.NET Core: Developing Cross-Platform Web Apps with ASP.NET Core – Workshop*PLUS*

< Engineer Name >

Customer Engineer

v3.1

Conditions and Terms of Use

Microsoft Confidential

This training package is proprietary and confidential, and is intended only for uses described in the training materials. Content and software is provided to you under a Non-Disclosure Agreement and cannot be distributed. Copying or disclosing all or any portion of the content and/or software included in such packages is strictly prohibited.

The contents of this package are for informational and training purposes only and are provided "as is" without warranty of any kind, whether express or implied, including but not limited to the implied warranties of merchantability, fitness for a particular purpose, and non-infringement.

Training package content, including URLs and other Internet Web site references, is subject to change without notice. Because Microsoft must respond to changing market conditions, the content should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information presented after the date of publication. Unless otherwise noted, the companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, e-mail address, logo, person, place, or event is intended or should be inferred.

Copyright and Trademarks

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

For more information, see Use of Microsoft Copyrighted Content at

http://www.microsoft.com/en-us/legal/intellectualproperty/permissions/default.aspx

Azure, Microsoft, Microsoft Corporate Logo, SQL Server, Visual Studio and Windows Phone are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. Other Microsoft products mentioned herein may be either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. All other trademarks are property of their respective owners.

Module 3: Models and EF Core

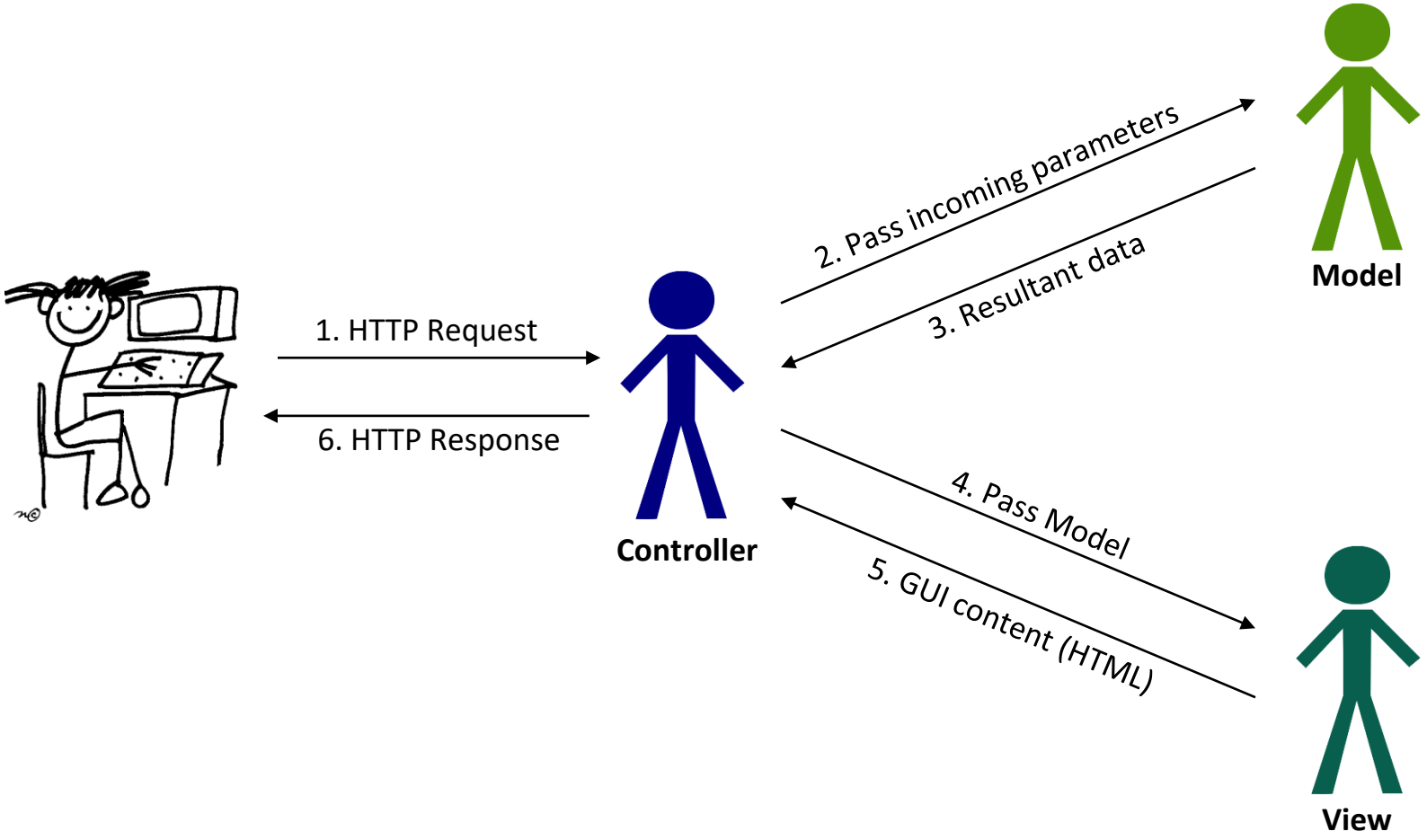
Module Overview

Module 3: Models

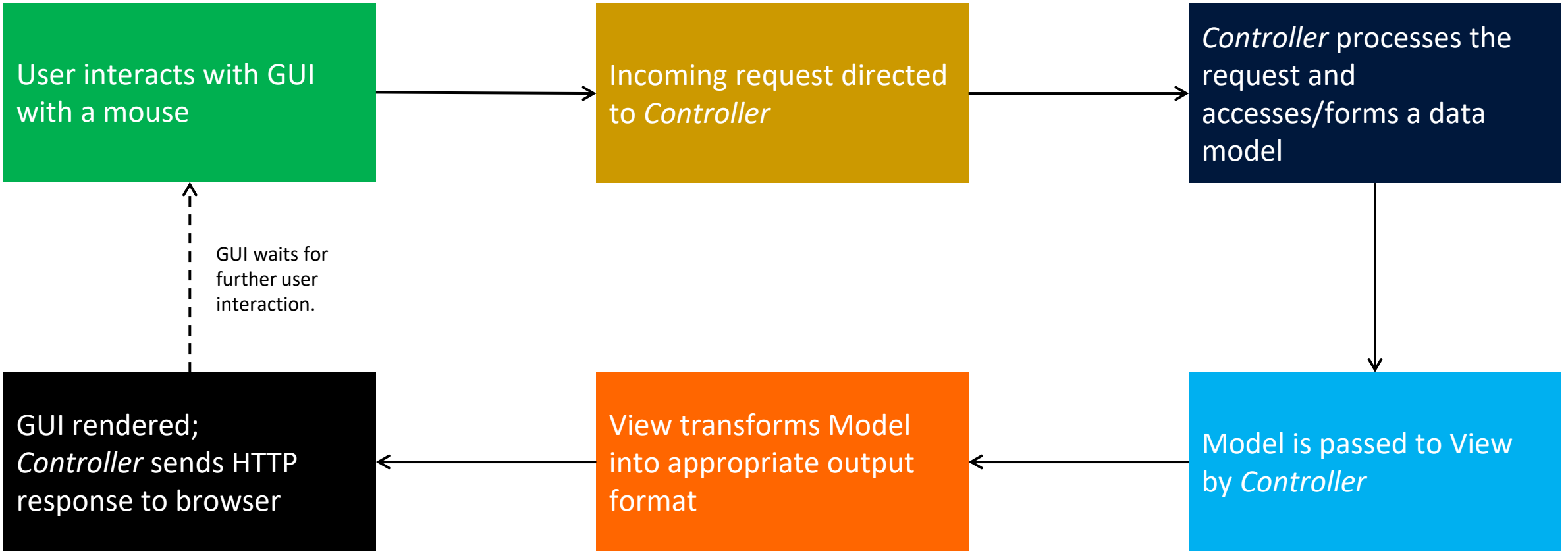
Section 1: MVC Design Pattern

Lesson: Overview

Model View Controller (MVC) Design Pattern



MVC Control Flow



Module 3: Models

Section 2: Model Fundamentals

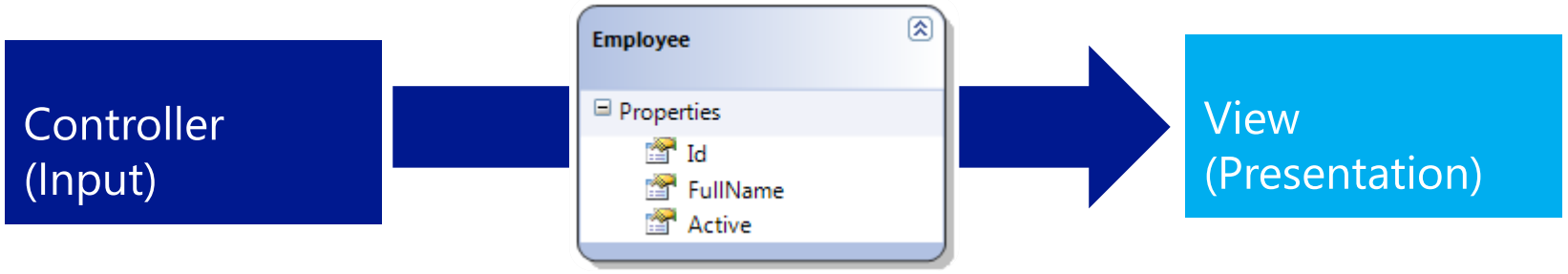
Lesson: Role of Models

Model

- A set of .NET classes that:
  - Describe data that the application is working with
  - Implement the **rules** or **logic** for how the data can be changed/manipulated
- Model state can be retrieved and stored in any form:
  - Relational databases
  - Comma-separated text files
  - RESTful web services
- It can use any data access technology for accessing and manipulating data
  - Object Relational Mapping frameworks like Entity Framework (EF)

