

ROC van Twente

Opleiding Software Developer

Marcel Roesink

November 2024

Inhoud

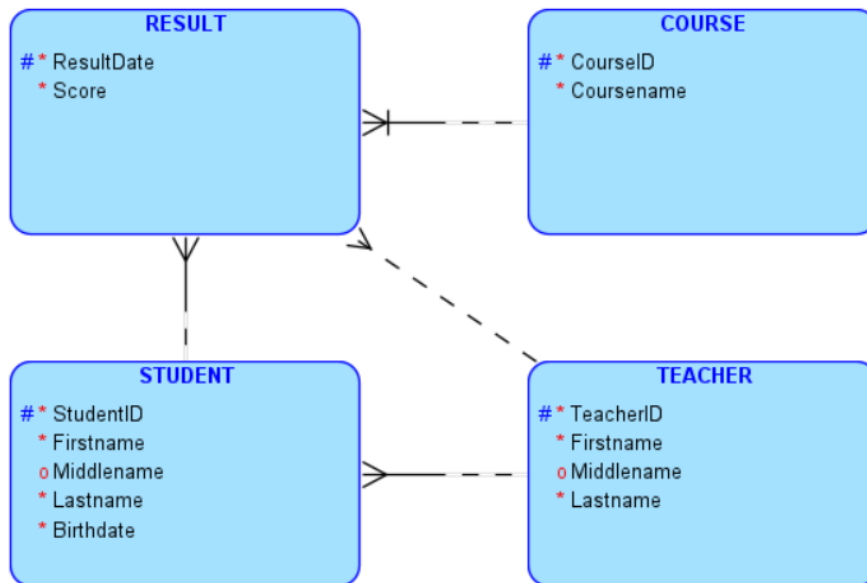
Casus.....	3
Models maken in ASP.NET/MVC met C#.....	4
Course.cs	6
Teacher.cs.....	6
Student.cs.....	7
Result.cs.....	8
Code first versus database first	9
Code First.....	9
Database First.....	9
Code First database realisatie	9
Controllers toevoegen.....	12
StudentController.....	12
Opslaan van connectionstring buiten appsettings.....	17
Meer info.....	19
ASP.NET Core Crash Course - C# App in One Hour.....	19
Getting Started with Entity Framework Core [1 of 5] Entity Framework Core for Beginners.....	19
C#.Net Tutorial For Beginners, Lesson 15: Introduction to LINQ.....	19
• LINQ to Objects: Voor het bevragen van in-memory data (bijv. lijsten).	19
• LINQ to SQL: Om SQL-databases te bevragen.	19
• LINQ to XML: Voor het werken met XML-documenten.	19
Hierdoor kun je met één uniforme syntax allerlei gegevensbronnen benaderen.....	19
Source code	19

Van ERD naar applicatie naar database

Casus

Tijdens de database lessen hebben we een (vereenvoudigd) data model gemaakt voor een applicatie waarin cijfers van toetsen geregistreerd kunnen worden.

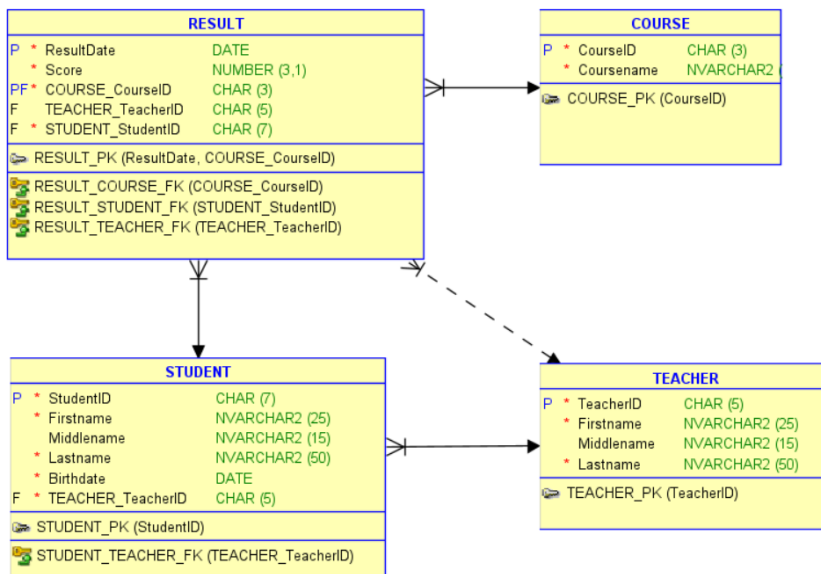
Het logische ontwerp ziet er zo uit:



Toelichting:

- Een student KAN één of meerdere resultaten behalen.
- Een docent KAN één of meerdere resultaten geven aan studenten.
- Voor een vak KUNNEN één of meerdere resultaten worden toegekend aan studenten.
- Een docent KAN SLBer zijn van één of meerdere studenten.
- De primary key van RESULT bestaat uit 3 attributen:
 - ResultDate
 - StudentID
 - CourseID

