

# .NET Advanced

EF Core introductie

# Topics

- Object Relational Mapper (ORM)
- Installatie EF Core
- Opbouw EDM
- Entiteiten in het EDM
- DbContext en DbSet objecten
- Migrations

# EF Core introductie

## Object Relational Mapper (ORM)

# EF CORE introductie - Object Relational Mapper

---



## Doel

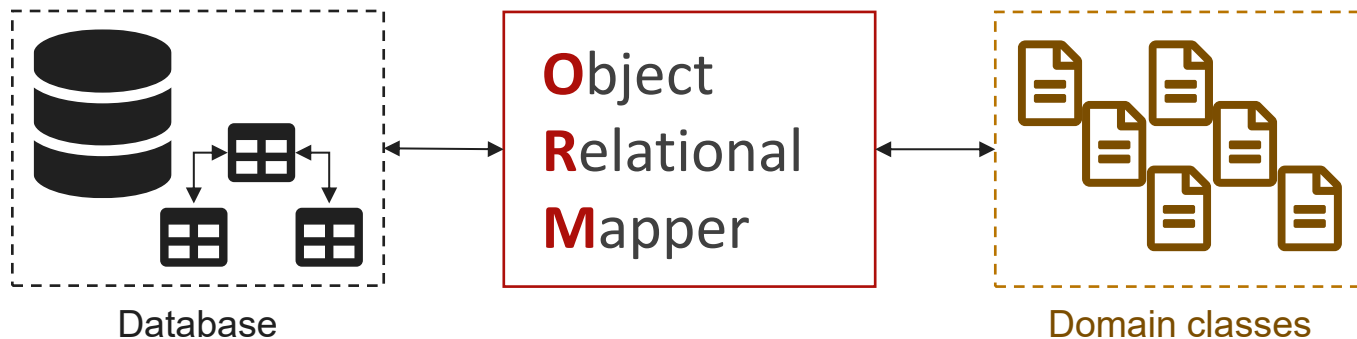
---

- Het concept van een ORM kennen
- De belangrijkste eigenschappen van een ORM kennen

# EF Core introductie - Object Relational Mapper

## Theorie

- In programmeertalen werken we met objecten en collections (Object)
- In een database werken we met tabellen en relaties (Relational)
- Nood aan een vertaalslag tussen beide werelden (Mapper)
- EF Core = ORM





# EF Core introductie - Object Relational Mapper

---

## Theorie

---

### Belangrijkste functionaliteiten in EF Core

- **Modelling**: Entity Data Model vormgeven via DbContext - DbSet - POCO classes
- **Conventions**: Configuratie van het model volgens bepaalde voorgedefinieerde regels
- **Configurations**: Manueel configureren van het model via Annotations of Fluent API
- **Migrations**: Genereren van script om database aan te maken/passen
- **Querying & Saving**: CRUD operaties op de onderliggende datasource m.b.v. LINQ
- **Change tracking**: bijhouden welke wijzigingen gebeuren op in-memory data
- **Concurrency**: Optimistic concurrency (meer info over concurrency [hier](#))
- **Transactions**: automatisch transactiebeheer bij CRUD operaties

# EF Core introductie

Installatie EF Core

# EF Core introductie - Installatie EF Core

---

## Doel

---

- De belangrijkste EF Core packages kennen
- EF Core packages installeren in een project



# EF Core introductie - Installatie EF Core

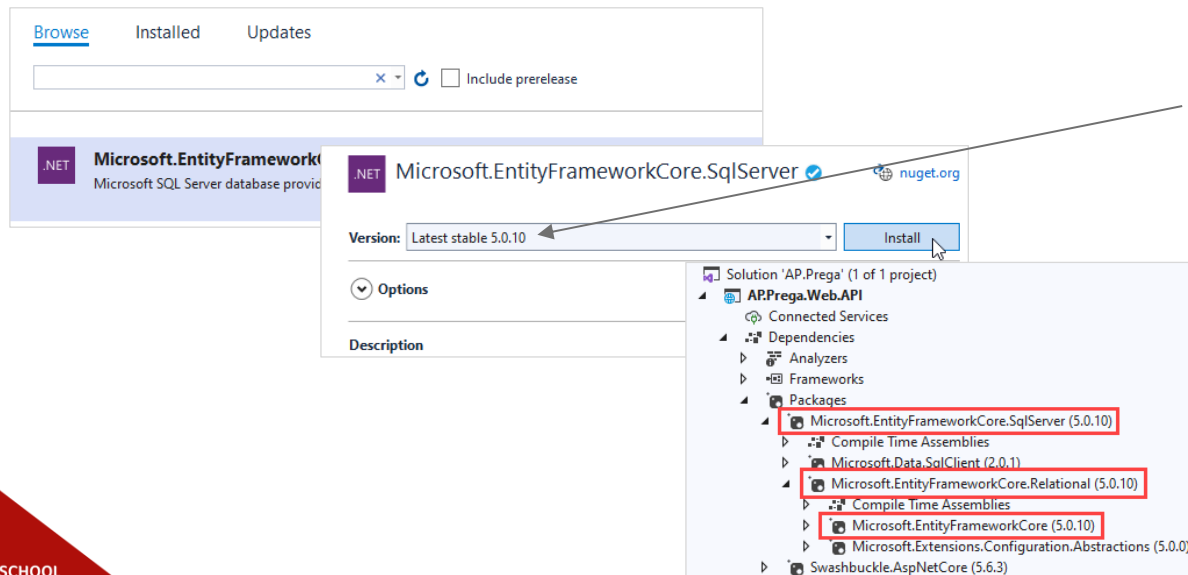
## Theorie

- EF Core packages toe te voegen aan project via NuGet
- Volgende packages zijn nodig bij gebruik van relationele database als datasource:
  - ***Microsoft.EntityFrameworkCore.<databaseprovider>*** (lijst van providers [hier](#))
    - Bij gebruik SQL Server wordt dit *Microsoft.EntityFrameworkCore.SqlServer*
    - Neemt ook ineens volgende benodigde packages mee op
      - *Microsoft.EntityFrameworkCore*
      - *Microsoft.EntityFrameworkCore.Relational*
  - ***Microsoft.EntityFrameworkCore.Tools***
    - Nodig om migrations en andere database generatie gerelateerde taken te kunnen uitvoeren

# EF Core introductie - Installatie EF Core

## Theorie

- Installatie package voor database provider.