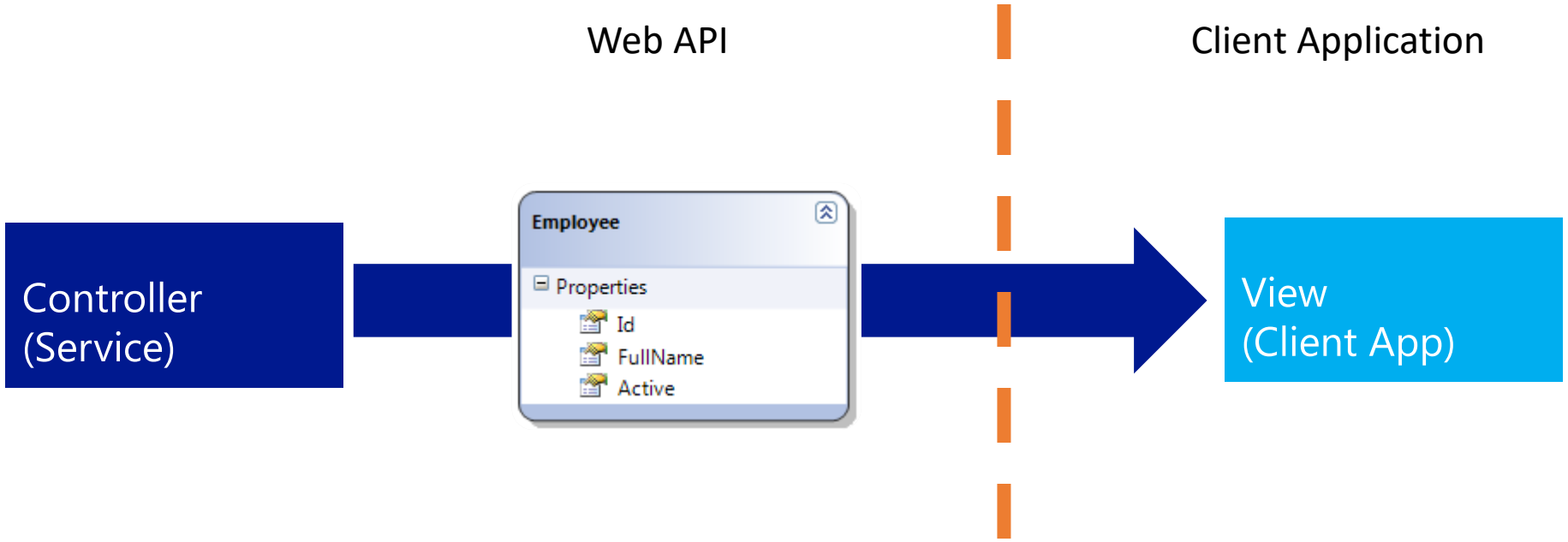
Role of a Model

- The “Model” is the medium of communication between Controllers and Views
- It responds to requests for information about its state (usually from view)
- It changes states in the data source as per the request of controller



Role of a Model

- Building a RESTful service or WebAPI? The pattern still applies!



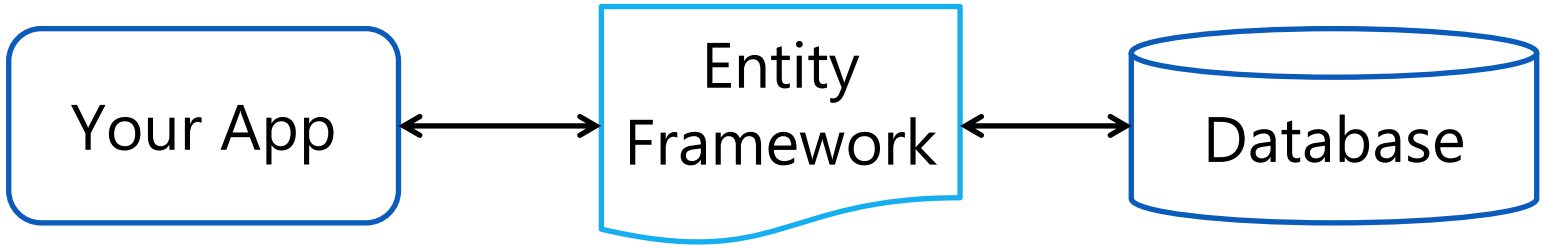
Module 3: Models

Section 3: Model Development

Lesson: Development with Entity Framework

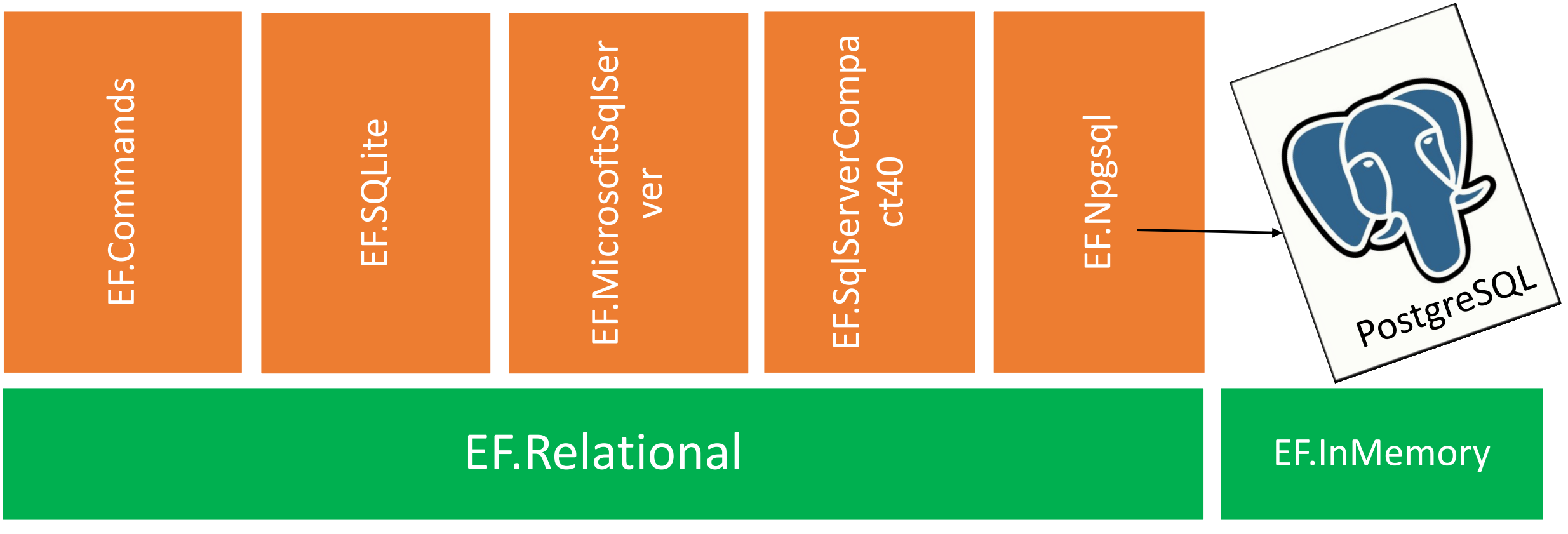
Entity Framework

- Object-relational mapping framework by Microsoft
  - It understands how to store .NET objects in a **relational** database.
  - It retrieves and manipulates data as strongly typed objects using LINQ query
- It provides:
  - Change tracking
  - Identity resolution
  - Dev-time tooling
  - Query translation
  - More!

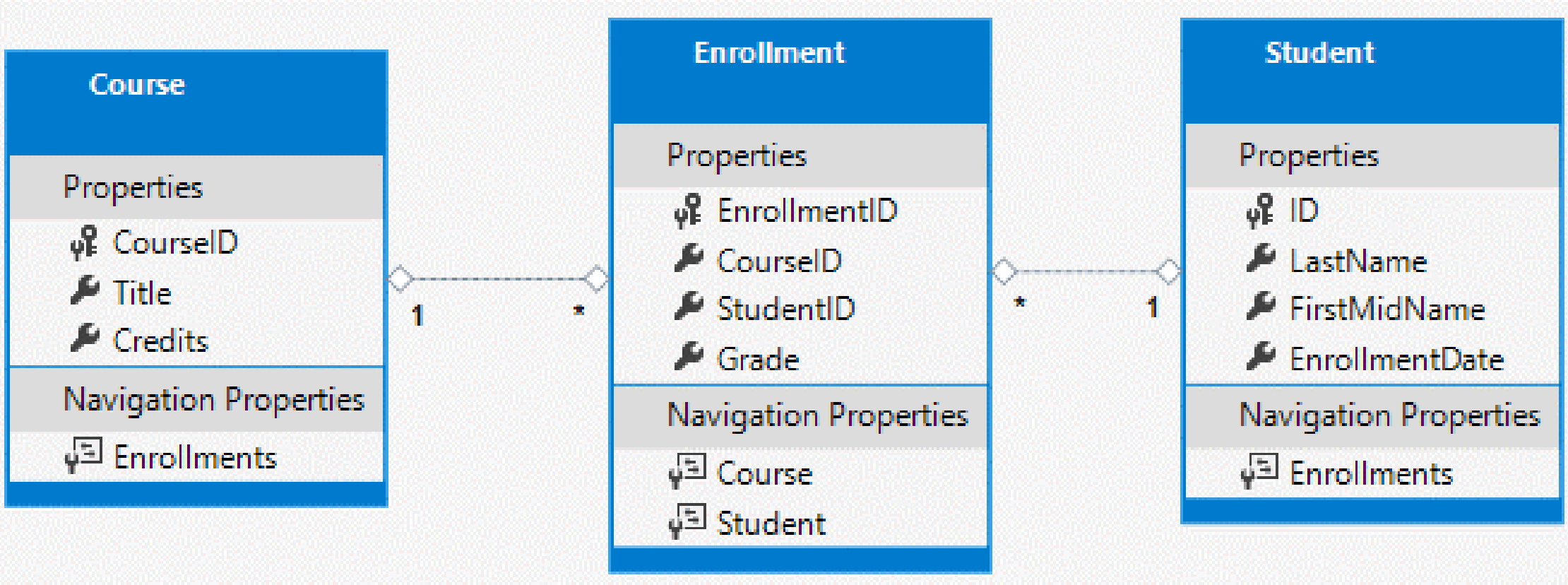


- Open-source and Cross-platform!
- Both Entity Framework and Entity Framework Core will continue to develop separately