Het bijbehorende relationele ontwerp ziet er zo uit:



Models maken in ASP.NET/MVC met C#

Maak eerst een nieuw ASP.NET/MVC project aan in Visual Studio:

Configure your new project

ASP.NET Core Web App (Model-View-Controller) C# Linux macOS Windows Cloud Service Web

Project name
I2SD_DemoEF

Location
C:\Users\Gebruiker\source\repos

Solution name
I2SD_DemoEF

☐ Place solution and project in the same directory

Project will be created in "C:\Users\Gebruiker\source\repos\I2SD_DemoEF\I2SD_DemoEF\"


Back Next


Natuurlijk mag je zelf andere namen kiezen en/of het project in een andere map opslaan!



Kies het gewenste .NET Framework (.NET 8.0) en Authentication type (Individual Accounts):


Additional information


ASP.NET Core Web App (Model-View-Controller) C# Linux macOS Windows Cloud Service Web



Framework  .NET 8.0 (Long Term Support)

Authentication type  Individual Accounts

☒ Configure for HTTPS 
☐ Enable container support 

Container OS  Linux

Container build type  Dockerfile

☐ Do not use top-level statements 
☐ Enlist in .NET Aspire orchestration 


Back Create

Het authentication type is op zich niet nodig, maar door te kiezen voor **Individual Accounts** worden al veel configuratieinstellingen uitgevoerd die we later nodig hebben.






Update nu, indien nodig, eerst alle package:

NuGet: I2SD.DemoEF - x USD.DemoEF - Overview



Browse Installed Updates

Search (Ctrl+L)  ☐ Include prerelease

☒ Select all packages Update

	Microsoft.AspNetCore.Diagnostics.EntityFrameworkCore by Microsoft ASP.NET Core middleware for Entity Framework Core error pages. Use this middleware to detect and diagnose errors with Entity Framework Core migrations.	8.0.8 8.0.10
	Microsoft.AspNetCore.Identity.EntityFrameworkCore by Microsoft ASP.NET Core Identity provider that uses Entity Framework Core.	8.0.8 8.0.10
	Microsoft.AspNetCore.Identity.UI by Microsoft ASP.NET Core Identity UI is the default Razor Pages built-in UI for the ASP.NET Core Identity framework.	8.0.8 8.0.10
	Microsoft.EntityFrameworkCore.SqlServer by Microsoft Microsoft SQL Server database provider for Entity Framework Core.	8.0.8 8.0.10
	Microsoft.EntityFrameworkCore.Tools by Microsoft Entity Framework Core Tools for the NuGet Package Manager Console in Visual Studio.	8.0.8 8.0.10

Voeg vervolgens de volgende package nog toe aan het project:

 **Microsoft.EntityFrameworkCore.Design**  by [aspnet](#), [dotnetframework](#), [EntityFramework](#), [Microsoft](#), **538M** downloads
 Shared design-time components for Entity Framework Core tools.

Voeg aan de map *Models* de volgende models toe:

TIP: Gebruik de volgende prompt om uitleg van ChatGPT te krijgen bij deze code:

Kun je onderstaande code uitleggen aan iemand die weinig kennis heeft van C#

Course.cs

```
public class Course
{
    private string _courseID = string.Empty;

    [Key, RegularExpression(@"[A-Z]{3}")]
    [MaxLength(3)]
    0 references
    public string CourseID
    {
        get => _courseID;
        set => _courseID = value.ToUpper();
    }
    [Required, MaxLength(75)]
    0 references
    public string CourseName { get; set; } = string.Empty;
}
```

TIP: Gebruik de volgende prompt om uitleg van ChatGPT te krijgen bij deze code:

Kun je onderstaande code uitleggen aan iemand die weinig kennis heeft van C#

Teacher.cs

```
public class Teacher
{
    [Key, RegularExpression(@"[A-Z]{3}\d{2}")]
    [Column(TypeName = "char(5)")]
    0 references
    public string TeacherID { get; set; } = string.Empty;
    [Required, MaxLength(25)]
    0 references
    public string FirstName { get; set; } = string.Empty;
    [MaxLength(15)]
    0 references
    public string? MiddleName { get; set; } = null;
    [Required, MaxLength(50)]
    0 references
    public string LastName { get; set; } = string.Empty;
}
```

TIP: Gebruik de volgende prompt om uitleg van ChatGPT te krijgen bij deze code:
Kun je onderstaande code uitleggen aan iemand die enigszins kennis heeft van C#