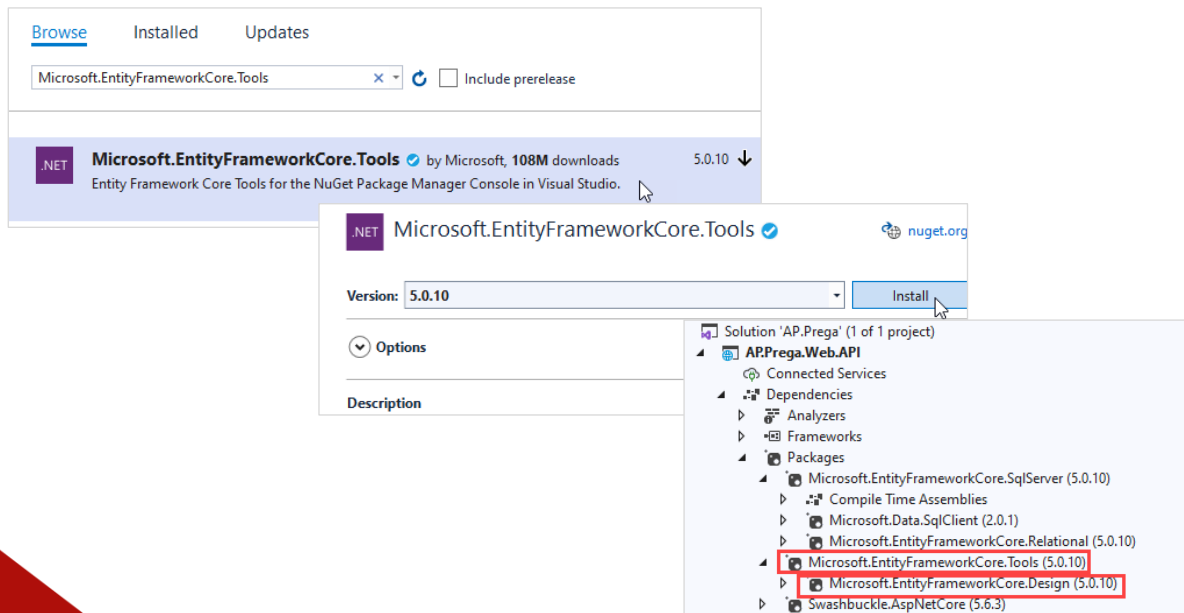
Let op:

Gebruik dezelfde versie als de .NET versie van je applicatie ! Dus als je applicatie met .NET 8 gebouwd wordt, dan neem je ook de EF Core packages van 8.0.xx (neem de hoogste voor xx)  
(dus niet versie 5.0.10 nemen zoals hier aangegeven)

# EF Core introductie - Installatie EF Core

## Theorie

- Installatie package voor Tools



# EF Core introductie

Opbouw EDM

# EF Core introductie - Opbouw EDM

---



## Doel

---

- Het concept van het EDM kennen
- Het doel van het EDM kennen
- De 3 deelmodellen van het EDM kennen
- De 2 manieren van aanpak om een EDM aan te maken kennen



# EF Core introductie - Opbouw EDM

---

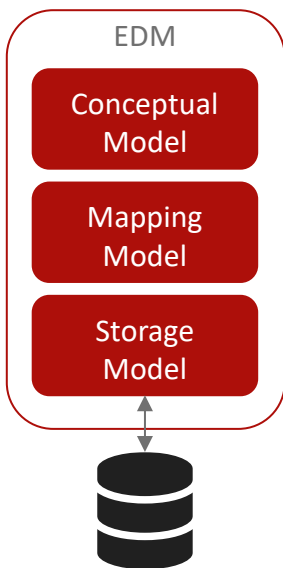
## Theorie

---

- Het Entity Data Model is
  - Een set van concepten
  - Die de structuur van gegevens beschrijft
  - Ongeacht de vorm waarin deze gegevens worden bewaard
- Het Entity Data Model bestaat uit 3 delen
  - Conceptual Model
  - Storage Model
  - Mapping Model

# EF Core introductie - Opbouw EDM

## Theorie



### 1. Conceptual Model

Object-based model, opgebouwd op basis van je DbContext, DbSet, Domain (poco) classes, conventies en configuraties

### 2. Storage Model

Opgebouwd op basis van het conceptuele model (code-first benadering) of database schema (database-first benadering)

### 3. Mapping Model

Zorgt voor vertaalslag tussen conceptuele model en database model

# EF Core introductie - Opbouw EDM

---

## Theorie

---

- Opbouwen van Entity Data Model kan op 2 manieren aangepakt worden
  - Code-first benadering
  - Database-first benadering
  - (Model-first, in EF 6 maar niet in EF Core)

# EF Core introductie - Opbouw EDM

## Theorie

### Code-first benadering

- Je bouwt je Conceptual Model en laat EF Core de database hiervoor genereren
- Meest gebruikte benadering bij ontwikkeling nieuwe app



# EF Core introductie - Opbouw EDM

## Theorie

### Database-first benadering

- Je hebt een database en laat EF Core op basis hiervan het Conceptual Model genereren
- Interessant voor herontwikkeling legacy apps waar database al bestaat