EF Architecture



Our Data Domain – Contoso University



Model Development

- A model can be created with a .NET class
- Primary key, foreign key, and navigation properties are defined in the class
- Class (Enrollment) will be converted into a database table
- Class properties (*EnrollmentID*, *CourseID*, etc.) will be converted into table attributes

```
namespace ContosoUniversity.Models
{
    public enum Grade
    {
        A, B, C, D, F
    }

    public class Enrollment
    {
        public int EnrollmentID { get; set; }
        public int CourseID { get; set; }
        public int StudentID { get; set; }
        public Grade? Grade { get; set; }

        public Course Course { get; set; }
        public Student Student { get; set; }
    }
}
```

Model Relationships

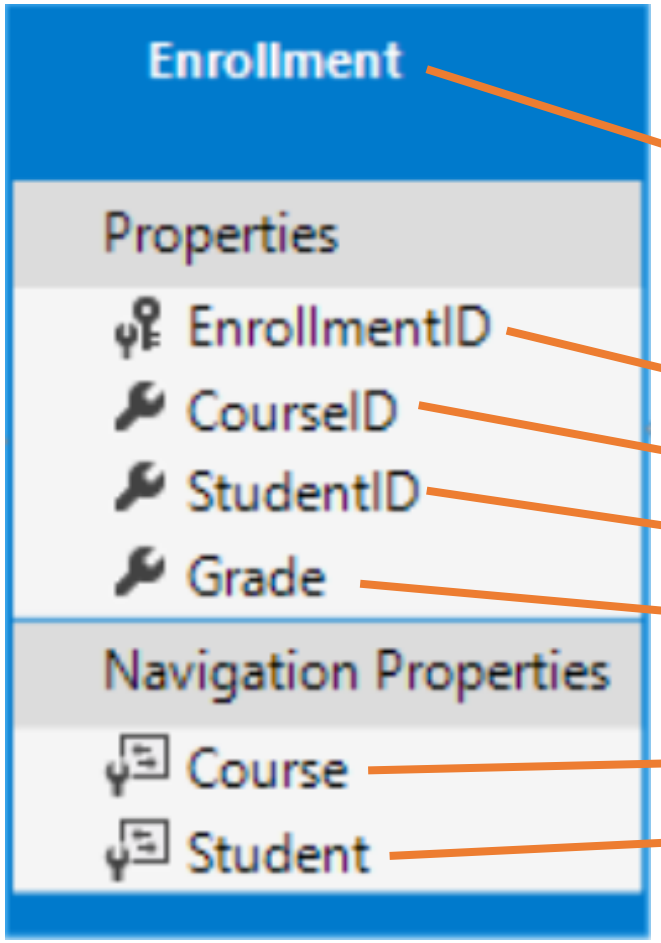
- Navigation property
  - Navigational property holds other entities that are related to this entity
  - *Student* and *Course* are navigation properties.
- Foreign key property
  - It is not required in a model object
  - It is used for convenience
  - *CourseID* and *StudentID* are foreign key properties

```
namespace ContosoUniversity.Models
{
    public enum Grade
    {
        A, B, C, D, F
    }

    public class Enrollment
    {
        public int EnrollmentID { get; set; }
        public int CourseID { get; set; }
        public int StudentID { get; set; }
        public Grade? Grade { get; set; }

        public Course Course { get; set; }
        public Student Student { get; set; }
    }
}
```

Model Relationships




```
namespace ContosoUniversity.Models
{
    public enum Grade
    {
        A, B, C, D, F
    }

    public class Enrollment
    {
        public int EnrollmentID { get; set; }
        public int CourseID { get; set; }
        public int StudentID { get; set; }
        public Grade? Grade { get; set; }

        public Course Course { get; set; }
        public Student Student { get; set; }
    }
}
```

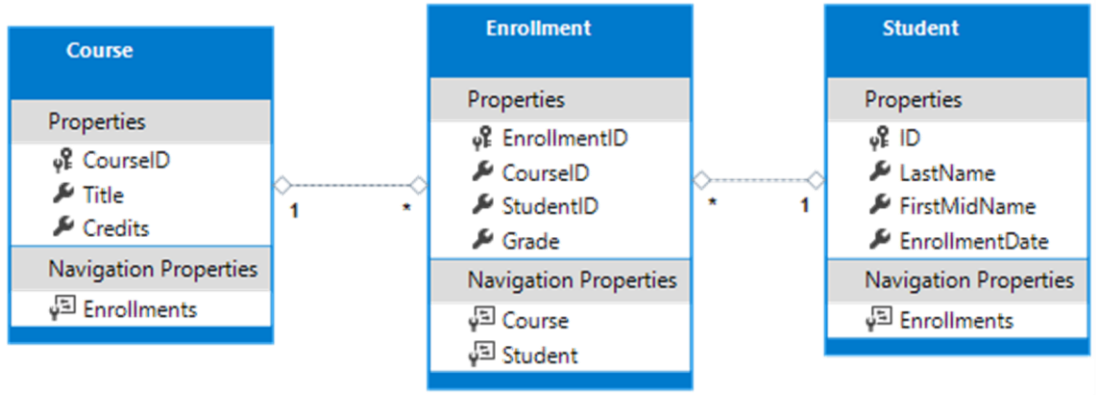
Model Relationships - DbContext

- The `DbContext` is how we expose our classes to EF
- Inherits from `Microsoft.EntityFrameworkCore.DbContext`
- It is also our gateway into the database in code

```
public class SchoolContext : DbContext
{
    public SchoolContext(DbContextOptions<SchoolContext> options) : base(options)
    {
    }

    public DbSet<Course> Courses { get; set; }
    public DbSet<Enrollment> Enrollments { get; set; }
    public DbSet<Student> Students { get; set; }
}
```

- `DbSet<T>` Where `T` is Class
  - How we tell EF which models to track relationships between



- Here we have told EF to track *Course*, *Enrollment*, and *Student* entities and their relationships

View Specific Models (DTOs)

- Data Transfer Objects (DTOs) or ViewModels can be used to create versions of your Entities that can be sent over the internet
- Prevents exposing schema information like relationships
- Creates more specific models for an application, like a flattened search result model
- Can be manually mapped or use a library like AutoMapper to move from entity to DTO and back

```
public class Enrollment
{
    5 references | 0 changes | 0 authors, 0 changes | 0 exceptions
    public int EnrollmentID { get; set; }
    12 references | 0 changes | 0 authors, 0 changes | 0 exceptions
    public int CourseID { get; set; }
    12 references | 0 changes | 0 authors, 0 changes | 0 exceptions
    public int StudentID { get; set; }
    9 references | 0 changes | 0 authors, 0 changes | 0 exceptions
    public Grade? Grade { get; set; }

    2 references | 0 changes | 0 authors, 0 changes | 0 exceptions
    public Course Course { get; set; }
    2 references | 0 changes | 0 authors, 0 changes | 0 exceptions
    public Student Student { get; set; }
}
```

View Specific Models (DTOs)

```
public class Enrollment
{
    5 references | 0 changes | 0 authors, 0 changes | 0 exceptions
    public int EnrollmentID { get; set; }
    12 references | 0 changes | 0 authors, 0 changes | 0 exceptions
    public int CourseID { get; set; }
    12 references | 0 changes | 0 authors, 0 changes | 0 exceptions
    public int StudentID { get; set; }
    9 references | 0 changes | 0 authors, 0 changes | 0 exceptions
    public Grade? Grade { get; set; }

    2 references | 0 changes | 0 authors, 0 changes | 0 exceptions
    public Course Course { get; set; }
    2 references | 0 changes | 0 authors, 0 changes | 0 exceptions
    public Student Student { get; set; }
}
```

```
public class EnrollmentDTO
{
    0 references | 0 changes | 0 authors, 0 changes | 0 exceptions
    public int EnrollmentID { get; set; }
    0 references | 0 changes | 0 authors, 0 changes | 0 exceptions
    public int StudentID { get; set; }
    0 references | 0 changes | 0 authors, 0 changes | 0 exceptions
    public string StudentLastName { get; set; }
    0 references | 0 changes | 0 authors, 0 changes | 0 exceptions
    public string StudentFirstName { get; set; }
    0 references | 0 changes | 0 authors, 0 changes | 0 exceptions
    public DateTime StudentEnrollmentDate { get; set; }
    0 references | 0 changes | 0 authors, 0 changes | 0 exceptions
    public int CourseID { get; set; }
    0 references | 0 changes | 0 authors, 0 changes | 0 exceptions
    public string CourseTitle { get; set; }
    0 references | 0 changes | 0 authors, 0 changes | 0 exceptions
    public int CourseCredits { get; set; }
    0 references | 0 changes | 0 authors, 0 changes | 0 exceptions
    public Grade? Grade { get; set; }
}
```

Entity Framework and ADO.NET

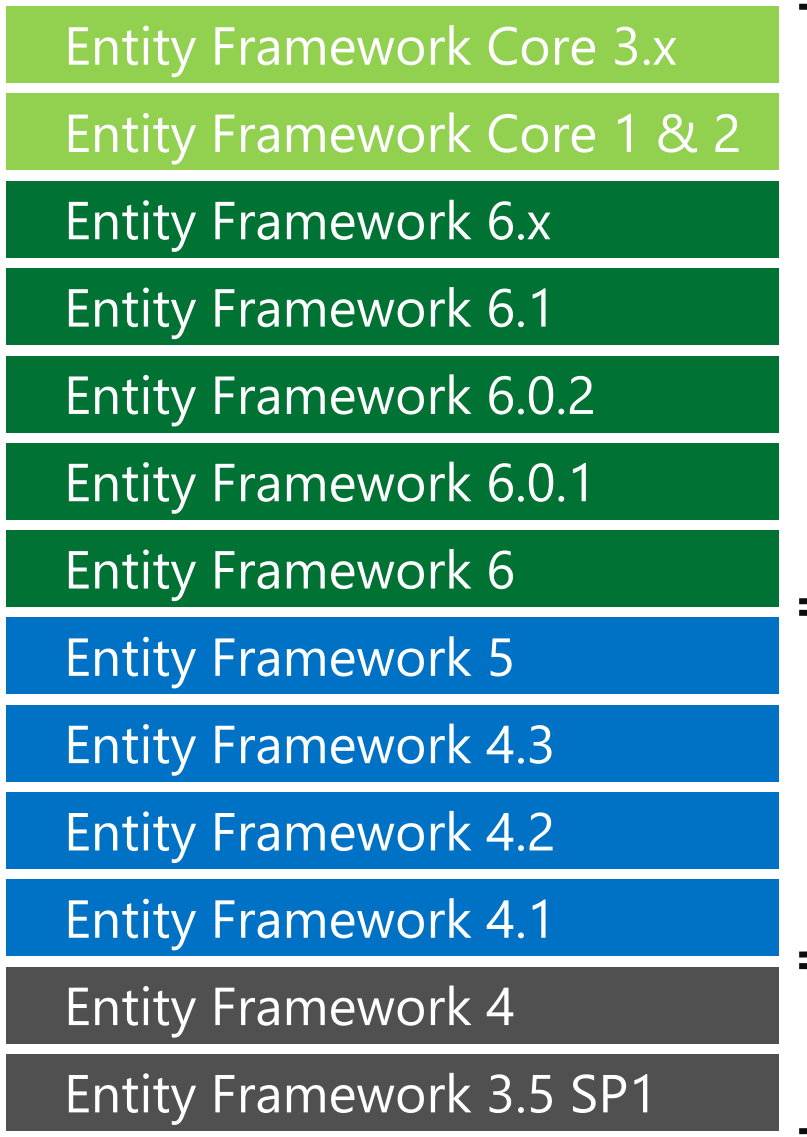
- EF build on top of ADO.NET since first EF version.
- EF generation T-SQL and than passing it to SQL Server via ADO.NET
- Code on ADO.NET is faster in a runtime, because you don’t need to generate t-sql
- Entity Framework code will take less time from developer
- To use ADO.NET you need to add NuGet package
  - `System.Data.SqlClient`

