van het using statement (weet u nog waarom ?). Binnen de context stellen we een LINQ query op waarmee we alle competities ophalen maar we willen ook het gerelateerde object Country mee ophalen. Hiervoor moeten we extension methode Include gebruiken. Vergeet niet bovenaan “using System.Data.Entity” toe te voegen, anders kan u niet type safe werken.

```
1 reference
public List<Competition> GetCompetitions() {
    using (ScoreContext context = new ScoreContext()) {

        var query = (from c in context.Competition.Include(c => c.Country) select c);
        return query.ToList<Competition>();

    }
}
```

Een tweede goede query die u kunt gebruiken ziet er als volgt uit:

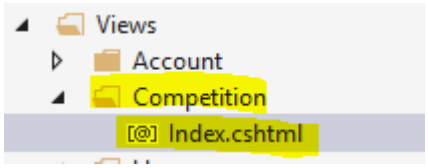
```
3 references
public List<Competition> GetCompetitions() {
    using (ScoreContext context = new ScoreContext()) {
        return context.Competition.Include(c => c.Country).ToList<Competition>();
    }
}
```

Beide zijn goed, u mag zelf de keuze maken.

De volgende stap is de repository gebruiken vanuit de controller. In de methode Index() voegen we onderstaande code toe. We maken een nieuw “CompetitionRepository” object aan. Daarna halen we via de methode GetCompetitions() alle competities op. Als laatste geven we deze door aan de view. Deze manier van werken kennen we reeds vanuit semester 3.

```
0 references
public ActionResult Index()
{
    CompetitionRepository repo = new CompetitionRepository();
    List<Competition> competitions = new List<Competition>();
    competitions = repo.GetCompetitions();
    return View(competitions);
}
```

Open nu de view “Index” in de map “Competition”.



Deze ziet er normaal als volgt uit:

```
@{
    ViewBag.Title = "Index";
}

<h2>Index</h2>
```

We moeten dus bovenaan opgeven welke object we wensen weer te geven in deze view. Dit doe we door het toevoegen van @model met daarna de volledige type naam “IEnumerable<NMCT.Scores.Models.Competition> . Nu weet de view engine wat hij mag verwachten van object.

```
@model IEnumerable<NMCT.Scores.Models.Competition>

@{
    ViewBag.Title = "Index";
}

<h2>Index</h2>
```

Als laatste voegen we nog wat html code toe zodat er een mooie lijst verschijnt met de competities en hun land. Hiervoor maken we gebruik van de reeds gekende foreach structuur en overlopen we ieder model binnen het @model. U mag dit opmaken naar uw smaak.

```
<table class="table">
    <tr>
        <th>
            @Html.DisplayNameFor(model => model.Name)
        </th>
        <th>
            Country
        </th>

        <th></th>
    </tr>

    @foreach (var item in Model) {
        <tr>
            <td>
                @Html.DisplayFor(modelItem => item.Name)
            </td>
            <td>
                @Html.DisplayFor(modelItem => item.Country.Name)
            </td>
            <td>
                @Html.ActionLink("Details", "Details", new { id = item.Id })
            </td>
        </tr>
    }

</table>
```

Het resultaat moet er als volgt uitzien:

Index

Create New

Name	Country	
Jupiler Pro League	Belgium	Details
La Liga	Spain	Details

Detail

Dit scherm activeren we door op de Details link te klikken in het overzicht scherm. Daarna moeten de details van de competitie verschijnen. Eerst staat de naam van de competitie met het land. Daaronder een lijst van alle uitslagen.

Voeg in de "CompetitionRepository" onderstaande methode toe. Deze methode zal in een query uitvoeren die de geselecteerde competitie zal ophalen maar daarnaast direct ook het land en de scores met hun team.