Student.cs

```
public class Student
{
    [Key, RegularExpression(@"\d{7}")]
    [Column(TypeName = "char(7)")]
    22 references
    public string StudentID { get; set; } = string.Empty;
    [Required, MaxLength(25)]
    14 references
    public string FirstName { get; set; } = string.Empty;
    [MaxLength(15)]
    14 references
    public string? MiddleName { get; set; } = null;
    [Required, MaxLength(50)]
    14 references
    public string LastName { get; set; } = string.Empty;
    [DataType(DataType.Date)]
    12 references
    public DateTime Birthdate { get; set; } = DateTime.MinValue;
    0 references
    public List<Result> Results { get; set; } = new List<Result>();
    3 references
    public string? SlbID { get; set; } = null;
    3 references
    public Teacher? Slb { get; set; } = null;
    8 references
    public override string ToString()
    {
        var fullName = MiddleName == null ? $"{LastName}, {FirstName}" :
            $"{LastName}, {FirstName} {MiddleName}";
        fullName += $" ({StudentID})";
        return fullName;
    }
}
```

TIP: Gebruik de volgende prompt om uitleg van ChatGPT te krijgen bij deze code:

Kun je onderstaande code uitleggen aan iemand die weinig kennis heeft van C#

Result.cs

```
[PrimaryKey(nameof(StudentID), nameof(CourseID), nameof(ResultDate))]
24 references
public class Result
{
    12 references
    public string StudentID { get; set; } = string.Empty;
    9 references
    public Student? Student { get; set; } = null!;
    7 references
    public string CourseID { get; set; } = string.Empty;
    9 references
    public Course? Course { get; set; } = null!;
    7 references
    public string? TeacherID { get; set; } = string.Empty;
    9 references
    public Teacher? Teacher { get; set; } = null!;
    12 references
    public double Score { get; set; }
    [DataType(DataType.Date)]
    6 references
    public DateTime ResultDate { get; set; } = DateTime.MinValue;
}
```

Toelichting:

- Course, Teacher en Student zijn zogenaamde navigation properties, die mogelijk maken om van een resultaat naar de gekoppelde objecten te navigeren.
- CourseID, TeacherID en StudentID zijn de foreign keys die verwijzen naar de primary keys van resp. Course, Teacher en Student.
- De primary key bestaat uit 3 attributen (StudentID, CourseID en ExamDate)
- Met de annotatie DataType(DataType.Date) geven we aan dat we alleen een datum willen zonder tijd.

We kunnen nu aan de models Student, Teacher en Course een navigation property toevoegen om naar de resultaten te navigeren:

```
public List<Result> Results { get; set; } = new List<Result>();
```

TIP: Gebruik de volgende prompt om uitleg van ChatGPT te krijgen bij deze code:

Wat is het nut van deze code, als die bijvoorbeeld wordt toegevoegd aan de class Course

Code first versus database first

Veel applicatie maken gebruik van een database en moeten daarom gegevens in die database kunnen opvragen en bewerken. Dat ben je al regelmatig tegengekomen middels de afkorting CRUD:

Create:

Er moeten nieuwe rijen aan een table toegevoegd kunnen worden

Read:

Er moeten rijen opgevraagd kunnen worden. Soms is dat een lijst met meerdere rijen en soms is dat één rij.

Update:

Een bestaande rij moet aangepast kunnen worden

Delete:

Een bestaande rij moet verwijderd kunnen worden. Daarbij moet wel altijd de vraag worden gesteld of verwijderen van een rij wenselijk is. Er kan dan namelijk historische data verloren gaan en je weet niet of dit wenselijk is. Goede communicatie met de opdrachtgever is daarvoor noodzakelijk.

Om een database te realiseren kun je doorgaans 2 wegen bewandelen:

Code First

Je maakt in C# models aan en in een database context geef je aan voor welke models een table in de database aangemaakt moet worden. Vervolgens gebruik je een tool als **Entity Framework** om eerst een migratie aan te maken (add-migration “naam”) en vervolgens update je de database (update-database). Natuurlijk heb je een connectionstring nodig, waarin wordt aangegeven welke database gebruikt moet worden op welke server en met behulp van welke user id.

Database First

Je maakt in SQL Server Management Studio de database aan inclusief alle gewenste tabellen en relaties en gebruikt een tool als **Entity Framework** om op basis van de database C# classes (models) te laten genereren. We noemen dit ook wel **Reverse Engineering**.

In deze tutorial gaan we gebruik maken van Code First, maar als je meer wilt weten over database first dan geeft de volgende site voldoende uitleg:

<https://learn.microsoft.com/en-us/ef/core/managing-schemas/scaffolding/?tabs=dotnet-core-cli>

Of je bekijkt deze video van Israel Quiroz:

<https://youtu.be/dB2V0hUJIR0?si=iN-la1DBcQyj8zO5>

Code First database realisatie

We hebben hiervoor al de benodigde models aangemaakt. Wat we nu nog nodig hebben is een database context, waarin we aangeven welke models we nodig hebben in onze database. Doordat we eerder hebben aangegeven dat we gebruik maken van *Individual Accounts* is er al een ApplicationDbContext class aangemaakt, die we kunnen vinden in de map Data. In het bestand *appsettings.json* treffen we de connectionstring voor onze database aan. Aangezien we gebruik

willen maken van de SQL Server installatie op onze eigen laptop passen we de connectionstring als volgt aan:

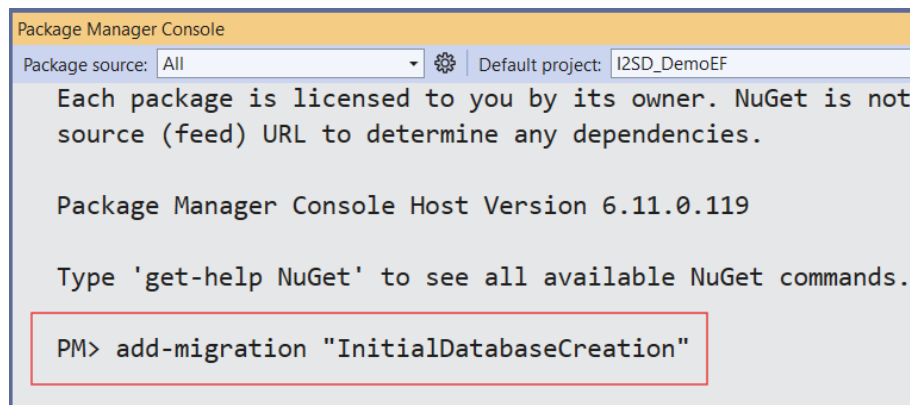
```
{
  "ConnectionStrings": {
    "DefaultConnection": "Server=.;Database=I2SD_DemoEF_DB;Trusted_Connection=True;MultipleActiveResultSets=true;Encrypt=False"
  },
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*"
}
```

LET OP: door een punt op te geven voor de server geven we aan dat het locale system gebruikt moet worden. Dit is echter wel afhankelijk van de manier waarop jij SQL Server hebt geïnstalleerd.

Voor ieder model maken we in *ApplicationDbContext.cs* een DbSet aan.

```
public class ApplicationDbContext : IdentityDbContext
{
    0 references
    public DbSet<Student> Students { get; set; } = null!;
    0 references
    public DbSet<Teacher> Teachers { get; set; } = null!;
    0 references
    public DbSet<Course> Courses { get; set; } = null!;
    0 references
    public DbSet<Result> Results { get; set; } = null!;
    0 references
    public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)
        : base(options)
    {
    }
}
```

Maak nu een eerste migration aan in de package manager console:



En vervolgens maken we de database aan met:

```

Package Manager Console
Package source: All [gear icon] Default project: I2SD_DemoEF

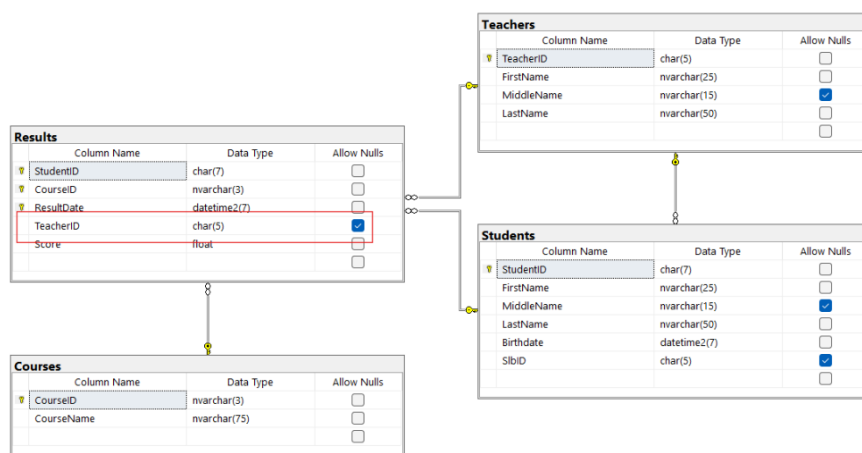
Package Manager Console Host Version 6.11.0.119

Type 'get-help NuGet' to see all available NuGet commands.

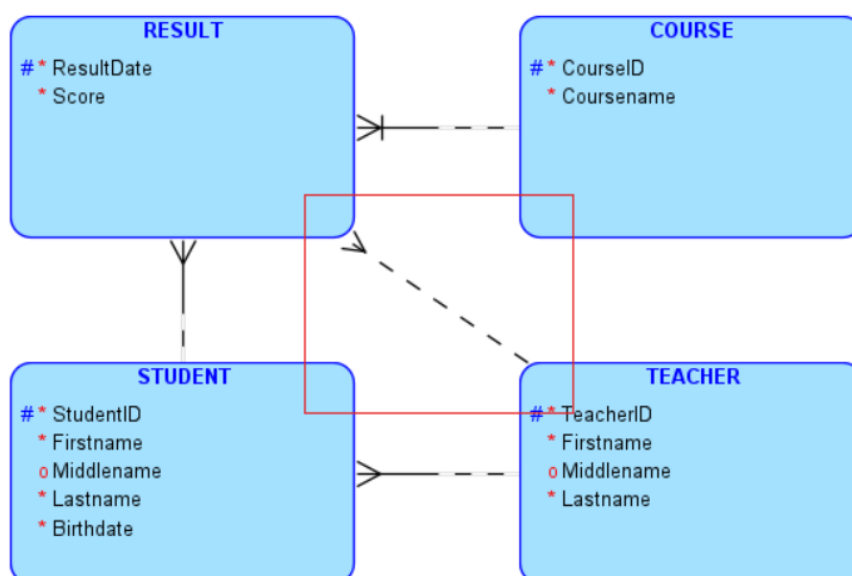
PM> add-migration "InitialDatabaseCreation"
Build started...
Build succeeded.
To undo this action, use Remove-Migration.
PM> update-database