Demo: ADO.NET

What is Entity Framework Core?

- Light-weight and extensible version of Entity Framework
- Open Source
- New platforms
  - ASP.NET Core, Windows Store, etc.
  - Linux
  - Mac
- **Edmx is discontinued**

Out of Band | Release History



- Runtime out-of-band (NuGet)
- Tooling out-of-band
- Latest version “chained in” to new Microsoft Visual Studio releases and through SDK CLI

- Runtime
  - Core components in .NET
  - DbContext Application Programming Interface (API) and Code First out-of-band (NuGet)
    - Latest version “chained in” to new Visual Studio releases
- Tooling in Visual Studio

- Runtime in .NET Framework
- Tooling in Visual Studio

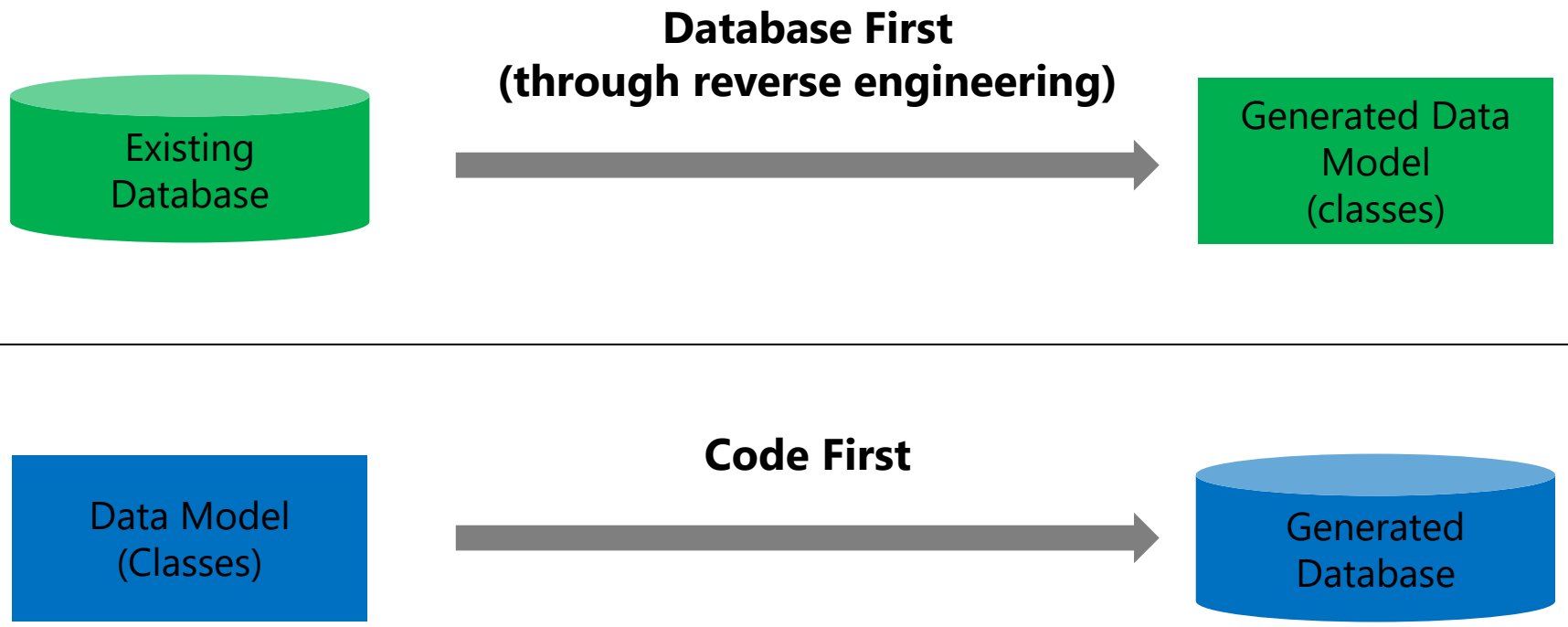
Module 3: Models

Section 3: Model Development

Lesson: Code-based Modeling

Entity Framework Core only supports
Code-based Modeling
(that is, Code First approach)

Entity Framework Development Approaches



Code-based modeling is the only approach supported in Entity Framework Core

Code-First Development

- Model code is written in .NET classes; model and database are created from the code
  - .NET Classes correspond to database tables
  - Properties correspond to database table columns
  - Classes can be used with or without EF!

- Relationships can be customized via the fluent API in the OnModelCreating override

- Code First can also work with existing database
  - Code is used for mapping instead of visual designer and XML

Code-First Approach: Pros and Cons

- Pros
  - Can realize complicated domain requirements
  - Can have a very clean and elegant domain model that is represented with Plain Old CLR Object (POCO)
  - Not tied with Entity Framework

- Cons
  - Needs more effort in creating domain classes
  - Needs to map the domain model to data model (with Data Annotations)

Tooling

- Entity Framework Core dotnet CLI – Our dev/design-time tooling

.NET Core CLI Copy

```
dotnet tool install --global dotnet-ef
```

- Enables `dotnet ef *` commands at the command line in the project directory, e.g.,
  - `dotnet ef migrations add Initial`
  - `dotnet ef database update Initial`
  - `dotnet ef dbcontext scaffold …`

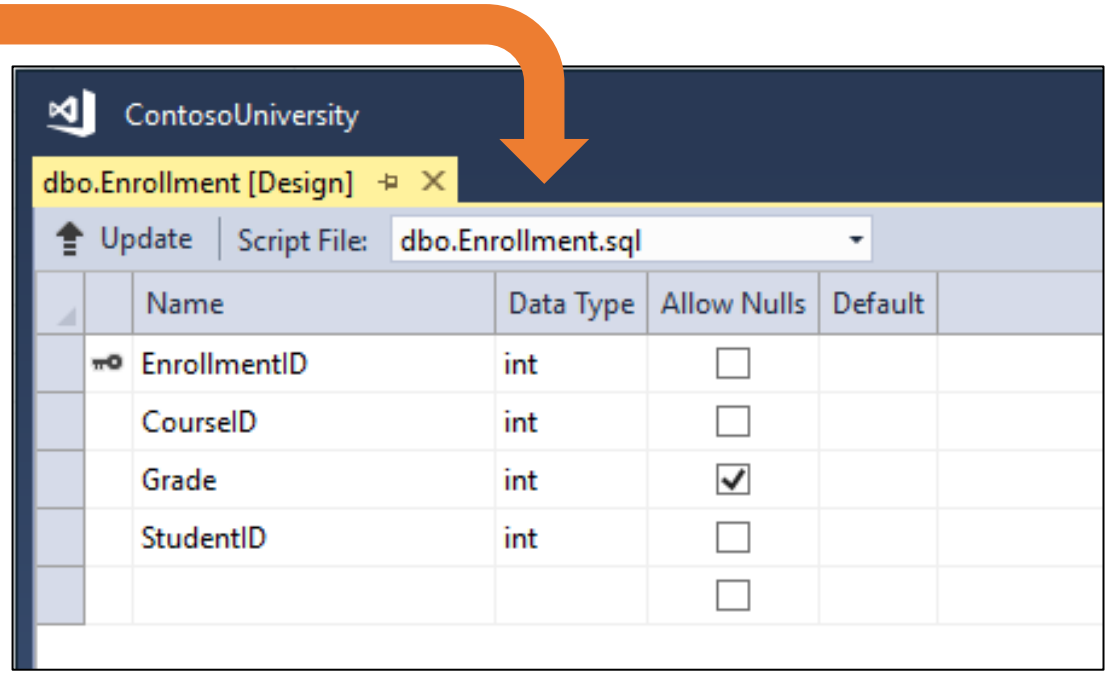
Note: version above may not be current, check docs when adding

Database Creation Using Entity Framework - I

- Model is created using .NET classes
- Executed using dotnet ef CLI tooling

```
    public class Enrollment
    {
        public int EnrollmentID { get; set; }
        public int CourseID { get; set; }
        public int StudentID { get; set; }
        public int? Grade { get; set; }

        public Course Course { get; set; }
        public Student Student { get; set; }
    }
}
```



Database Creation Using Entity Framework - II

- Tooling scans project for `Microsoft.EntityFrameworkCore.DbContext` based classes
- Contexts are used as the entry point into your code base
- We can override or reinforce how EF interprets relationships via the fluent API

```
public class SchoolContext : DbContext
{
    public SchoolContext(DbContextOptions<SchoolContext> options) : base(options)
    {
    }

    public DbSet<Course> Courses { get; set; }
    public DbSet<Enrollment> Enrollments { get; set; }
    public DbSet<Student> Students { get; set; }
}
```

For example, EF will create Students, Enrollments and Courses tables in the database