# EF Core introductie

Entiteiten in het EDM

# EF Core introductie - Entiteiten in het EDM

---

## Doel

---

- Het concept van een entiteit kennen
- Een entiteit kunnen aanmaken in het EDM

# EF Core introductie - Entiteiten in het EDM

---

## Theorie

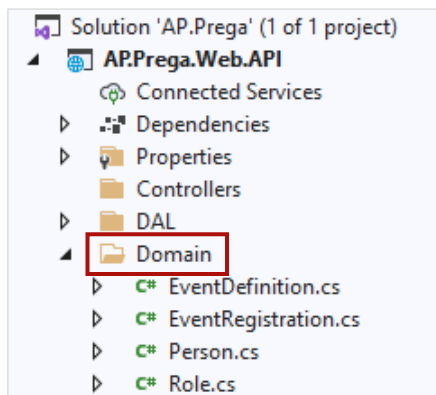
---

- Een entiteit is een enkelvoudig, identificeerbaar en afzonderlijk object
- Voor elke entiteit voorzien we een POCO class object
- Een entiteit komt meestal overeen met een tabel in de onderliggende datastore
- Bij voorkeur in een aparte directory om cluttering van het project te vermijden
- Let op namespace naam bij toevoegen van POCO class objecten in aparte directory

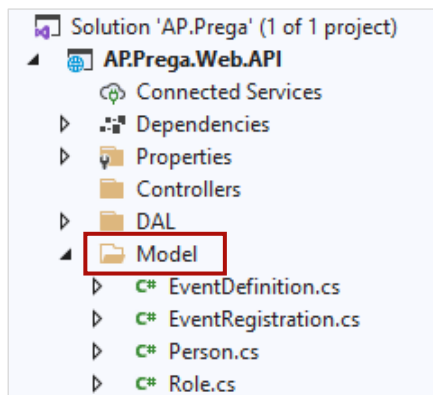
# EF Core introductie - Entiteiten in het EDM

## Theorie

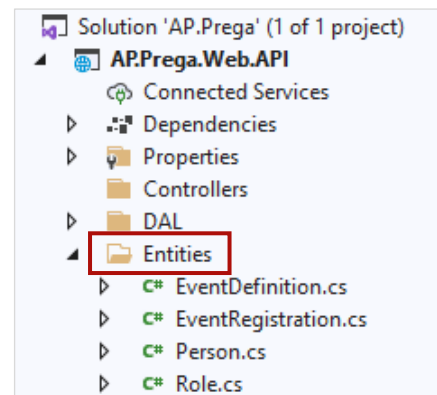
- Mogelijke namen voor directory
  - Domain (referentie naar het concept domein wat verwijst naar alle functionaliteit)
  - Model (Verwijst naar het ganse data model van de onderliggende datastore)
  - Entities (Verzamelnaam)



OF



OF



# EF Core introductie - Entiteiten in het EDM

---

## Aandachtspunten

---

- Let op naamgeving entiteit!
  - Zelfstandig naamwoord
  - Enkelvoud
  - Engelstalig
  - Camel Casing



# EF Core introductie - Entiteiten in het EDM

---

## Theorie

---

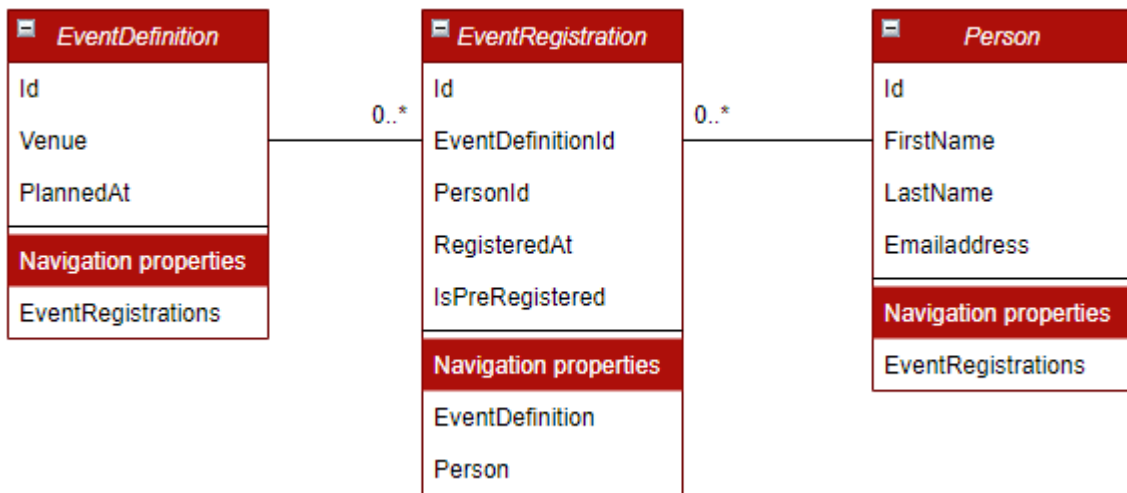
Hoe opbouwen?

- Elke eigenschap implementeren als een property (Ook PK en FK)
  - Hanteer voor FK conventie [Parent\_entity\_name]Id (bv. PersonId)
- Navigation properties voorzien
  - Representeren een relatie tussen 2 objecten
  - Beide kanten van de relatie (PK en FK) implementeren
  - Parent entiteit: Property van het child type (meestal als een List)
  - Child entiteit: Property van het parent type

# EF Core introductie - Entiteiten in het EDM

## Voorbeelden

Een voorbeeld model

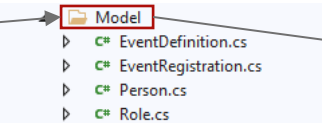


# EF Core introductie - Entiteiten in het EDM

## Voorbeelden

```
namespace AP.Prega.Web.API.Model
{
    2 references
    public class EventDefinition
    {
        References
        public int Id { get; set; }
        References
        public string Venue { get; set; }
        References
        public DateTime PlannedAt { get; set; }

        References
        public List<EventRegistration> EventRegistrations { get; set; }
    }
}
```



Naam van directory wordt toegevoegd aan namespace!