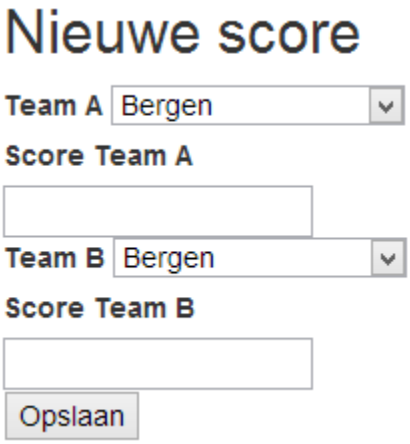
```
1 reference
public Competition GetCompetition(int id) {
    using (ScoreContext context = new ScoreContext()) {
        var query = (from c in context.Competition.Include(c => c.Country)
                                                  .Include(s => s.Scores.Select(t => t.TeamA))
                                                  .Include(s => s.Scores.Select(t => t.TeamB))
                     where c.Id == id
                     select c);

        return query.Single<Competition>();
    }
}
```

Voeg nu zelf volgende zaken toe:

- **Controller action Details**
- **View Detail**

Werk bovenstaande uit zodat je de details krijgt van de competitie (zie screenshot bovenaan).

We wensen ook een score toe te voegen. Voeg onderaan naast "Back To list" een link "Nieuwe Score". Via deze link komen we op een scherm die ons toelaat een nieuwe score toe te voegen voor een competitie. Dit scherm ziet er als volgt uit:



Dit is een iets complexer scherm dan deze die we reeds kennen. We moeten een team kunnen selecteren uit een dropdownlist. Daarnaast moeten we ook een score kunnen opgegeven voor elk team. Als u een complex scherm ziet met u altijd richting ViewModel denken. Een ViewModel is een speciale op maat gemaakte klasse voor onze view. Voeg een folder ViewModels toe aan het project en maak er een nieuwe klasse "AddScoreVM.cs" aan. De klasse zelf bevat een lijst van teams, het id

van het geselecteerde team A en geselecteerde team B , de score voor team A en de score voor team B. we houden ook het id van de competitie bij (weet u waarom ?).

```
0 references
public class AddScoreVM {

    0 references
    public List<Team> Teams { get; set; }
    0 references
    public int SelectedTeamA { get; set; }
    [Required]
    0 references
    public int ScoreA { get; set; }
    0 references
    public int SelectedTeamB { get; set; }
    [Required]
    0 references
    public int ScoreB { get; set; }
    0 references
    public int CompetitionId { get; set; }
}
```

We maken nu een actie AddScore met als parameter het id van de competitie. We duiden ook aan dat het om een HttpGet actie gaat. In deze methode maken we een TeamRepository. We vragen alle teams op van een bepaalde competitie. We vullen ons AddScoreVm object op en geven dit door aan de view AddScore

```
[HttpGet]
0 references
public ActionResult AddScore(int competitionId) {

    TeamRepository repoTeam = new TeamRepository();
    AddScoreVM vm = new AddScoreVM();
    vm.Teams = repoTeam.GetTeams(competitionId);
    vm.CompetitionId = competitionId;
    return View(vm);
}
```

Voeg nu zelf volgende zaken toe:

- **TeamRepository klasse**
- **Een methode GetTeams die voor een bepaalde competitie de teams ophaalt**
- **View AddScore (zie screenshot hierboven)**
    - **De dropdownlist voor de landkeuze mag je doen ZONDER helper. Je mag dus manueel de html <select/> tag gebruiken.**

Als bovenstaande zaken goed zijn toegevoegd dan moet je het scherm zien waar je een score kan toevoegen.

Na het invullen van de gegevens moeten we de data doorsturen naar de server (POST). In de CompetitionController voegen we onderstaande methode toe. AddScore met als parameter ons viewmodel. We maken een nieuw score object aan en vullen dit op met de gegevens uit ons viewmodel. We halen ook de twee teams op via de methode GetTeam() die u nog zelf moet voorzien. Als laatste is er de methode AddScore die een score zal toevoegen aan de database.

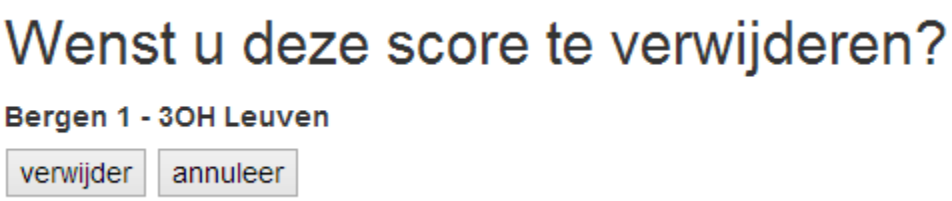
```
[HttpPost]
0 references
public ActionResult AddScore(AddScoreVM vm) {

    TeamRepository repoTeam = new TeamRepository();
    Score score = new Score();
    score.ScoreA = vm.ScoreA;
    score.ScoreB = vm.ScoreB;
    score.TeamA = repoTeam.GetTeam(vm.SelectedTeamA);
    score.TeamB = repoTeam.GetTeam(vm.SelectedTeamB);
    score.CompetitionId = vm.CompetitionId;

    CompetitionRepository repoComp = new CompetitionRepository();
    repoComp.AddScore(score);
    return RedirectToAction("Index");
}
```