Database Seeding

- Database Initializers are deprecated in EF Core
  - Use DI to write and inject your own
- This code should be executed in `Main`, outside your app

```
public static class DbInitializer
{
    public static void Initialize(SchoolContext context)
    {
        context.Database.EnsureCreated();

        // Look for any students.
        if (context.Students.Any())
        {
            return;   // DB has been seeded
        }

        var students = new Student[]
        {
        new Student{FirstMidName="Carson",LastName="Alexander",EnrollmentDate=DateTime.Parse("2005-09-01")},
        new Student{FirstMidName="Meredith",LastName="Alonso",EnrollmentDate=DateTime.Parse("2002-09-01")},
        new Student{FirstMidName="Arturo",LastName="Anand",EnrollmentDate=DateTime.Parse("2003-09-01")},
        new Student{FirstMidName="Gytis",LastName="Barzdukas",EnrollmentDate=DateTime.Parse("2002-09-01")},
        new Student{FirstMidName="Yan",LastName="Li",EnrollmentDate=DateTime.Parse("2002-09-01")},
        new Student{FirstMidName="Peggy",LastName="Justice",EnrollmentDate=DateTime.Parse("2001-09-01")},
        new Student{FirstMidName="Laura",LastName="Norman",EnrollmentDate=DateTime.Parse("2003-09-01")},
        new Student{FirstMidName="Nino",LastName="Olivetto",EnrollmentDate=DateTime.Parse("2005-09-01")}
        };
        foreach (Student s in students)
        {
            context.Students.Add(s);
        }
        context.SaveChanges();
```

Database Seeding

- Database Initializers are deprecated in EF Core
  - Use DI to write and inject your own
- This code should be executed in `Main`, outside your app

Program.cs

```
public static void Main(string[] args)
{
    var host = BuildWebHost(args);

    using (var scope = host.Services.CreateScope())
    {
        var services = scope.ServiceProvider;
        try
        {
            var context = services.GetRequiredService<SchoolContext>();
            DbInitializer.Initialize(context);
        }
        catch (Exception ex)
        {
            var logger = services.GetRequiredService<ILogger<Program>>();
            logger.LogError(ex, "An error occurred while seeding the database.");
        }
    }

    host.Run();
}
```

Configuring Connections with Entity Framework

- Database connection string is typically stored in configuration (often appsettings.json)

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddDbContext<SchoolContext>(options =>
        options.UseSqlServer(Configuration.GetConnectionString("DefaultConnection")));

    services.AddMvc();
}
```

Startup.cs

```
{
  "ConnectionStrings": {
    "DefaultConnection": "Server=(localdb)\\mssqllocaldb;Database=ContosoUniversity1;Trusted_Connection=True;MultipleActiveResultSets=
  },
  "Logging": {
    "IncludeScopes": false,
    "LogLevel": {
      "Default": "Warning"
    }
  }
}
```

appsettings.json

Code First Migrations

- Enables changing the data model and deploying the change in production without dropping and re-creating the database
- Effective strategy for real-world production databases
- **Up** method used for creating/updating database schema
- **Down** method used for rollback logic
- Maintains version of each change
- Not required, but are very helpful if your schema changes

Migration Methods

Up Method

```
protected override void Up(MigrationBuilder migrationBuilder)
{
    migrationBuilder.CreateTable(
        name: "Course",
        columns: table => new
        {
            CourseID = table.Column<int>(type: "int", nullable: false),
            Credits = table.Column<int>(type: "int", nullable: false),
            Title = table.Column<string>(type: "nvarchar(max)", nullable: true)
        },
        constraints: table =>
        {
            table.PrimaryKey("PK_Course", x => x.CourseID);
        });

    migrationBuilder.CreateTable(
        name: "Student",
        columns: table => new
        {
            StudentID = table.Column<int>(type: "int", nullable: false)
                .Annotation("SqlServer:ValueGenerationStrategy", SqlServerValueGeneration
            EnrollmentDate = table.Column<DateTime>(type: "datetime2", nullable: false),
            FirstName = table.Column<string>(type: "nvarchar(max)", nullable: true),
            LastName = table.Column<string>(type: "nvarchar(max)", nullable: true)
        },
        constraints: table =>
```

Down Method

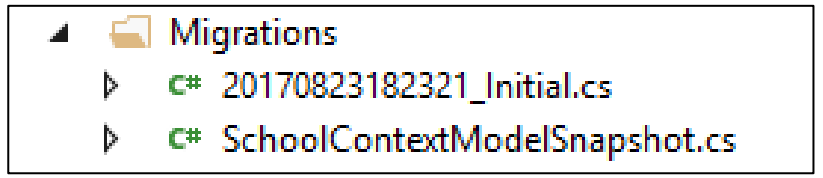
```
protected override void Down(MigrationBuilder migrationBuilder)
{
    migrationBuilder.DropTable(
        name: "Enrollment");

    migrationBuilder.DropTable(
        name: "Course");

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

Creating and Applying Migrations

```
$ dotnet ef migrations add Initial
```




```
$ dotnet ef database update Initial
```

```
info: Microsoft.EntityFrameworkCore.Migrations[200402]
      Applying migration '20170823182321_Initial'.
Applying migration '20170823182321_Initial'.
info: Microsoft.EntityFrameworkCore.Database.Command[200101]
      Executed DbCommand (1ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
      CREATE TABLE [Course] (
          [CourseID] int NOT NULL,
          [Credits] int NOT NULL,
          [Title] nvarchar(max) NULL,
          CONSTRAINT [PK_Course] PRIMARY KEY ([CourseID])
      );
info: Microsoft.EntityFrameworkCore.Database.Command[200101]
      Executed DbCommand (1ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
      CREATE TABLE [Student] (
          [StudentID] int NOT NULL IDENTITY,
          [EnrollmentDate] datetime2 NOT NULL,
          [FirstName] nvarchar(max) NULL,
          [LastName] nvarchar(max) NULL,
          CONSTRAINT [PK_Student] PRIMARY KEY ([StudentID])
      );
info: Microsoft.EntityFrameworkCore.Database.Command[200101]
      Executed DbCommand (2ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
      CREATE TABLE [Enrollment] (
          [EnrollmentID] int NOT NULL IDENTITY,
          [CourseID] int NOT NULL,
          [Grade] int NULL,
          [StudentID] int NOT NULL,
          CONSTRAINT [PK_Enrollment] PRIMARY KEY ([EnrollmentID]),
          CONSTRAINT [FK_Enrollment_Course_CourseID] FOREIGN KEY ([CourseID]) REFERENCES [Course] ([CourseID]) ON DELETE CASCADE,
          CONSTRAINT [FK_Enrollment_Student_StudentID] FOREIGN KEY ([StudentID]) REFERENCES [Student] ([StudentID]) ON DELETE CASCADE
      );
```

Entity Framework Tools & CLI

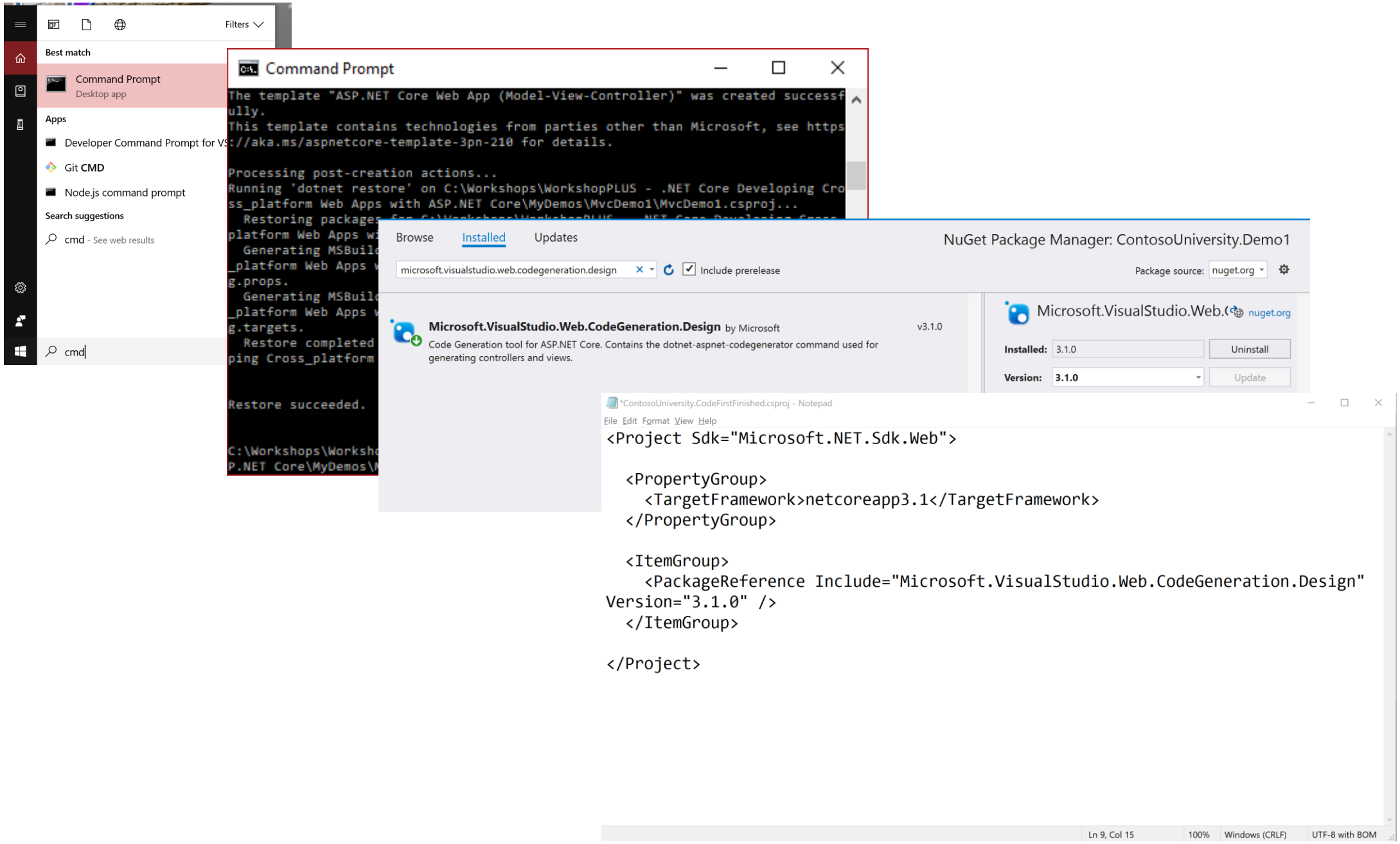
- Since .NET Core 3.0 the **dotnet ef** command line tool is no longer included in .NET Core SDK
- You must install it as a global or local tool using ollowing command installs tool globally