Property voor elke eigenschap van een entiteit

Navigation property naar child entiteit

# EF Core introductie - Entiteiten in het EDM

## Voorbeelden

```
namespace AP.Prega.Web.API.Model
```

```
{
```

```
    2 references
```

```
    public class EventRegistration
```

```
    {
```

```
        0 references
```

```
        public int Id { get; set; }
```

```
        0 references
```

```
        public int EventDefinitionId { get; set; }
```

```
        0 references
```

```
        public int PersonId { get; set; }
```

```
        0 references
```

```
        public DateTime RegisteredAt { get; set; }
```

```
        0 references
```

```
        public bool IsPreRegistered { get; set; }
```

```
    }  
}
```

```
    0 references
```

```
    public EventDefinition EventDefinition { get; set; }
```

```
    0 references
```

```
    public Person Person { get; set; }
```

Foreign Key velden als properties

Navigation properties naar parent entiteiten

# EF Core introductie

DbContext en DbSet objecten



# EF Core introductie - DbContext en DbSet objecten

---

## Doel

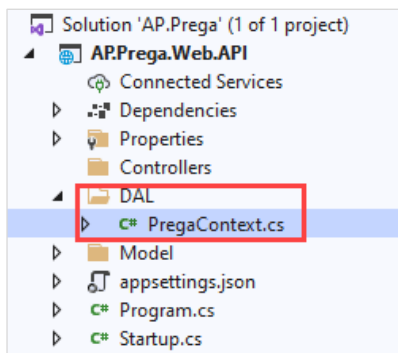
---

- Het doel van de DbContext en DbSet objecten kennen
- Een DbContext object kunnen opzetten

# EF Core introductie - DbContext en DbSet objecten

## Theorie

- DbContext object instantie representeert een sessie met een database
- Altijd laten overerven van het Microsoft.EntityFrameworkCore.DbContext object
- Naamgeving: *<app naam>Context*
- Maak een aparte map DAL om alle datalogica gerelateerde classes onder te brengen



```
namespace AP.Prega.Web.API.DAL
{
    1 reference
    public class PregaContext : DbContext
    {
    }
}
```

# EF Core introductie - DbContext en DbSet objecten

---

## Theorie

---

- Alle interactie met de onderliggende datastore gebeurt via het DbContext object
- Gebeurt via DbSet<TEntity> properties
  - Je voorziet dus voor elke entiteit een DbSet property
  - Je voorziet dus het type van de entiteit op de plaats van de generic
  - DbSet<TEntity> gebruiken om instances van TEntity te queryen of bewaren
  - Interactie met database-tabellen gebeurt via deze DbSet<TEntity> properties
  - Query's worden geschreven in LINQ en achterliggend vertaald naar SQL

# EF Core introductie - DbContext en DbSet objecten

## Voorbeelden

```
public class PregaContext : DbContext
{
    // 0 references
    public DbSet<EventDefinition> EventDefinitions { get; set; }
    // 0 references
    public DbSet<EventRegistration> EventRegistrations { get; set; }
    // 0 references
    public DbSet<Person> Persons { get; set; }
}
```

Context object erft over van DbContext

Elke entiteit wordt geïmplementeerd als een DbSet

# EF Core introductie

## Migrations

# EF Core introductie - Migrations

---



## Doel

---

- Het concept van een migration kennen
- Een migration kunnen aanmaken
- Een database updaten op basis van een migration
- Een SQL script kunnen genereren op basis van een migration

# EF Core introductie - Migrations

---

## Theorie

---

- Bij creatie/wijziging van het EDM moet ook de database mee volgen
- M.b.v. EF Core Tools kan dit geautomatiseerd worden
  - Via een commando wordt een code file gegeneerd
  - Hierin zit een Up() en Down() method
  - Up() method beschrijft wijzigingen die doorgevoerd zullen worden in de database
  - Down() method beschrijft hoe deze wijzigingen ongedaan te maken
  - In geval database nog niet bestaat wordt deze aangemaakt bij het uitvoeren van een migration
- Dit concept noemen we het aanmaken van een **Migration**



# EF Core introductie - Migrations

---

## Theorie

---

- Benodigheden om een migratie aan te maken
  1. EF Core packages toevoegen aan project
  2. DbContext object waarin het model via DbSet is opgebouwd
  3. Connectionstring om te bepalen waar de migratie moet worden uitgevoerd
    - Te plaatsen in appsettings.json file, nooit hardcoded in code
  4. Registratie van het DbContext object in de DI (Dependency Injection) Container
  5. Uitwerken DbContext constructor voor gebruik met DI
  
- Punt 1 en 2 reeds uitgewerkt

# EF Core introductie - DbContext en DbSet objecten

## Voorbeelden

3. Connectionstring om te bepalen waar de migratie moet worden uitgevoerd  
(Te plaatsen in appsettings.json)

Object waaronder connectionstrings gebundeld worden



```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft": "Warning",
      "Microsoft.Hosting.Lifetime": "Information"
    }
  },
  "ConnectionStrings": {
    "Prega": "Data Source=DESKTOP-544U9VH;Initial Catalog=Prega;Integrated Security=True"
  },
  "AllowedHosts": "*"
}
```

Naam om naar te refereren in code

De effectieve connectionstring

# EF Core introductie - Migrations

---

## Theorie

---

4. Registratie van het DbContext object in de DI (Dependency Injection) Container
  - Gebeurt in de main() method in de **Program** class
  - Aanroepen van AddDbContext<TContext>() method van het ServiceCollection object (waarbij TContext de naam van je DbContext is)
  - Action van het type DbContextOptionsBuilder meegeven als argument (meer over Actions in het topic LINQ)
  - Via deze Action de method UseSqlServer() van het DbContextOptionsBuilder aanroepen met de connectionstring als parameter (Bij gebruik van een andere Database Provider zal een andere method nodig zijn)