```

Maak vervolgens in SQL Server Management Studio een database diagram aan om te controleren of alle tabellen en relaties correct zijn aangemaakt:



Merk op dat bij de *Results* tabel TeacherID geen verplicht veld is. Dit hebben we in ons eerste ontwerp in de Oracle Data Modeler ook aangegeven:

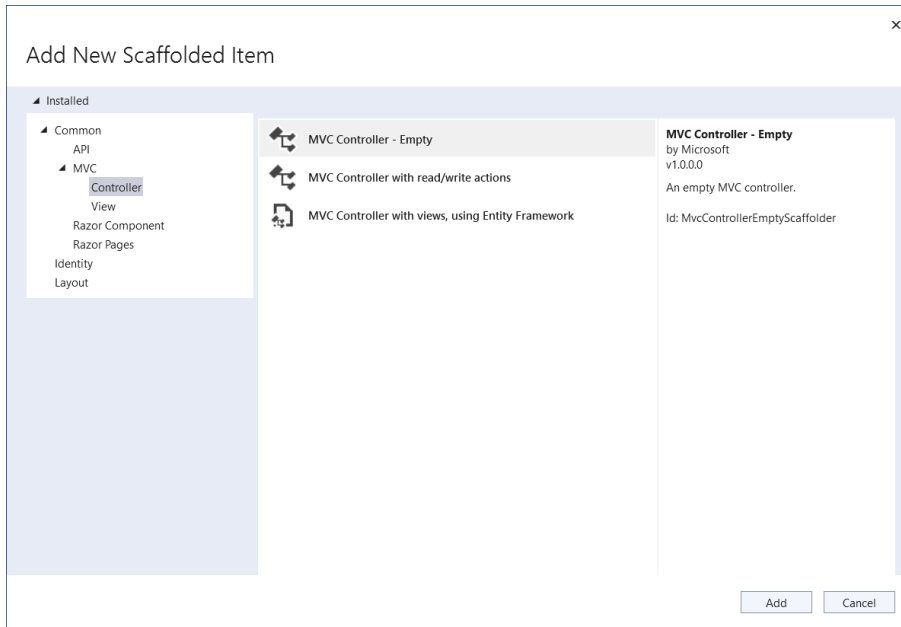


Controllers toevoegen

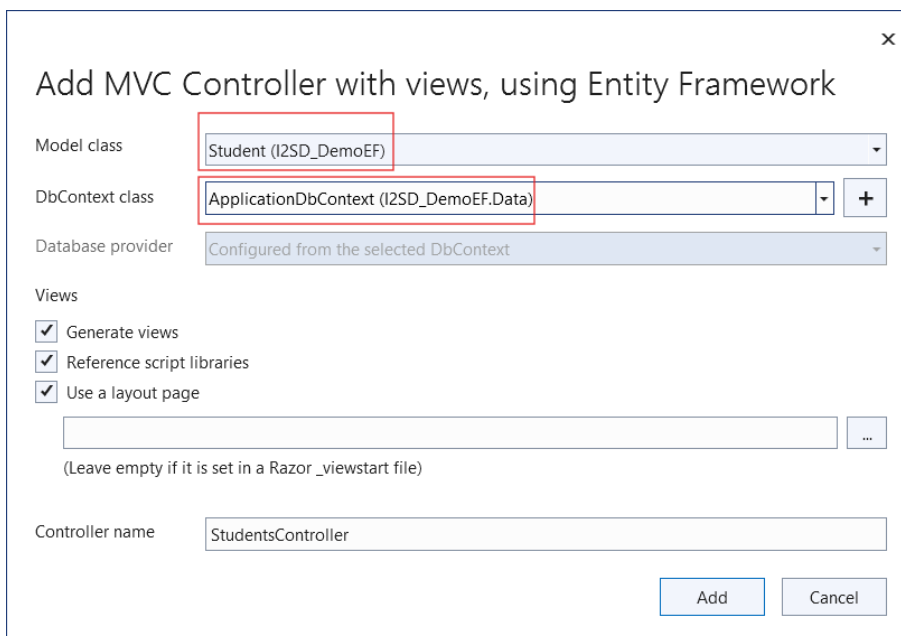
We gaan nu met behulp van wizards controllers toevoegen, zodat we CRUD operaties uit kunnen voeren op de tabellen van onze database

StudentController

Klik rechts op de map *Controllers* en kies dan *Add -> Controller*:



Kies **MVC Controller with views, using Entity Framework** en klik op **Add**.