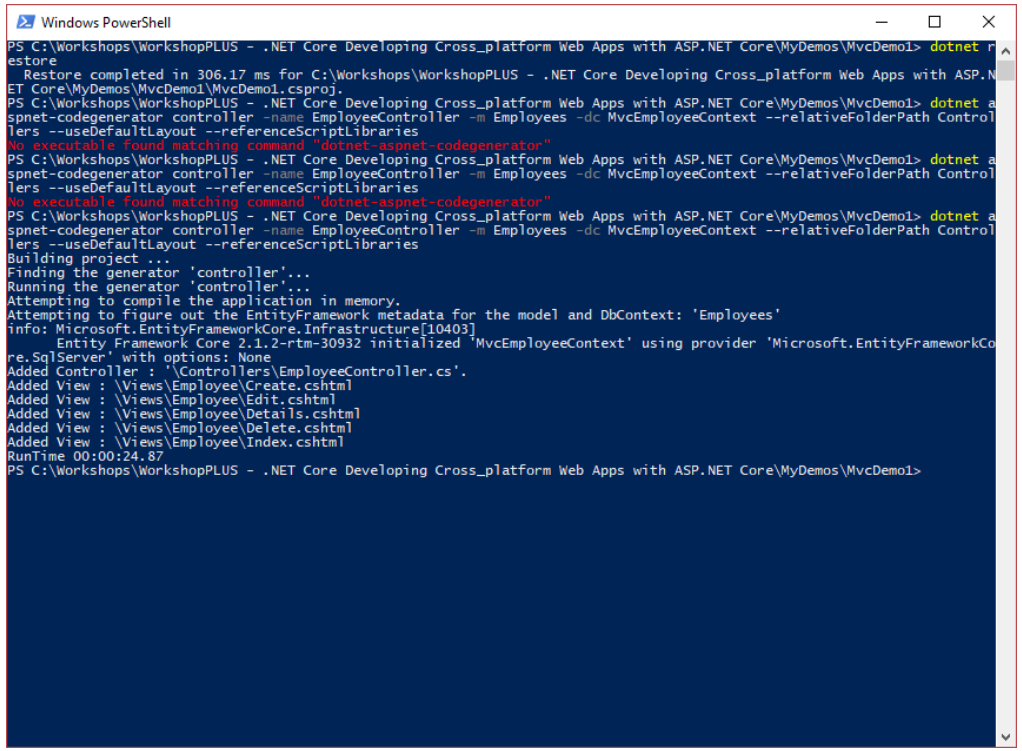
```
dotnet tool install --global dotnet-ef --version 3.1.0-*
```

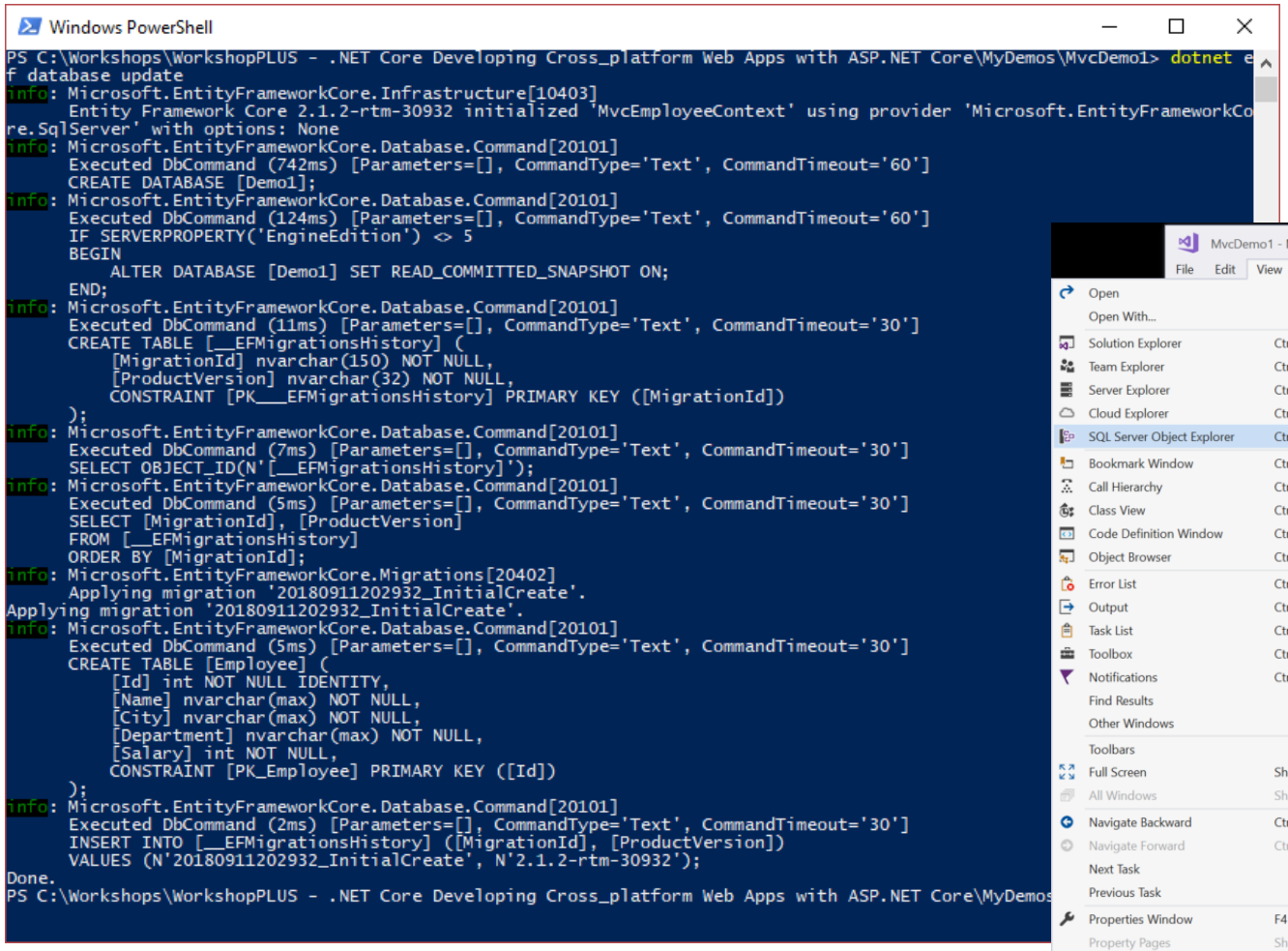
- Following command scaffolds all schemas and tables and put the new files in Models folder

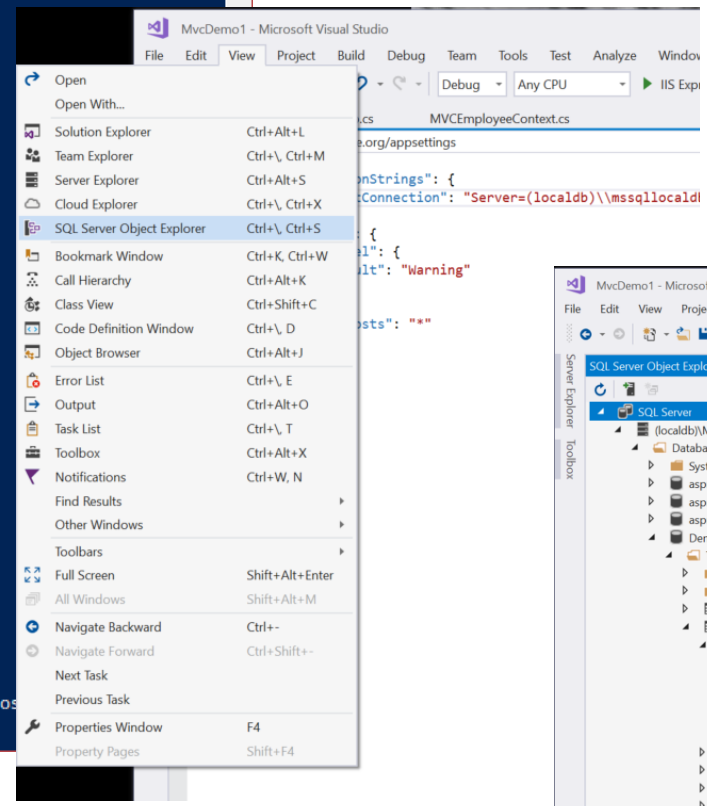
```
dotnet ef dbcontext scaffold
"Server=(localdb)\mssqllocaldb;Database=Blogging;Trusted_Connection=True;"
Microsoft.EntityFrameworkCore.SqlServer -o Models
```

