Building a Data-driven ASP.NET Core Application with EF Core

Setting up a New ASP.NET Core Project with Entity Framework Core

Gill Cleeren
CTO Xebia Microsoft Services Belgium

@gillcleeren

Overview

Understanding the requirements

Looking at the starter solution

Understanding the concepts of EF Core

Adding and configuring EF Core in the application

Working with migrations

Version Check

This version was created by using:

- ASP.NET Core 8 and EF Core 8
- Visual Studio 2022

Version Check

This course is 100% applicable to:

- ASP.NET Core 6/7/8 and EF Core 6/7/8
- Visual Studio 2022 (any edition)

Understanding the Requirements

“Hi, I’m Bethany,
from Bethany’s Pie Shop!”

“Our site is doing great, sales are up!

I need a way to easily manage my products and see all orders placed in the online store.”

“We will need to create an administration site for Bethany, so she can manage the store herself.”

“We will use ASP.NET Core for this too
and we will rely on Entity Framework Core
to work with the data.

That’s a great combination!”

The Scenario:

Bethany's Pie Shop Admin Site

- Manage pies
- Manage categories
- See orders and their order details

We will rely on what we have covered in the “ASP.NET Core Fundamentals” course.

Demo

Exploring the finished application

Looking at the Starter Solution

Already included in the starter solution:

- Basic configuration of the ASP.NET Core application
- Layout and navigation menu
- Home page

Demo

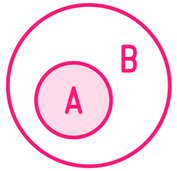
Looking at the starter solution

Understanding the Concepts of EF Core

Hello Entity Framework Core!

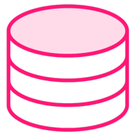
Entity Framework (EF) Core is a lightweight, extensible, open source and cross-platform version of the popular Entity Framework data access technology.
 - Microsoft Learn

Hello Entity Framework Core



Popular ORM, built-in to .NET

Eliminate need to write (most) SQL code



Support for several types of database engines

Code-first, although it can also work with existing database

Understanding the Concept of an ORM

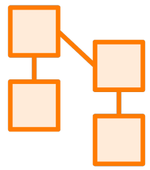
Class

```
public class Pie
{
    public int PieId { get; set; }
    public string Name { get; set; }
    public string Description { get; set; }
}
```

Table

PieId	Int (PK)
Name	nvarchar (max)
Description	nvarchar (max)

It All Starts with a Model



Set of classes (POCOs)

Context sits between application and database

Used to interact with data, not through handwritten query code

Identifying the Entities

Pie

Ingredient

Category

Order

OrderDetail

Exploring the Category Class

Notice the use of attributes on properties

```
public class Category
{
    public int CategoryId { get; set; }

    [Required]
    public string Name { get; set; } = string.Empty;

    [Required]
    public string? Description { get; set; }
}
```

Demo

Creating the model

Adding basic attributes on the properties

```
public class Category
{

    public int CategoryId { get; set; }

    [Display(Name = "Name")]
    public string Name { get; set; } = string.Empty;

    [Display(Name = "Description")]
    public string? Description { get; set; }

}
```

EF Core Uses Conventions

ID or <EntityName>ID becomes primary key
Created columns will match the property names

EF Core Uses Conventions

Using foreign keys

Category.cs

```
public class Category
{
    public int CategoryId
      { get; set; }

    public string Name
      { get; set; } = string.Empty;
}
```

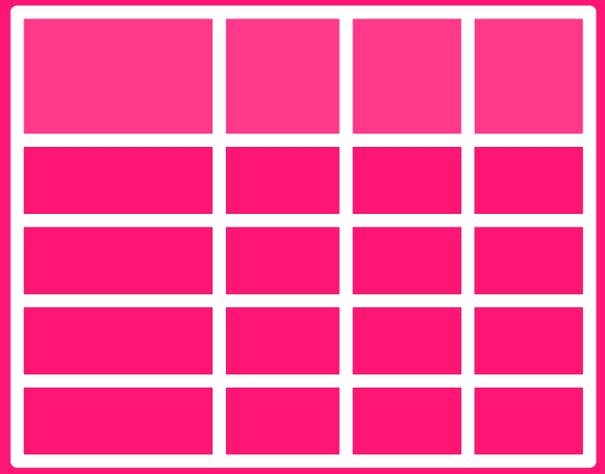
Pie.cs

```
public class Pie
{
    public int PieId
      { get; set; }

    public string Name
      { get; set; } = string.Empty;

    public int CategoryId
      { get; set; }

    public Category? Category
      { get; set; }
}
```