# EF Core introductie - Migrations

## Voorbeelden

```
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Logging;
using Prega.Application;
using Prega.Domain;
using Prega.Infrastructure;
using Prega.Web;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

public static void Main(string[] args)
{
    var builder = WebApplication.CreateBuilder(args);

    // Add services to the container.

    builder.Services.AddControllers();
    builder.Services.AddDbContext<GamestoreContext>(options => options.UseSqlServer("name=ConnectionStrings:Prega"));

    // Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swashbuckle
    builder.Services.AddEndpointsApiExplorer();
    builder.Services.AddSwaggerGen();

    var app = builder.Build();
}
```

```
"ConnectionStrings": {
  "Prega": "Data Source=DESKTOP-544U9VH;Initial
```

DbContext registreren via AddDbContext method

Action van type DbContextOptionsBuilder met aanroep van UseSqlServer() method

# EF Core introductie - Migrations

---

## Theorie

---

5. Uitwerken DbContext constructor voor gebruik met DI
  - Context object erft over van base DbContext class
  - Gebruik DbContextOptions<TContext> als argument voor constructor
  - Roep de base constructor aan met ditzelfde argument

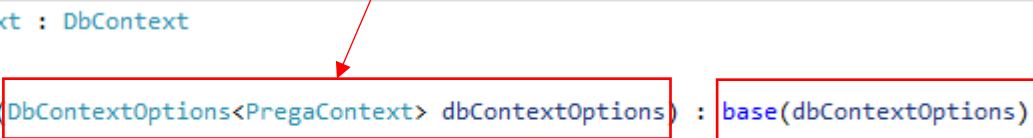
# EF Core introductie - Migrations

## Voorbeelden

- Bij het registreren van de DbContext in de DI container creëren we via een Lambda expression een DbContextOptions instance waarbij we aangeven dat SQL Server als database provider gebruikt zal worden.
- Wanneer later de DI Container een DbContext object instantieert zal een DbContextOptions object met deze instelling meegegeven worden aan de constructor van de DbContext

```
services.AddDbContext<PregaContext>(options => options.UseSqlServer("name=ConnectionStrings:Prega"));
```

```
public class PregaContext : DbContext
{
    References
    public PregaContext(DbContextOptions<PregaContext> dbContextOptions) : base(dbContextOptions)
    {
    }
}
```



# EF Core introductie - Migrations

---

## Theorie

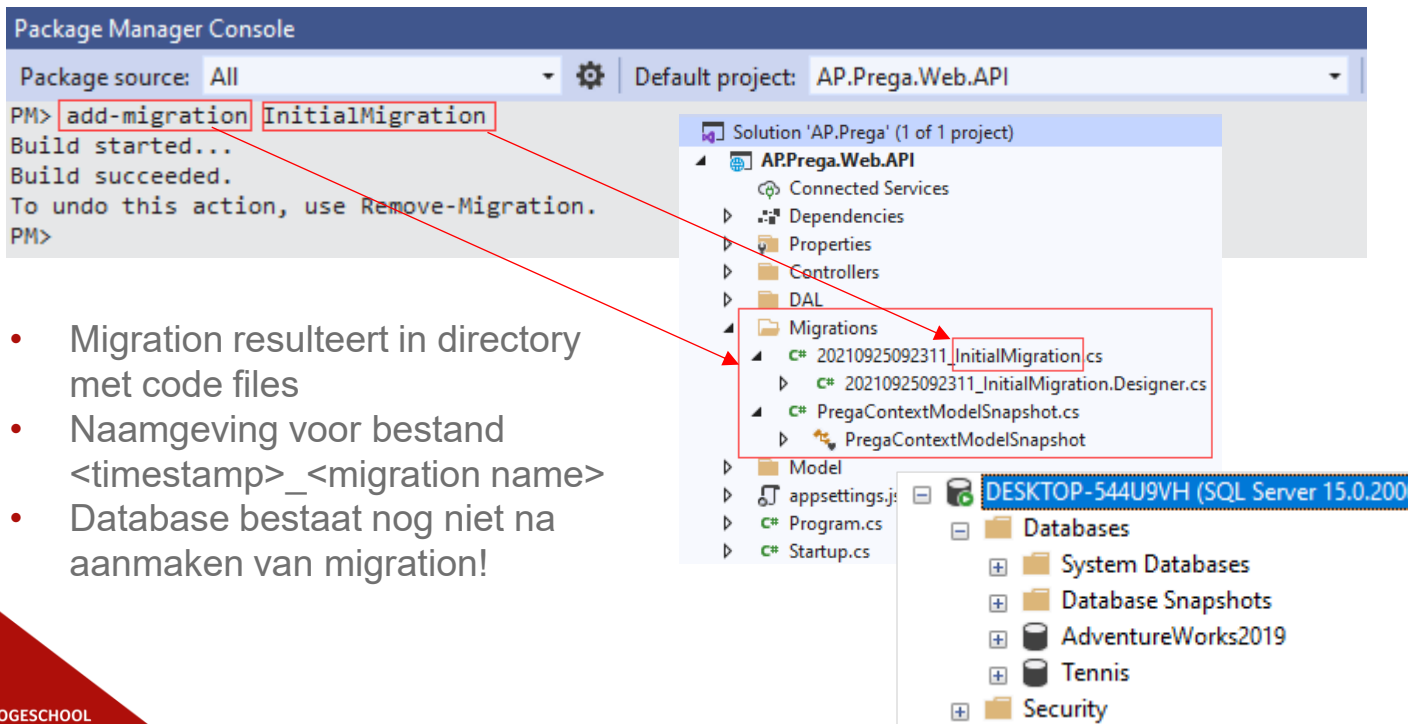
---

- Migration aanmaken:
  - ⇒ Los eventuele build errors op in de solution
  - ⇒ Ga naar de Package Manager Console (PMC)
  - ⇒ Selecteer het API project in de keuzelijst
  - ⇒ Typ het commando `add-migration <migration name>`
- Migrations folder wordt aangemaakt in project
- In deze folder worden de code files per migratie gegenereerd