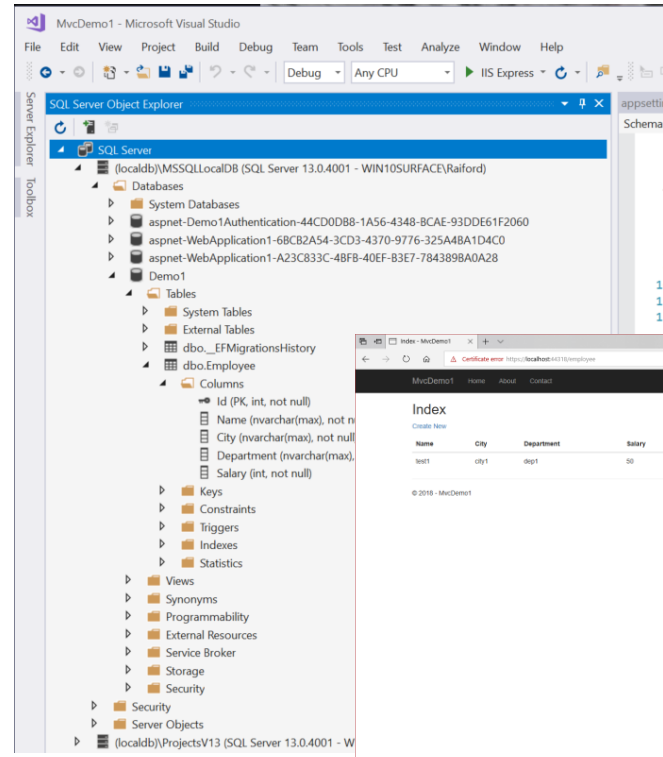
Demo: CRUD using EF

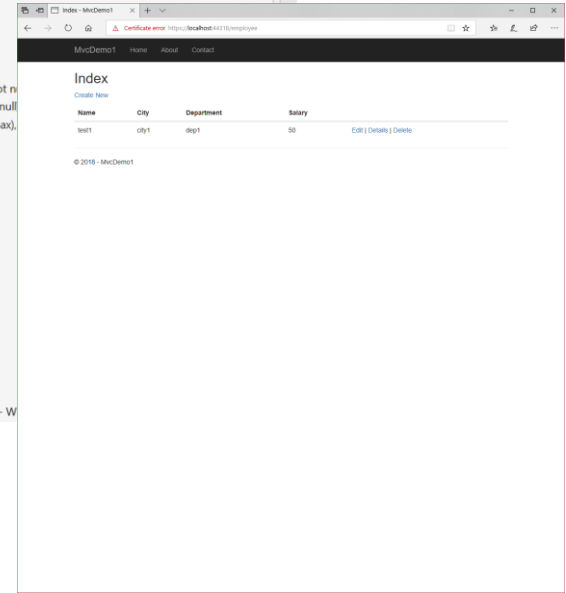

Filters
Best match
Command Prompt
Desktop app
Apps
Developer Command Prompt for VS
Git CMD
Node.js command prompt
Search suggestions
cmd - See web results
cmd
Command Prompt
The template "ASP.NET Core Web App (Model-View-Controller)" was created successf
ully.
This template contains technologies from parties other than Microsoft, see https
://aka.ms/aspnetcore-template-3pn-210 for details.
Processing post-creation actions...
Running 'dotnet restore' on C:\Workshops\WorkshopPLUS - .NET Core Developing Cro
ss_platform Web Apps with ASP.NET Core\MyDemos\MvcDemo1\MvcDemo1.csproj...
Restoring packages
platform Web Apps wi
Generating MSBuild
_platform Web Apps w
g.props.
Generating MSBuild
_platform Web Apps w
g.targets.
Restore completed
ping Cross_platform
Restore succeeded.
C:\Workshops\Worksho
P.NET Core\MyDemos\M
Browse
Installed
Updates
NuGet Package Manager: ContosoUniversity.Demo1
microsoft.visualstudio.web.codegeneration.design
Include prerelease
Package source:
nuget.org
Microsoft.VisualStudio.Web.CodeGeneration.Design by Microsoft
v3.1.0
Code Generation tool for ASP.NET Core. Contains the dotnet-aspnet-codegenerator command used for generating controllers and views.
Microsoft.VisualStudio.Web.C
nuget.org
Installed:
3.1.0
Uninstall
Version:
3.1.0
Update
*ContosoUniversity.CodeFirstFinished.csproj - Notepad
File Edit Format View Help
<Project Sdk="Microsoft.NET.Sdk.Web">
<PropertyGroup>
<TargetFramework>netcoreapp3.1</TargetFramework>
</PropertyGroup>
<ItemGroup>
<PackageReference Include="Microsoft.VisualStudio.Web.CodeGeneration.Design"
Version="3.1.0" />
</ItemGroup>
</Project>
Ln 9, Col 15
100%
Windows (CRLF)
UTF-8 with BOM


Windows PowerShell
PS C:\Workshops\WorkshopPLUS - .NET Core Developing Cross_platform Web Apps with ASP.NET Core\MyDemos\MvcDemo1> dotnet r
estore
Restore completed in 306.17 ms for C:\Workshops\WorkshopPLUS - .NET Core Developing Cross_platform Web Apps with ASP.N
ET Core\MyDemos\MvcDemo1\MvcDemo1.csproj.
PS C:\Workshops\WorkshopPLUS - .NET Core Developing Cross_platform Web Apps with ASP.NET Core\MyDemos\MvcDemo1> dotnet a
spnet-codegenerator controller -name EmployeeController -m Employees -dc MvcEmployeeContext --relativeFolderPath Control
lers --useDefaultLayout --referenceScriptLibraries
No executable found matching command "dotnet-aspnet-codegenerator"
PS C:\Workshops\WorkshopPLUS - .NET Core Developing Cross_platform Web Apps with ASP.NET Core\MyDemos\MvcDemo1> dotnet a
spnet-codegenerator controller -name EmployeeController -m Employees -dc MvcEmployeeContext --relativeFolderPath Control
lers --useDefaultLayout --referenceScriptLibraries
No executable found matching command "dotnet-aspnet-codegenerator"
PS C:\Workshops\WorkshopPLUS - .NET Core Developing Cross_platform Web Apps with ASP.NET Core\MyDemos\MvcDemo1> dotnet a
spnet-codegenerator controller -name EmployeeController -m Employees -dc MvcEmployeeContext --relativeFolderPath Control
lers --useDefaultLayout --referenceScriptLibraries
Building project ...
Finding the generator 'controller'...
Running the generator 'controller'...
Attempting to compile the application in memory.
Attempting to figure out the EntityFramework metadata for the model and DbContext: 'Employees'
info: Microsoft.EntityFrameworkCore.Infrastructure[10403]
Entity Framework Core 2.1.2-rtm-30932 initialized 'MvcEmployeeContext' using provider 'Microsoft.EntityFrameworkCo
re.SqlServer' with options: None
Added Controller : '\Controllers\EmployeeController.cs'.
Added View : \Views\Employee\Create.cshtml
Added View : \Views\Employee\Edit.cshtml
Added View : \Views\Employee\Details.cshtml
Added View : \Views\Employee\Delete.cshtml
Added View : \Views\Employee\Index.cshtml
RunTime 00:00:24.87
PS C:\Workshops\WorkshopPLUS - .NET Core Developing Cross_platform Web Apps with ASP.NET Core\MyDemos\MvcDemo1>


Windows PowerShell
PS C:\Workshops\WorkshopPLUS - .NET Core Developing Cross_platform Web Apps with ASP.NET Core\MyDemos\MvcDemo1> dotnet ef database update
info: Microsoft.EntityFrameworkCore.Infrastructure[10403]
Entity Framework Core 2.1.2-rtm-30932 initialized 'MvcEmployeeContext' using provider 'Microsoft.EntityFrameworkCore.SqlServer' with options: None
info: Microsoft.EntityFrameworkCore.Database.Command[20101]
Executed DbCommand (742ms) [Parameters=[], CommandType='Text', CommandTimeout='60']
CREATE DATABASE [Demo1];
info: Microsoft.EntityFrameworkCore.Database.Command[20101]
Executed DbCommand (124ms) [Parameters=[], CommandType='Text', CommandTimeout='60']
IF SERVERPROPERTY('EngineEdition') <> 5
BEGIN
ALTER DATABASE [Demo1] SET READ_COMMITTED_SNAPSHOT ON;
END;
info: Microsoft.EntityFrameworkCore.Database.Command[20101]
Executed DbCommand (11ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
CREATE TABLE [__EFMigrationsHistory] (
[MigrationId] nvarchar(150) NOT NULL,
[ProductVersion] nvarchar(32) NOT NULL,
CONSTRAINT [PK___EFMigrationsHistory] PRIMARY KEY ([MigrationId])
);
info: Microsoft.EntityFrameworkCore.Database.Command[20101]
Executed DbCommand (7ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
SELECT OBJECT_ID(N'[__EFMigrationsHistory]');
info: Microsoft.EntityFrameworkCore.Database.Command[20101]
Executed DbCommand (5ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
SELECT [MigrationId], [ProductVersion]
FROM [__EFMigrationsHistory]
ORDER BY [MigrationId];
info: Microsoft.EntityFrameworkCore.Migrations[20402]
Applying migration '20180911202932_InitialCreate'.
Applying migration '20180911202932_InitialCreate'.
info: Microsoft.EntityFrameworkCore.Database.Command[20101]
Executed DbCommand (5ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
CREATE TABLE [Employee] (
[Id] int NOT NULL IDENTITY,
[Name] nvarchar(max) NOT NULL,
[City] nvarchar(max) NOT NULL,
[Department] nvarchar(max) NOT NULL,
[Salary] int NOT NULL,
CONSTRAINT [PK_Employee] PRIMARY KEY ([Id])
);
info: Microsoft.EntityFrameworkCore.Database.Command[20101]
Executed DbCommand (2ms) [Parameters=[], CommandType='Text', CommandTimeout='30']
INSERT INTO [__EFMigrationsHistory] ([MigrationId], [ProductVersion])
VALUES (N'20180911202932_InitialCreate', N'2.1.2-rtm-30932');
Done.
PS C:\Workshops\WorkshopPLUS - .NET Core Developing Cross_platform Web Apps with ASP.NET Core\MyDemos


MvcDemo1 - Microsoft Visual Studio
File Edit View Project Build Debug Team Tools Test Analyze Window
Open
Open With...
Solution Explorer Ctrl+Alt+L
Team Explorer Ctrl+\, Ctrl+M
Server Explorer Ctrl+Alt+S
Cloud Explorer Ctrl+\, Ctrl+X
SQL Server Object Explorer Ctrl+\, Ctrl+S
Bookmark Window Ctrl+K, Ctrl+W
Call Hierarchy Ctrl+Alt+K
Class View Ctrl+Shift+C
Code Definition Window Ctrl+\, D
Object Browser Ctrl+Alt+J
Error List Ctrl+\, E
Output Ctrl+Alt+O
Task List Ctrl+\, T
Toolbox Ctrl+Alt+X
Notifications Ctrl+W, N
Find Results
Other Windows
Toolbars
Full Screen Shift+Alt+Enter
All Windows Shift+Alt+M
Navigate Backward Ctrl+-
Navigate Forward Ctrl+Shift+-
Next Task
Previous Task
Properties Window F4
Property Pages Shift+F4
Debug
Any CPU
MVCEmployeeContext.cs