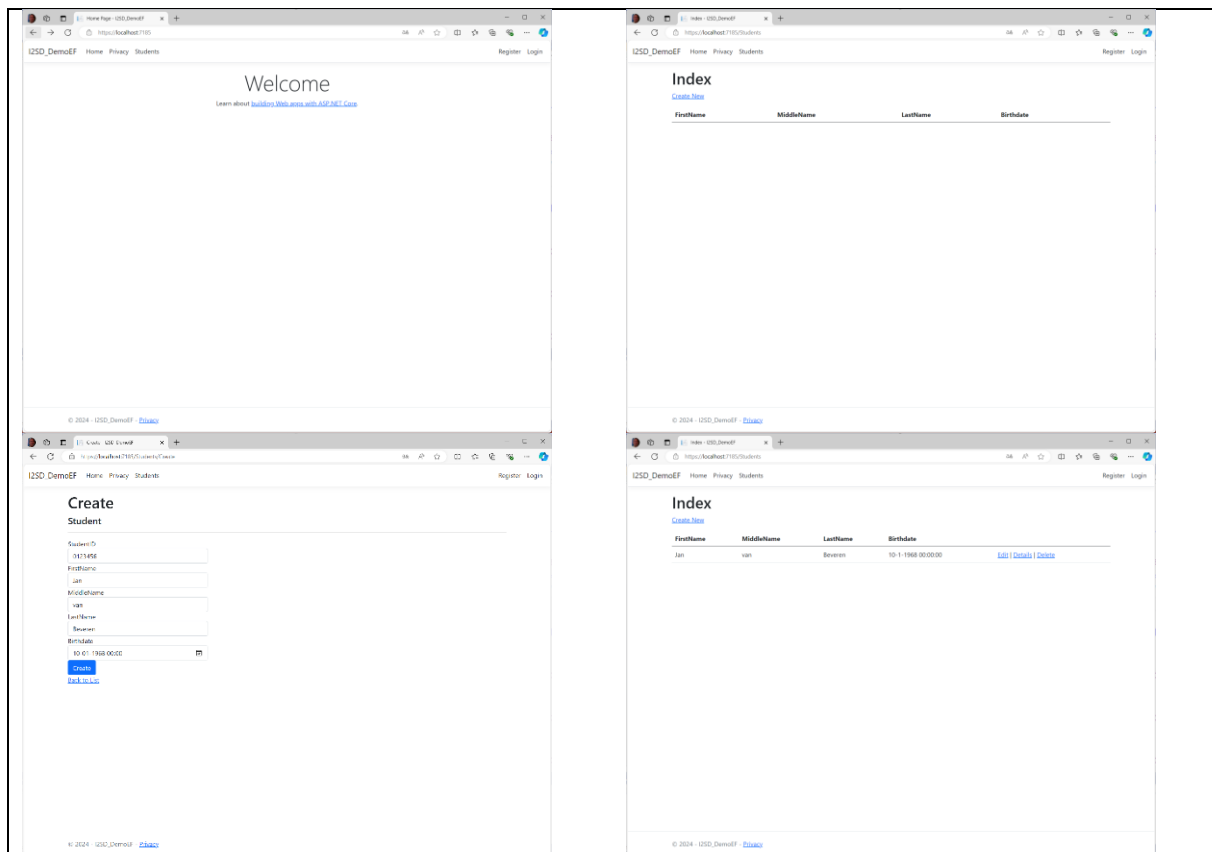
Selecteer het gewenste model en de DbContext en klik op **Add**.

Na ongeveer een halve minuut is de controller inclusief alle benodigde views aangemaakt. Als er een foutmelding komt heb je waarschijnlijk niet alle noodzakelijk packages geïnstalleerd.

Voeg in *_Layout.cshtml* een menuoptie toe voor het beheren van de studenten:

```
class= navbar-collapse collapse d-sm-inline-flex justify-content-between >
<ul class="navbar-nav flex-grow-1">
  <li class="nav-item">
    <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="Index">Home</a>
  </li>
  <li class="nav-item">
    <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="Privacy">Privacy</a>
  </li>
  <li class="nav-item">
    <a class="nav-link text-dark" asp-area="" asp-controller="Students" asp-action="Index">Students</a>
  </li>
</ul>
<partial name="LoginPartial" />
```

Test nu je applicatie en kijk of je een student kunt toevoegen.



Op dezelfde manier kun je ook controllers inclusief bijbehorende views maken voor Teacher en Course. En om het helemaal af te maken voegen we ook nog een controller toe voor Result. Entity Framework is slim genoeg om te herkennen dat Result een-op-veel relaties heeft met Teacher, Course en Student en zal automatisch dropdowns maken. Echter, in de dropdowns wordt de primary key getoond en dat is niet wat we willen. Voor het vak moet de vaknaam worden getoond en voor student en docent de volledige naam (b.v. als *Beveren, Jan van* of *Roesink, Marcel*). En dan ook nog graag in alfabetische volgorde in de dropdowns. De vaknaam kunnen we rechtstreeks uit de tabel halen, maar de naam van de studenten en docenten niet. Daarom gaan we in deze models de ToString() methode overriden.