# EF Core introductie - Migrations

## Voorbeelden



The screenshot shows the Package Manager Console with the command `PM> add-migration InitialMigration` entered. The console output indicates that the build started and succeeded, and provides the instruction: "To undo this action, use Remove-Migration." The Solution Explorer on the right shows the project structure for 'AP.Prega.Web.API'. A new 'Migrations' folder has been created, containing the files `20210925092311_InitialMigration.cs` and `20210925092311_InitialMigration.Designer.cs`. A red box highlights these files, and a red arrow points from the 'InitialMigration' text in the command to this box. Below the Solution Explorer, a snippet of the SQL Server Enterprise Manager shows the 'Databases' folder expanded, listing 'System Databases', 'Database Snapshots', 'AdventureWorks2019', 'Tennis', and 'Security'.

- Migration resulteert in directory met code files
- Naamgeving voor bestand `<timestamp>_<migration name>`
- Database bestaat nog niet na aanmaken van migration!

# EF Core introductie - Migrations

---

## Theorie

---

- *<timestamp>\_<migration name>.cs*
  - Up() method (logica om database up te daten)
  - Down() method (logica om de migratie terug te draaien op de database)
- *<timestamp>\_<migration name>.Designer.cs*
  - Metadata gebruikt door EF Core
- *<contextclassname>ModelSnapshot.cs*
  - Bevat een snapshot van het huidige model
  - Bij een volgende creatie van een migration wordt deze snapshot gebruikt om verschillen te detecteren in het model

# EF Core introductie - Migrations

## Voorbeelden

0 references

```
protected override void Up(MigrationBuilder migrationBuilder)
{
    migrationBuilder.CreateTable(
        name: "EventDefinitions",
        columns: table => new
        {
            Id = table.Column<int>(type: "int", nullable: false)
                .Annotation("SqlServer:Identity", "1, 1"),
            Venue = table.Column<string>(type: "nvarchar(max)", nullable: true),
            PlannedAt = table.Column<DateTime>(type: "datetime2", nullable: false)
        },
        constraints: table => [...]);
```

0 references

```
protected override void Down(MigrationBuilder migrationBuilder)
{
    migrationBuilder.DropTable(
        name: "EventRegistrations");

    migrationBuilder.DropTable(
        name: "EventDefinitions");

    migrationBuilder.DropTable(
        name: "Persons");
}
```

# EF Core introductie - Migrations

---

## Theorie

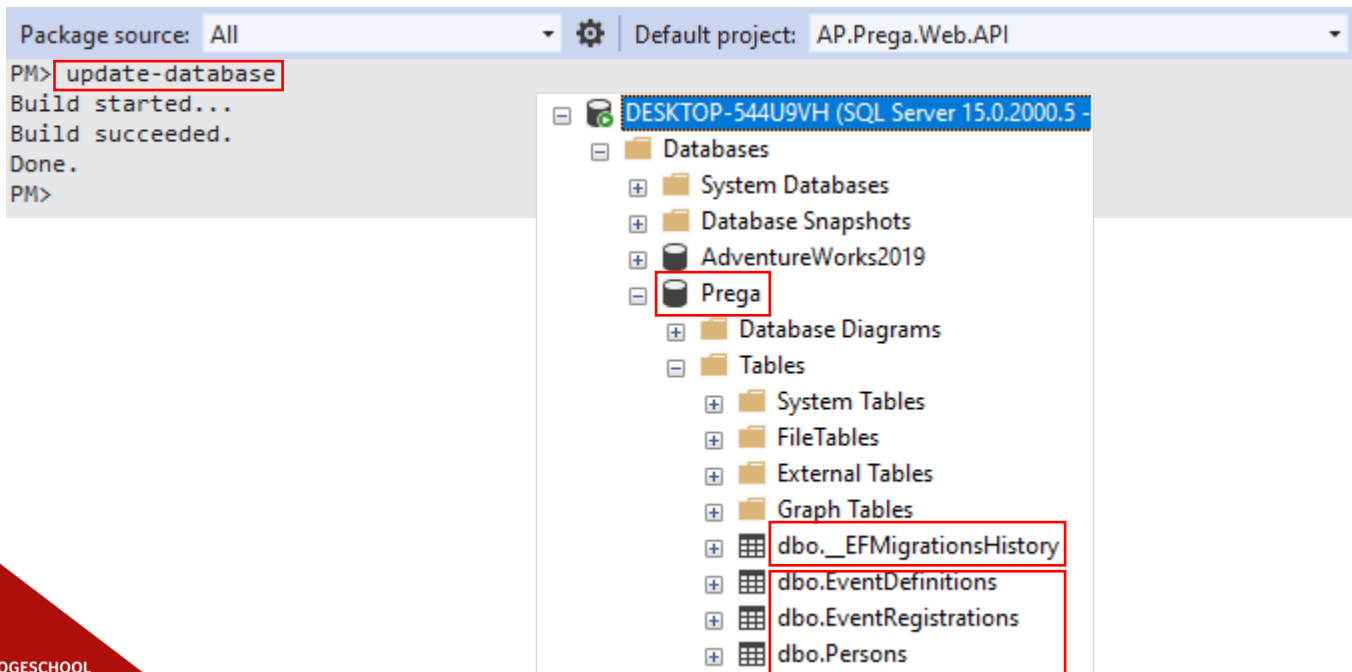
---

- Na aanmaken van een migration moeten we deze uitvoeren
  - ⇒ Ga naar de Package Manager Console (PMC)
  - ⇒ Selecteer het API project in de keuzelijst
  - ⇒ Typ het commando `update-database`
- In geval je niet rechtstreeks de database mag benaderen kan een SQL script gegenereerd worden
  - ⇒ Ga naar de Package Manager Console (PMC)
  - ⇒ Selecteer het API project in de keuzelijst
  - ⇒ Typ het commando `script-migration`