MvcDemo1 - Microsoft Visual Studio
File Edit View Project Build Debug Team Tools Test Analyze Window Help
Debug
Any CPU
IIS Express
SQL Server Object Explorer
SQL Server
(localdb)\MSSQLLocalDB (SQL Server 13.0.4001 - WIN10SURFACE\Raiford)
Databases
System Databases
aspnet-Demo1Authentication-44CD0DB8-1A56-4348-BCAE-93DDE61F2060
aspnet-WebApplication1-6BCB2A54-3CD3-4370-9776-325A4BA1D4C0
aspnet-WebApplication1-A23C833C-4BF8-40EF-B3E7-784389BA0A28
Demo1
Tables
System Tables
External Tables
dbo.__EFMigrationsHistory
dbo.Employee
Columns
Id (PK, int, not null)
Name (nvarchar(max), not n
City (nvarchar(max), not null
Department (nvarchar(max),
Salary (int, not null)
Keys
Constraints
Triggers
Indexes
Statistics
Views
Synonyms
Programmability
External Resources
Service Broker
Storage
Security
Security
Server Objects
(localdb)\ProjectsV13 (SQL Server 13.0.4001 - W


Index
Create New
Name
City
Department
Salary
test1
city1
dep1
50
Edit | Details | Delete
© 2018 - MvcDemo1

Code First (Existing Database)

- Database schema reverse-engineered to Model classes
- Creates POCO classes
- POCO classes modified to customize database generation
- Corresponding partial classes used for customization
- Originally generated classes are replaced with each generation
- Indexes, functions, and stored procedures ignored

Module 3: Models

Section 4: Model Design

Lesson: Code First Development

Code-First Conventions - I

- Naming
  - Class Name or Object Type ➔ Table Name
- Primary Key
  - Property named 'Id' or '<class name>Id' ➔ Primary key value
    - Auto-increment is set for primary key values
- Relationship Inverses
  - Both types define *only one* navigation property
  - Product.Category and Category.Products represents different ends of the same relationship

```
public class Product
{
    public int ProductId { get; set; }
    public string Name { get; set; }
    public Category Category { get; set; }
}

public class Category
{
    public int CategoryId { get; set; }
    public string Name { get; set; }
    public ICollection<Product> Products { get; set; }
}
```

Code-First Conventions - II

- Type Discovery
  - Referenced object types are automatically included in the model without explicitly registering them as object sets

```
public class BookReview
{
    public int Id { get; set; }
    public Book Subject { get; set; }
    public string SubjectISBN { get; set; }
}

public class Book
{
    [Key]
    public string ISBN { get; set; }
    public string Name { get; set; }
    public ICollection<BookReview> Reviews { get; set; }
}
```

- Foreign Keys
  - Following conventions are used for foreign keys:

    <navigation property name><primary key property name>

    that is, '**SubjectISBN**';

    <principal class name><primary key property name>

    that is, '**BookISBN**';

    <primary key property name> that is, '**ISBN**';

- Code-First conventions can be overridden using **Data Annotations**, which can in turn be overridden using Fluent API

View-Specific Model

- It is a model that exists just to supply information to a view
- It is mostly used for views that show accumulated data from different tables
- It is also used to prevent “over-posting” attack

```
public class Review
{
    public int ReviewID { get; set; } // Primary key
    public int ProductID { get; set; } // Foreign key
    public Product Product { get; set; } // Foreign entity
    public string Name { get; set; }
    public string Comment { get; set; }
    public bool Approved { get; set; }
}
```

Model created to exclude *Approved* status

```
public class ReviewViewModel
{
    public string Name { get; set; }
    public string Comment { get; set; }
}
```

EF Core Fluent API

- Used inside of the `OnModelCreating` override in your DbContext
  - As of 2.0, can be defined in their own class and invoked inside `OnModelCreating`
- Can be used to override convention, explicitly define relationships, define custom conventions

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{

}
```

EF Core Fluent API – Shadow Properties

- A Shadow Property is a property that is *not* defined on your code class, but *does* exist *logically* on the entity (and in the datastore)
  - Example: a class with a navigation property, but no foreign key, *must* logically have a foreign key defined for the navigation property
- They can only be accessed via the EF API
- EF can implicitly create them, or you can explicitly define them via the fluent API

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Course>().Property<DateTime>("LastUpdated");
}
```

Create a shadow property

```
var query = _context.Courses
    .OrderBy(c => EF.Property<DateTime>(c, "LastUpdated"));
```

Use a shadow property in a LINQ query

```
_context.Entry(course).Property("LastUpdated").CurrentValue = DateTime.Now;
```

Get/Set the value of shadow property

EF Core Fluent API – Concurrency Token

- A Concurrency Token is the property on a model EF will use to determine if it has been modified since retrieved
- Timestamps are often used as concurrency tokens
- Can be marked via ConcurrencyCheckAnnotation or Fluent API
- Can be applied to Shadow Properties and regular properties

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Course>()
        .Property<byte[]>("Timestamp")
        .IsConcurrencyToken();
}
```

Create Shadow Property

Mark as Concurrency Token

EF Core Fluent API – Relationships

- Relationships can be explicitly defined using the fluent API

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Enrollment>(en =>
    {
        en.HasOne<Course>().WithMany(c => c.Enrollments).HasForeignKey(e => e.CourseID);
        en.HasOne<Student>().WithMany(c => c.Enrollments).HasForeignKey(e => e.StudentID);
    });
}
```

EF Core Fluent API – Conventions

- Default conventions can be overridden using the Fluent API

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    foreach (var entity in modelBuilder.Model.GetEntityTypes())
    {
        modelBuilder.Entity(entity.Name).ToTable($"tbl_{entity.ClrType.Name.ToLower()}");

        foreach (var property in entity.GetProperties().Where(p => p.ClrType == typeof(string)))
        {
            property.Relational().ColumnName = $"{property.Name.ToLower()}_str";
            property.SetMaxLength(200);
        }

        foreach (var fk in entity.GetForeignKeys())
        {
            fk.Relational().Name = fk.Relational().Name.ToLower();
        }
    }
}
```

EF Core Fluent API – Model-Level Query Filters

- Create a filter at the model level that filters all queries on that entity type

```
public class SchoolContext : DbContext
{
    0 references | 0 changes | 0 authors, 0 changes | 0 exceptions
    public SchoolContext(DbContextOptions<SchoolContext> options) : base(options) { }

    8 references | 0 changes | 0 authors, 0 changes | 0 exceptions
    public DbSet<Course> Courses { get; set; }
    7 references | 0 changes | 0 authors, 0 changes | 0 exceptions
    public DbSet<Enrollment> Enrollments { get; set; }
    2 references | 0 changes | 0 authors, 0 changes | 0 exceptions
    public DbSet<Student> Students { get; set; }

    private Guid TenantId;

    1 reference | 0 changes | 0 authors, 0 changes | 0 exceptions
    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Course>().HasQueryFilter(c => !c.IsDeleted && c.TenantId == this.TenantId);
    }
}
```

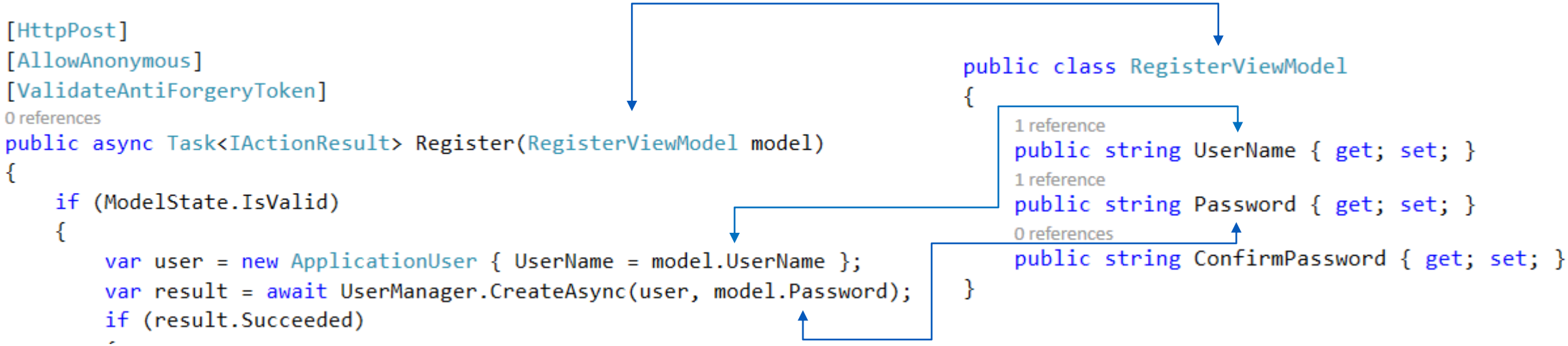
Module 3: Models

Section 3: Model Design

Lesson: Model Binding

Model Binding

- Model binder: Automatically maps posted form value to a .NET framework type based on naming conventions
- Default Model Binder is a default Model Binder implementation
  - Takes care of mundane property mapping and type conversion
  - Uses the name attribute of input elements
  - Automatically matches parameter names for simple data types
  - Complex objects are mapped by property name; use dotted notation



Async Query and Save

- What is it?
  - Task based async pattern for query and save
- Why did we build it?
  - Appropriate use of async can improve performance and scalability
- When should you use it?
  - Reduce server resource usage by freeing up blocked threads
  - Improve client UI responsiveness by not blocking main thread
  - Parallelism – but not on the same context instance

Model Development Strategies

- Strive for fat models and skinny controllers
  - Encapsulate logic in services
  - Private methods on controllers are a code smell
- Be careful of looping references and N+1 queries
  - Bad LINQ queries make bad T-SQL queries
  - Leverage data contracts, serialization configuration, or ViewModels/Data Transfer Objects to avoid looping references

Demo: Model binding

Module Summary

- In this module, you learnt about:
  - Model and its role in MVC pattern
  - Model development
  - Entity Framework Core
  - Scaffolding and scaffolding templates
  - Entity Framework development approaches
  - Code-first development and conventions
  - View-specific Model
  - Model binding and security
  - Model development Strategies

Lab: Models


Microsoft