In *Teacher.cs* voegen we toe:

```
public override string ToString()
{
    var fullName = MiddleName == null ? $"{LastName}, {FirstName}" :
        $"{LastName}, {FirstName} {MiddleName}";
    fullName += $" ({TeacherID})";
    return fullName;
}
```

En in *Student.cs*:

```
public override string ToString()
{
    var fullName = MiddleName == null ? $"{LastName}, {FirstName}" :
        $"{LastName}, {FirstName} {MiddleName}";
    fullName += $" ({StudentID})";
    return fullName;
}
```

Vervolgens passen we de code in de Result controller aan:

Create action (get):

```
public IActionResult Create()
{
    ViewData["CourseID"] =
        new SelectList(_context.Courses.OrderBy(c => c.CourseName), "CourseID", "CourseName");
    ViewData["StudentID"] =
        new SelectList(
            _context.Students.ToList().OrderBy(s => s.ToString())
                .Select(s => new { s.StudentID, Name = s.ToString() }),
            "StudentID",
            "Name"
        );
    ViewData["TeacherID"] =
        new SelectList(
            _context.Teachers.ToList().OrderBy(t => t.ToString())
                .Select(s => new { s.TeacherID, Name = s.ToString() }),
            "TeacherID",
            "Name"
        );
    return View();
}
```

Create action (post):

```
// To protect from overposting attacks, enable the specific properties you want to bind to.
// For more details, see http://go.microsoft.com/fwlink/?LinkId=317598.
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Create([Bind("StudentID,CourseID,TeacherID,Score,ResultDate")] Result result)
{
    if (ModelState.IsValid)
    {
        _context.Add(result);
        await _context.SaveChangesAsync();
        return RedirectToAction(nameof(Index));
    }
    ViewData["CourseID"] =
        new SelectList(_context.Courses.OrderBy(c => c.CourseName), "CourseID", "CourseName");
    ViewData["StudentID"] =
        new SelectList(
            _context.Students.ToList().OrderBy(s => s.ToString())
                .Select(s => new { s.StudentID, Name = s.ToString() }),
            "StudentID",
            "Name"
        );
    ViewData["TeacherID"] =
        new SelectList(
            _context.Teachers.ToList().OrderBy(t => t.ToString())
                .Select(s => new { s.TeacherID, Name = s.ToString() }),
            "TeacherID",
            "Name"
        );
    return View(result);
}
```

Edit action (get):

```
public async Task<IActionResult> Edit(string id)
{
    if (id == null)
    {
        return NotFound();
    }

    var result = await _context.Results.FindAsync(id);
    if (result == null)
    {
        return NotFound();
    }
    ViewData["CourseID"] = new SelectList(_context.Courses.OrderBy(c => c.CourseName), "CourseID", "CourseName");
    ViewData["StudentID"] = new SelectList(
        _context.Students.ToList().OrderBy(s => s.ToString())
        .Select(s => new { s.StudentID, Name = s.ToString() }),
        "StudentID",
        "Name"
    );
    ViewData["TeacherID"] = new SelectList(
        _context.Teachers.ToList().OrderBy(t => t.ToString())
        .Select(s => new { s.TeacherID, Name = s.ToString() }),
        "TeacherID",
        "Name"
    );
    return View(result);
}
```

Edit action (post):

```
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Edit(string id, [Bind("StudentID,CourseID,TeacherID,Score,ResultDate")] Result result)
{
    if (id != result.StudentID)
    {
        return NotFound();
    }

    if (ModelState.IsValid)
    {
        try
        {
            _context.Update(result);
            await _context.SaveChangesAsync();
        }
        catch (DbUpdateConcurrencyException)
        {
            if (!ResultExists(result.StudentID))
            {
                return NotFound();
            }
            else
            {
                throw;
            }
        }
        return RedirectToAction(nameof(Index));
    }
    ViewData["CourseID"] =
        new SelectList(_context.Courses.OrderBy(c => c.CourseName), "CourseID", "CourseName");
    ViewData["StudentID"] =
        new SelectList(
            _context.Students.ToList().OrderBy(s => s.ToString())
            .Select(s => new { s.StudentID, Name = s.ToString() }),
            "StudentID",
            "Name"
        );
    ViewData["TeacherID"] =
        new SelectList(
            _context.Teachers.ToList().OrderBy(t => t.ToString())
            .Select(s => new { s.TeacherID, Name = s.ToString() }),
            "TeacherID",
            "Name"
        );
    return View(result);
}
```


Rest nog één vervelend probleem dat we ook in de Bookstore tutorial al tegen zijn gekomen. Als de taalinstellingen van de gebruiker staan ingesteld op Nederlands gaat het invoeren van een cijfer voor een toets fout als een cijfer met decimalen wordt ingevoerd.