# EF Core introductie - Migrations

## Voorbeelden

Migratie uitvoeren om de database up-to-date te brengen (of aan te maken)



The screenshot displays the Visual Studio interface during a database migration. The top toolbar shows 'Package source: All' and 'Default project: AP.Prega.Web.API'. The command window on the left shows the following output:

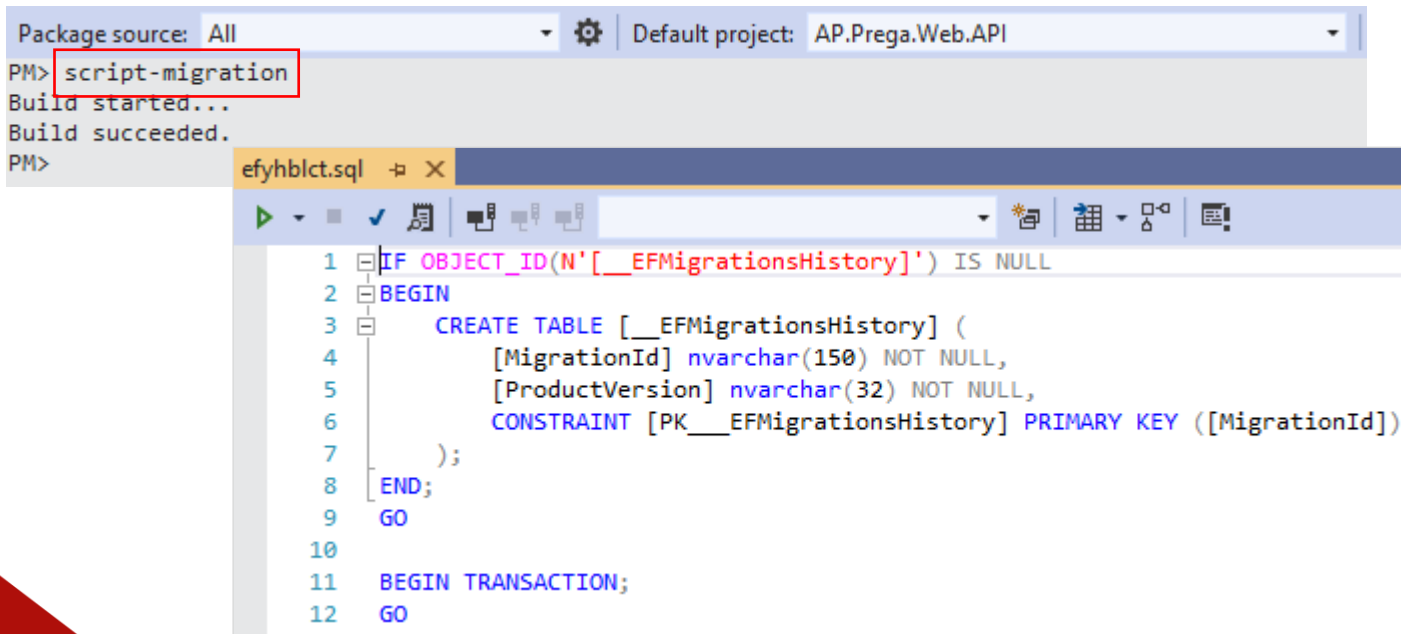
```
PM> update-database
Build started...
Build succeeded.
Done.
PM>
```

The right pane shows the 'DESKTOP-544U9VH (SQL Server 15.0.2000.5 -)' server explorer. The 'Databases' folder is expanded, showing 'System Databases', 'Database Snapshots', 'AdventureWorks2019', and 'Prega'. The 'Prega' database is selected and expanded, showing 'Database Diagrams', 'Tables', 'System Tables', 'FileTables', 'External Tables', 'Graph Tables', and a list of tables: 'dbo.\_\_EFMigrationsHistory', 'dbo.EventDefinitions', 'dbo.EventRegistrations', and 'dbo.Persons'. The 'Prega' database and the list of tables are highlighted with red boxes.

# EF Core introductie - Migrations

## Voorbeelden

Creatie van een SQL script



The screenshot shows the Visual Studio interface. At the top, the 'Package source' is set to 'All' and the 'Default project' is 'AP.Prega.Web.API'. Below this, the command prompt shows the command 'script-migration' being executed, with the output 'Build started...', 'Build succeeded.', and 'PM>'. The SQL script editor, titled 'efyhlbt.sql', contains the following SQL code:

```
1 IF OBJECT_ID(N'[__EFMigrationsHistory]') IS NULL
2 BEGIN
3     CREATE TABLE [__EFMigrationsHistory] (
4         [MigrationId] nvarchar(150) NOT NULL,
5         [ProductVersion] nvarchar(32) NOT NULL,
6         CONSTRAINT [PK__EFMigrationsHistory] PRIMARY KEY ([MigrationId])
7     );
8 END;
9 GO
10
11 BEGIN TRANSACTION;
12 GO
```

# EF Core introductie

CRUD operaties



# EF Core introductie - CRUD operaties

---

## Theorie

---

- We werken vanuit een Controller in het API project
- Injecteren van DbContext in Controller via constructor van Controller
- Toewijzen aan private member

# EF Core introductie - CRUD operaties

## Voorbeelden

Injecteren van de DbContext in de controller

```
[Route("api/[controller]")]
[ApiController]
1 reference
public class EventDefinitionController : ControllerBase
{
    private PregaContext _context;

    0 references
    public EventDefinitionController(PregaContext context)
    {
        _context = context;
    }
}
```