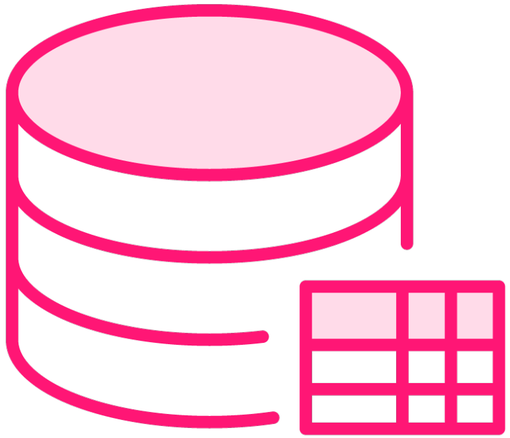


Column Data Types

Using a string in code will result in a nvarchar(max) being used by default

Non-nullable will become required column

Behavior can be influenced through fluent API or attributes



Adding the database context

- Class in application
- Represents session with database
- Allows for communication with actual database
  - Connection
  - Model building
  - Change tracking
  - Querying
  - …
- Derives from DbContext base class

Adding the Database Context

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

    public DbSet<Category> Categories { get; set; }
}
```

```
public class BethanysPieShopDbContext : DbContext
{

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {

        modelBuilder.Entity<Pie>().ToTable("Pie");

    }

}
```

Overriding the Created Tables

```
public class BethanysPieShopDbContext : DbContext
{

     protected override void OnModelCreating(ModelBuilder modelBuilder)
     {

          //configuration using Fluent API
          modelBuilder.Entity<Category>()
          .Property(b => b.Name)
          .IsRequired();
     }
}
```

Configuring the Mappings in OnModelCreating

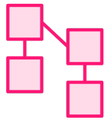
Using Fluent API or through attributes

Querying the Database using LINQ

```
_appDbContext.Pies.Where(p =>p.IsPieOfTheWeek);
```

Adding and Configuring EF Core in the Application

Steps to Add EF Core to the Application



Define the model classes - DONE

Add required packages to the project

Create the database context

Configure the application to use EF Core

Add connection string

Demo

Adding and configuring EF Core

Working with Migrations

Introducing Migrations

```
add-migration <nameOfMigration>
```

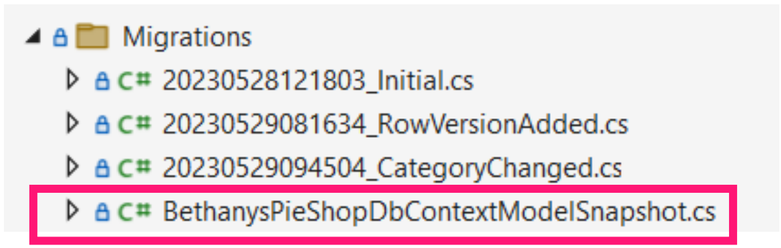
```
update-database
```

- Creates a new migration
- Brings the database to the level of the code

Looking at the Generated Migration

```
public partial class Initial : Migration
{
     protected override void Up(MigrationBuilder migrationBuilder)
     {
          migrationBuilder.CreateTable(
          name: "Categories",
          columns: table => new
          {
               CategoryId = table.Column<int>(type: "int", nullable: false)
                    .Annotation("SqlServer:Identity", "1, 1"),
               Name = table.Column<string>(type: "nvarchar(max)", nullable: false),
               Description = table.Column<string>(type: "nvarchar(max)", nullable: true),
               DateAdded = table.Column<DateTime>(type: "datetime2", nullable: false)
          },
          constraints: table =>
          {
               table.PrimaryKey("PK_Categories", x => x.CategoryId);
          });
     }
}
```

The Model Snapshot



Demo

Creating migrations

Demo

Seeding the database

Summary

EF Core is the default ORM in .NET

Integrates well with ASP.NET Core apps

Focus on code-first

DbContext is the link between the app and the database

Migrations are used to bring model and database in sync

Up Next:

Creating the List and Detail Pages