Je kunt dit oplossen door onderaan de view *Create.cshtml* en *Edit.cshtml* de volgende javascript/jQuery code toe te voegen (copy-paste):

```
@section Scripts {
    @{await Html.RenderPartialAsync("_ValidationScriptsPartial");}
    <script>
        $.validator.methods.number = function (value, element) {
            return this.optional(element) ||
                /^-?(?:\d+|\d{1,3}(?:[\s\.,]\d{3})+)(?:[\.\,]\d+)?$/\.test(value);
        }
    </script>
}
```

Opslaan van connectionstring buiten appsettings

Het opslaan van een connectionstring met gevoelige gegevens (zoals wachtwoorden) vereist zorgvuldige beveiligingsmaatregelen om ongeautoriseerde toegang te voorkomen. Een methode om de connectionstring veiliger op te slaan in een .NET-project is het gebruik van User Secrets. :

1. Gebruik van User Secrets (in Development)

Voor lokale ontwikkeling kun je User Secrets gebruiken om gevoelige informatie, zoals connectionstrings, buiten je broncode te houden.

- **Stappen:**

1. Open de map *Connected Services* en klik rechts op *Secrets.json* en kies voor *Manage User Secrets*.
2. Verplaats je connectionstring van *appsettings.json* naar *secrets.json*.

Deze gegevens worden dan veilig opgeslagen in een JSON-bestand in een gebruikersmap buiten de projectmap en niet gedeeld via versiebeheer. Dit kun je doen met alle gevoelige informatie, zoals bijvoorbeeld ook toegangsgegevens voor een mailserver en API-keys.

2. Azure Key Vault of Secret Managers (voor Cloud Deployments)

In productie kun je een cloudgebaseerde geheimenbeheerder gebruiken, zoals **Azure Key Vault**, **AWS Secrets Manager**, of **HashiCorp Vault**. Deze zijn specifiek ontworpen voor het veilig beheren van geheime gegevens.

- **Stappen in Azure Key Vault:**

1. Maak een Azure Key Vault aan.
2. Voeg je connectionstring toe als een geheim in de Key Vault.
3. Geef je applicatie of je gebruikersaccount toegang tot de Key Vault.
4. Configureer je applicatie om verbinding te maken met de Key Vault en de connectionstring op te halen via [Microsoft.Extensions.Configuration.AzureKeyVault](https://docs.microsoft.com/en-us/azure/key-vault/configuration).

We werken deze laatste stap niet verder uit in deze tutorial, maar het is van groot belang dat je dit goed gaat toepassen als je ooit een ASP.NET applicatie online gaat zetten.

Meer info

ASP.NET Core Crash Course - C# App in One Hour

Voor wie alles nog eens op een rijtje wil in een beknopte video:

<https://youtu.be/BfEjDD8mWYg?si=uCZEmo3SCTzheLUR>

Weliswaar voor een oudere versie van ASP.NET, maar evengoed wel bruikbaar. In oudere versie wordt gebruik gemaakt van 2 opstartbestanden voor de website:

- Program.cs
- Startup.cs

In latere versies is deze gecombineerd in één file:

- Program.cs

Getting Started with Entity Framework Core [1 of 5] | Entity Framework Core for Beginners

Een mooie introductie in Entity Framework

<https://youtu.be/SryQxUeChMc?si=JkzsJlMA17Xjkvpc>

C#.Net Tutorial For Beginners, Lesson 15: Introduction to LINQ

LINQ (Language-Integrated Query) is een functie in .NET (zoals in C# en VB.NET) waarmee je op een eenvoudige en uniforme manier gegevens kunt opvragen en manipuleren. Het integreert query-functionaliteit direct in de programmeertaal, zodat je queries kunt schrijven die werken met verschillende soorten gegevensbronnen, zoals lijsten, databases en XML. LINQ maakt de code leesbaarder door een declaratieve stijl te gebruiken waarin je beschrijft wat je wilt bereiken, niet hoe. LINQ biedt ook ondersteuning voor IntelliSense en compile-time checking, wat fouten tijdens het programmeren helpt voorkomen.

Er zijn verschillende typen LINQ, zoals:

- LINQ to Objects: Voor het bevragen van in-memory data (bijv. lijsten).
- LINQ to SQL: Om SQL-databases te bevragen.
- LINQ to XML: Voor het werken met XML-documenten.

Hierdoor kun je met één uniforme syntax allerlei gegevensbronnen benaderen.

<https://youtu.be/SFuXt6q5U0M?si=hjMBkn6CFPiB47bg>

Source code

Het is verstandig de hele tutorial door te lopen en de code zelf over te nemen. Als je dat zorgvuldig doet leer je er meer van dan wanneer je alleen de code van iemand anders (in dit geval die van mij) doorleest. Maar voor het geval je vastloopt: de source code van het project is te clonen van GitHub:

https://github.com/mgroesink/I2SD_DemoEF.git