# EF Core introductie - Create

---

## Theorie

---

- Create (Toevoegen van een entity instance)
- Entity instance toevoegen aan bijhorende DbSet property
  - Via Add() method
  - Voegt entity instance toe aan de DbSet maar niet automatisch aan de database
  - ChangeTracker houdt per instance de EntityState bij (In dit geval Added)
  - Pushen naar database via SaveChanges() method van DbContext

# EF Core introductie - Create

## Voorbeelden

[HttpPost]

0 references

```
public IActionResult Add([FromBody] EventDefinition eventDefinition)
{
    _context.EventDefinitions.Add(eventDefinition);
    _context.SaveChanges();

    return Created("", eventDefinition);
}
```

JSON object wordt via de body van de message bezorgd

Toevoegen van de instance aan de DbSet

Pushen van de toegevoegde instance naar de database

Terugsturen van de bewaarde entiteit zoals ze momenteel in de databank werd bewaard (dus met de Id die door de databank werd toegekend reeds ingevuld).  
Status code = 201 (met succes aangemaakt)

# EF Core introductie - Read

---

## Theorie

---

- Read (Lezen van gegevens)
- Via DbSet property rechtstreeks (alle rows ophalen uit de database)
- Via LINQ Query op DbSet (specifiek)
- EF Core vertaalt query naar SQL achter de schermen
- Voert SQL uit
- Geeft resultaat terug

# EF Core introductie - Read

## Voorbeelden

[HttpGet]

0 references

```
public IActionResult GetAll()
{
    var result = _context.EventDefinitions;
    return Ok(result);
}
```

Ophalen van alle rijen

Teruggeven van het resultaat in de response (omgezet naar JSON) met http status code 200

**Merk op: zelfs als er geen gegevens in de databank zouden aanwezig zijn, dat we dan gewoon een lege lijst terugsturen (dus geen 404 in dit geval)**

# EF Core introductie - Read

## Voorbeelden

```
[HttpGet]  
[Route("{id}")]
```

0 references

```
public IActionResult GetById(int id)
```

```
{
```

```
    var result = _context.EventDefinitions.Find(id);
```

```
    if (result == null)  
        return NotFound();
```

```
    return Ok(result);
```

```
}
```

Opvragen van 1  
element aan de hand  
van de primaire sleutel

Teruggeven van het resultaat in de response met http status code 200  
(of status code 404 indien het element niet aanwezig was in de databank)



# EF Core introductie - Update

---

## Theorie

---

- Update (Aanpassen van een entity instance)
- Bestaande entity opvragen uit de db (ook ineens om na te gaan of ze wel degelijk bestaat)
- Entity instance aanpassen (alle properties die mogen worden aangepast worden overschreven)
- SaveChanges() method van de overkoepelende DbContext aanroepen om de changes te pushen naar de onderliggende datastore
  - Zoekt naar alle entity instances met EntityState Modified om deze te updaten

# EF Core introductie - Update

## Voorbeelden

```
[HttpPut]
[Route("{id}")]
0 references
public IActionResult Update(int id, [FromBody] EventDefinition eventDefinition)
{
    if (eventDefinition == null || id != eventDefinition.Id)
        return BadRequest("The Id in the route does not match the Id in the body");

    var result = _context.EventDefinitions.Find(id);
    if (result == null)
        return NotFound();

    //Update of all members.
    result.Venue = eventDefinition.Venue;
    result.PlannedAt = eventDefinition.PlannedAt;

    _context.SaveChanges();
    return Ok(result);
}
```

Pushen van de update naar de database.

- Bij een update wordt (=conventie) ook steeds de Id meegegeven via de route.
- Wanneer de properties worden aangepast zal EF onthouden dat het betreffende object nog moet worden aangepast in de db (de entity krijgt een status: "Modified").
- De update naar de db wordt echter pas gedaan bij de `SaveChanges()` aanroep !

# EF Core introductie - Delete

---

## Theorie

---

- Delete (Verwijderen van een entity instance)
- Bestaande entity ophalen om na te gaan of ze wel degelijk (nog) bestaat
- DbContext method Remove() uitvoeren met entity instance als parameter
  - EntityState van entity instance wordt ingesteld op Deleted
- SaveChanges() method van de overkoepelende DbContext aanroepen om de changes te pushen naar de onderliggende datastore
  - Zoekt naar alle entity instances met EntityState Deleted om deze te verwijderen

# EF Core introductie - Delete



## Voorbeelden

```
[HttpDelete("{id}")]
```

0 references

```
public IActionResult Remove(int id)
```

```
{  
    var current = _context.EventDefinitions.Find(id);  
    if (current == null)  
        return NotFound($"The entity with the specified Id does not exist");  
  
    _context.EventDefinitions.Remove(current);  
  
    _context.SaveChanges();  
    return NoContent();  
}
```

Nagaan of de entiteit wel degelijk bestaat

Row verwijderen via Remove()  
method van de DbContext

Status code = 204 ,  
verwijderen is gelukt.

Pushen naar de database van de  
delete operatie op de te verwijderen  
entity instance

# Oefeningen

---

## Deel 2 - Oefenbundels 1 & 2

---

- Surf naar Digitap
- Download de opgave
- Los de vragen op

# EF Core introductie



## Bronnen

- <https://www.entityframeworktutorial.net/>
- <https://www.entityframeworktutorial.net/efcore/entity-framework-core.aspx>
- [https://www.tutorialspoint.com/entity\\_framework/entity\\_framework\\_data\\_model.htm](https://www.tutorialspoint.com/entity_framework/entity_framework_data_model.htm)
- <https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/entity-data-model>
- <https://www.entityframeworktutorial.net/efcore/entity-framework-core-dbcontext.aspx>
- <https://www.entityframeworktutorial.net/efcore/entity-framework-core-console-application.aspx>
- <https://www.entityframeworktutorial.net/efcore/entity-framework-core-migration.aspx>
- <https://www.entityframeworktutorial.net/efcore/update-data-in-entity-framework-core.aspx>