Voeg nu zelf volgende zaken toe:

- **Methode GetTeam aan de TeamRepository met als parameter het Id van het team**
- **Methode AddScore aan de CompetitionRepository met als parameter een Score object**

Als laatste moet het mogelijk zijn om een score te verwijderen. Naast de score voeg je een link toe om een score te verwijderen.

Als je op verwijder klikt krijg je eerst de vraag of je dit echt wel wenst te verwijderen.



Indien we nu expliciet “verwijder” kiezen dan mag de score uit de database verwijderd worden.

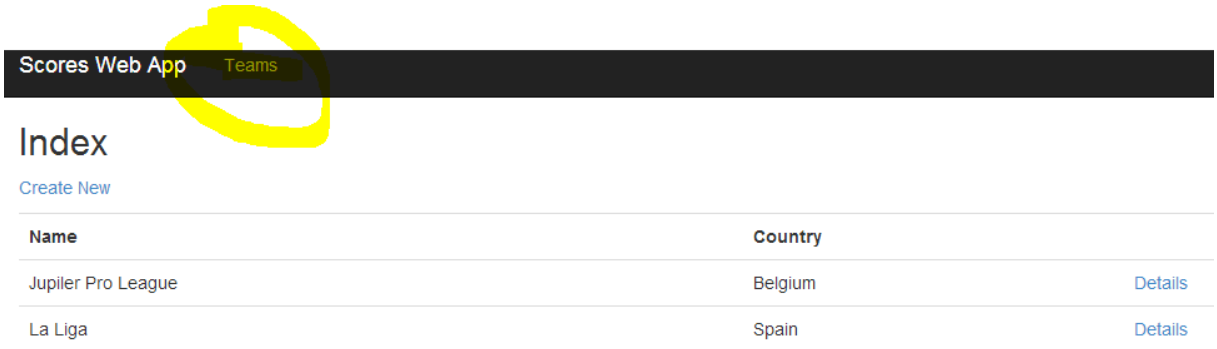
Voeg nu zelf volgende zaken toe:

- **Actie die vraag stelt om score te verwijderen**
  - **Binnen deze actie zal je naar de database moeten gaan en de score opvragen. Voorzie een methode GetScore met een score id als parameter om deze op te halen**
  - **Voeg een view toe die bovenstaande vraag toont**

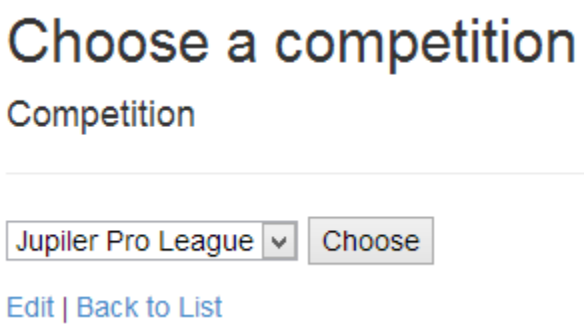
- **Actie die effectief de score zal verwijderen**
  - **Deze actie moet je uitvoeren als je op verwijder klikt**
  - **Hier zal je een methode RemoveScore aanroepen uit de repository competition. U moet zelf deze methode nog voorzien.**

Teams

We voorzien één menu bovenaan in het overzicht scherm. Pas de view aan zodat het startscherm er als volgt uit ziet:



Als we Teams klikken dan krijgen we het onderstaande scherm te zien. Een dropdownlist met de competities in de database en een knop op een competitie te selecteren.



Wanneer je een competitie kiest krijg je onderstaande scherm:

Werk dit volledig zelfstandig uit. Maak hiervoor een aparte TeamController aan. Gebruik indien nodig een ViewModel.